15
25

# UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES

CAMPUS ARAGÓN

# CONTROLADOR DE CONEXIONES PARA EL ACCESO A BASES DE DATOS

# T E S I S

QUE PARA OBTENER EL TITULO DE

INGENIERO EN COMPUTACION

P R E S E N T A :

MARTIN E. MOCTEZUMA HERNANDEZ

ASESOR DE TESIS:
LIC. ISRAEL JUAREZ ORTEGA

269995

MÉXICO 1999

UNIVERSIDAD NACIONAL AUTÓNOMA DE MEXICO

VNIVERSIDAD NACIONAL
AVTÓNOMA DE
MEXICO



*ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES ARAGÓN*

*JEFATURA DE INGENIERÍA EN COMPUTACIÓN*

*OFICIO ENAR/JACO/435/98*

***ASUNTO:*** *Asignación de jurado.*

***LIC. ALBERTO IBARRA ROSAS***
***Secretario Académico***
***P r e s e n t e .***

Por este conducto me permito presentar a usted, nombres de los Profesores que sugiero integren el ***Sínodo*** del ***Examen Profesional*** del alumno ***MOCTEZUMA HERNANDEZ MARTIN E.*** , *que* presenta el tema de tesis: ***"CONTROLADOR DE CONEXIONES PARA EL ACCESO A BASES DE DATOS."***

| | |
|---|---|
| ***PRESIDENTE:*** | ***ING. FRANCISCO ARISTA PATIÑO*** |
| ***VOCAL:*** | ***ING. ROBERTO BLANCO BAUTISTA*** |
| ***SECRETARIO:*** | ***ING. ISRAEL JUAREZ ORTEGA*** |
| ***SUPLENTE:*** | ***ING. RAUL CARBAJAL PINAL*** |
| ***SUPLENTE:*** | ***ING.ERNESTO PEÑALOZA ROMERO*** |

Quiero subrayar que la director de tesis es el **Ing. Israel Júarez Ortega** el cual está incluida con base en lo que reza el reglamento de Exámenes Profesionales de esta Escuela.

Sin otro particular, aprovecho la ocasión para enviarle un cordial saludo.

***A T E N T A M E N T E***
***"POR MI RAZA HABLARA EL ESPÍRITU"***
***San Juan de Aragón, Edo. de Méx., Noviembre 18 de 1998.***
***EL JEFE DE CARRERA***

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES ARAGÓN U. N. A. M. INGENIERIA EN COMPUTACION

***ING. JUAN GASTALDI PÉREZ***

***c.c.p.*** ***Lic. Ma. Teresa Luna Sánchez.- Jefa del Departamento de Servicios Escolares.***
***Ing. Israel Juarez Ortega.- Director de Tesis.***

***JGP/mav.***

Quiero expresar mi reconocimiento a las personas que de alguna manera con su apoyo, consejos y colaboración forman parte de esta tesis. En primer lugar a Dios, por la oportunidad que me ha dado para vivir y buscar el éxito. A mis padres, Eusebio y Clementina por su inmenso amor y apoyo en todo momento. A mi asesor, Israel Juárez por sus consejos y opiniones. A Arturo y Max, por sus consejos y ayuda. A Ivonne, por formar parte de mi vida. Y por supuesto a la Universidad Nacional Autónoma de México que me brindó un lugar en sus aulas.

A todos sinceramente muchas gracias.

Martín.

# INDICE

# INTRODUCCION

El continuo avance tecnológico en el área computacional, así como la exigencia de mas recursos dentro de dicha área, hacen necesario el planteamiento de nuevos modelos para el acceso a bases de datos.

Se tiene como fin, presentar los conocimientos y herramientas básicas para la conectividad entre un cliente y un servidor de datos teniendo un modelo diferente con un nuevo enfoque: el uso de sockets.

El contenido del presente trabajo trata sobre las partes mínimas necesarias para la implantación de dicho modelo. No es la realización de un sistema en particular, sino mas bien el desarrollo de un modelo el cual quiere presentarse como prototipo, y el cual podría aplicarse de manera general en el acceso a bases de datos dentro del área de desarrollo de sistemas.

La primera parte contiene los fundamentos necesarios para la implantación de este nuevo modelo:

El primer capítulo es una exposición básica de la arquitectura Cliente/Servidor.

El segundo capítulo expone que es ODBC, el porqué fue creado, en que consiste, así como los diferentes tipos de drivers y cómo usar los Data Source Names.

El tercer capítulo es la presentación del Objeto RDO, para el acceso a datos remotos, sus propiedades, métodos y eventos.

El cuarto capítulo describe que es un socket, como funciona y como podemos implantarlo en la presente investigación.

La segunda parte es la exposición, el desarrollo y la presentación final del modelo de acceso a bases de datos mediante sockets:

El quinto capítulo presenta el modelo actual de acceso a bases de datos y el modelo propuesto, así como sus respectivas ventajas y desventajas.

El sexto capítulo es el desarrollo del modelo propuesto, de cada una de las partes: el servidor y cliente de datos, así como sus respectivas propiedades, métodos y eventos. De igual forma este capítulo contiene la presentación final del nuevo modelo, y la manera como funciona cada una de las partes de dicho modelo: el acceso a datos en el servidor y la recuperación de datos desde el cliente de datos.

# 1. LA ARQUITECTURA CLIENTE/SERVIDOR

La arquitectura Cliente/Servidor es el diseño que está actualmente dominando todas las formas de la computación en redes, desde que la implementación de redes de área local (LAN) se ha convertido en el estándar en las áreas de cómputo, se ha buscado nuevas y mejores formas de compartir recursos.

La tendencia en la actualidad consiste en tener una separación bien definida de labores que realizan las máquinas que procesan información (servidores) y las que realizan la petición de información (clientes), esta es la característica principal de los sistemas Cliente/Servidor.

El auge de la arquitectura Cliente/Servidor es cada vez más notorio en casi todos los lugares donde se tienen equipos de cómputo, de hecho, muchos de los sistemas operativos se están rediseñando para aplicar este nuevo concepto, por ejemplo la versión 4.0 de Windows NT está basada en componentes los cuales, como se verá mas adelante son servidores de objetos.

Los sistemas actuales Cliente/Servidor pueden compartir recursos y datos, así como distribuir la responsabilidad de procesos tanto de software como hardware en el cliente y el servidor.

Para explicar esta arquitectura, se presenta una analogía con relación al proceso de negocio que se maneja en un restaurante, que en este caso será el sistema de red; donde el cliente solicita el servicio; el mesero es el enlace de comunicación entre el cliente y el cheff o servidor, quien es el encargado de proveer el servicio.

Al ordenar un platillo, el cliente solicita al mesero lo que desea comer. Después de haber hecho su petición el cliente ignora la forma de cómo prepararán el platillo, que ingredientes utilizarán o bien qué cheff específicamente lo cocinará, de tal manera que este proceso es transparente para el cliente.

Por su parte el mesero lleva las peticiones del cliente a manera de instrucciones u órdenes al cheff, y este las elabora junto con las peticiones de otros clientes. Al término del procesamiento y cuando los platillos están listos, el cheff (servidor) hace llegar la comida al cliente, nuevamente por medio del mesero, finalmente, el cliente recibe los platillos. El ciclo concluye cuando el cliente obtiene la respuesta concreta a su solicitud: los platillos cocinados listos para comer.

La arquitectura Cliente/Servidor consiste, como se puede ver en el ejemplo, en satisfacer necesidades de los clientes, sin que estos, estén enterados del procesamiento y transporte de la información que es llevado a cabo en el servidor. Esto permite transparencia en las aplicaciones y así mismo ciertas ventajas sobre la implantación de sistemas centralizados.

Existen diferentes definiciones de Cliente/Servidor, pero todas coinciden en una misma estructura:

"El modelo Cliente/Servidor es una formalización del procesamiento distribuido en el que una computadora actúa como un servidor de recursos y otras computadoras actúan como los clientes de los recursos"[1].

"La computación Cliente/Servidor es la relación entre dos procesos que son cooperativos en la ejecución de una misma tarea. El cliente requiere que alguna función se procese, el servidor procesa esta función, la función se refiere a un servicio"[2].

[1] DAY *Downsizing to Netware.* p 44.
[2] *Guide to building Client/Server solutions.* pp 1-2.

## 1.1 Estructura

Todos los sistemas Cliente/Servidor están divididos en tres partes: el cliente, el middleware y el servidor. El siguiente diagrama (figura 1.1) muestra el modelo general de un sistema Cliente/Servidor, cada sistema Cliente/Servidor puede esquematizarse mediante este diagrama, el cual puede ser usado como una herramienta conceptual para el entendimiento de la arquitectura Cliente/Servidor.

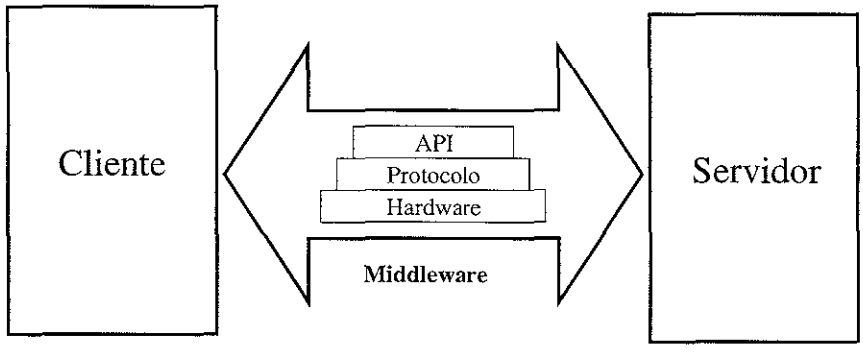



**Figura 1.1 El diagrama cliente-middleware-servidor**

*El cliente* típicamente se refiere a una entidad que hace uso de servicios de uno o varios servidores, e incluso de otro cliente mediante la ejecución de procesos remotos en el que funge como servidor. Dicho cliente frecuentemente usa una Interfaz Gráfica de Usuario (GUI por sus siglas en inglés: Graphical User Interfase).

*El Middleware* es la capa que comunica al cliente con el servidor y la cual es transparente en el diseño Cliente/Servidor. El middleware es comúnmente una combinación de las tres capas que se observan en la figura 1.1: hardware, protocolos y APIs.

*El servidor* es normalmente el programa o equipo de cómputo centralizado. Por lo regular existen mas de un cliente por cada servidor. Un servidor puede ser de diferentes tipos: servidor de archivos, servidor de base de datos, servidor de objetos, etc. Frecuentemente múltiples programas servidores se encuentran trabajando en un equipo de cómputo servidor, por ejemplo un servidor de correo electrónico y un servidor de base de datos pueden estar trabajando en un mismo equipo a un mismo tiempo.

Una característica importante del modelo Cliente/Servidor es la de poder distribuir la carga de trabajo en diferentes puntos, esto quiere decir que es posible separar los tres componentes básicos de las aplicaciones y repartirlos; así tanto los programas como los datos, el procesamiento, y el desplegado de la información puede distribuirse en diferentes equipos, de tal forma que se pueden utilizar al máximo las capacidades de todos los puntos de la red.

Estas tres partes de un sistema Cliente/Servidor pueden llegar a ser extremadamente complicadas, en muchos casos un mismo equipo contiene al servidor y al cliente, de cualquier forma, la arquitectura Cliente/Servidor permite la flexibilidad para realizar un diseño de acuerdo a las necesidades de cada sistema.

### *1.1.1 El Cliente*

Es la entidad por medio de la cual un usuario solicita un servicio, realiza una petición o demanda el uso de recursos. El cliente se encarga de realizar el *front end*, que es la parte de

la aplicación que interactúa con el usuario y que muchas veces se encuentra en un ambiente gráfico. El cliente oculta las partes más complejas de la interacción Cliente/Servidor presentando una interfaz sencilla para el usuario mediante el front end, dicho front end se desarrolla con herramientas de desarrollo de aplicaciones. Por ejemplo Delphi, Visual Basic, FoxPro, etc.

Algunas de las características del cliente son:

- Es el medio de enlace y/o comunicación entre el usuario y el equipo de cómputo usado.
- Es la entidad que requiere o solicita el servicio.
- Requiere el uso de los recursos del servidor para realizar su actividad.
- Es el medio por el cual se envía la solicitud y se reciben los resultados o notificaciones del servidor.
- Contiene la interfaz gráfica (GUI).
- Puede interactuar con uno o varios servidores.
- Es el responsable de mantener el diálogo con el usuario.

### *1.1.2 El interfaz Cliente/Servidor (Middleware)*

Es la infraestructura de software que permite la comunicación entre clientes y servidores, este componente permite:

- Establecer conexiones a recursos (bases de datos, correo electrónico, etc.)
- Procesar y acceder a la información
- Interactuar con las interfaces de usuario
- Mantener la seguridad, la auditoria y el control de versiones.

El middleware es la parte mas compleja de la interacción Cliente/Servidor. El término middleware es generalmente usado para referirse a la parte que habilita la comunicación entre el servidor y el cliente. El middleware está formado por una o varias capas de protocolos e interfaces. Por ejemplo, para comunicar un cliente de Visual Basic a una base de datos SQL Server, el middleware puede incluir WinSockets, el protocolo TCP/IP y un driver ODBC.

El middleware es transparente a la aplicación final y consiste en una serie de drivers o programas. A manera de ejemplo tenemos un caso real, una persona que funge como intérprete inglés/español puede decirse que está realizando la función de middleware dentro de una comunicación oral o escrita, dicho intérprete puede hablar ambos lenguajes permitiendo a una persona que hable inglés comunicarse con alguien que hable español. Si extendemos este ejemplo, supongamos que nuestro intérprete no habla español, solo habla inglés y francés, en este caso es necesario emplear otro intérprete que hable francés y español. Para realizar nuestra comunicación inglés/español ahora tenemos dos intérpretes (el cual es llamado "stacked middleware" o middleware apilado). Cada nuevo intérprete es "apilado" en el middleware.

El middleware habilita la efectiva comunicación entre diferentes tipos de software o hardware. Es difícil tener una comunicación con un dispositivo no común o que pertenece a otro género, este problema puede solucionarse mediante el uso del middleware, el cual nos provee de una interfaz que realiza la comunicación.

### *1.1.3 El Servidor*

El servidor es el conjunto de software y hardware que responde a los requerimientos de los clientes. El servidor provee de servicios: ejecuta procesamiento de datos, aplicaciones y

manejo de la información o recursos. En el servidor se realiza el *back end*, que es la parte destinada a recibir las solicitudes del cliente y donde se ejecutan dichas solicitudes. Por ejemplo, si se tiene un servidor de base datos el back end bien podría ser una base de datos como Informix, Oracle o cualquier otra.

Algunas de sus características son:

- Responde a las peticiones de los clientes.
- Tiene una gran capacidad de procesamiento y almacenamiento.
- Provee el uso de los recursos y servicios a los clientes.
- Tiene capacidad de procesamiento transaccional.
- Puede fungir como cliente de otros servidores.
- Contiene los datos, programas, información, así como las reglas y estándares de la aplicación.
- Realiza procesos como acceso, organización, almacenamiento, y manejo de datos.
- Comparte recursos.

El servidor usualmente maneja la parte compleja de la aplicación: tareas complejas y con un alto nivel de cálculos. Un servidor raramente tiene una sofisticada interfaz de usuario, porque solo tiene interacción con el usuario cuando se realizan tareas como instalación, configuración y monitoreo.

Por lo regular los servidores siempre tienen una gran capacidad de memoria RAM, esto con el fin de tener una capacidad de cálculo rápida. Poseen de igual forma una gran capacidad de almacenamiento porque es requerido el almacenamiento centralizado de datos e información. Actualmente los servidores a nivel hardware están siendo fabricados con múltiples procesadores. Si estos procesadores pueden trabajar juntos mediante la coordinación del sistema operativo, entonces el sistema es llamado Sistema de Multiprocesamiento Simétrico (SMP Symmetrical Multiprocessing System). Servidores

como de bases de datos que toman ventaja de la tecnología SMP son llamados *Servidores Escalables.*

## 1.2 Evolución

La implementación de sistemas Cliente/Servidor puede confundir a cualquier desarrollador de sistemas, es frecuente escuchar gente que describe un sistema en particular como "no es un verdadero sistema Cliente/Servidor". La razón para argumentar cuando se trata de un sistema Cliente/Servidor es histórica.

El procesamiento de la información ha evolucionado de tal manera que se han ido presentando algunas formas y fenómenos particulares para el tratamiento de la misma hasta llegar al modelo Cliente/Servidor, en la figura 1.2 se describe como es que se origina este modelo.

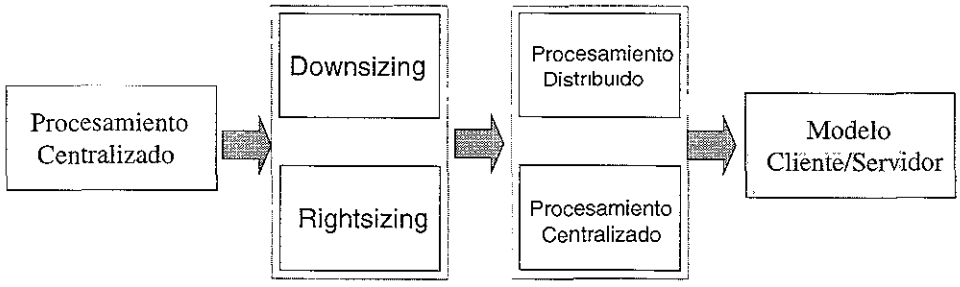



**Figura 1.2 Antecedentes del modelo Cliente/Servidor**

El primer modelo de procesamiento es el centralizado, basado en mainframe y es el que procesa la información de todos los usuarios, posteriormente surge el *downsizing*. Downsizing significa la reestructuración de los sistemas de cómputo donde se reemplazan los mainframes y minicomputadoras por servidores y estaciones de trabajo basados en redes de área local y, el *rightsizing* (que significa la reevaluación de los recursos de cómputo para mover los sistemas de información a través de mainframes y redes de área local basadas en computadoras personales independientes a sistemas distribuidos).

Con ello surge el modelo distribuido en el que se dividen el proceso y los datos entre varios servidores distribuidos en diferentes zonas geográficas para dar servicio a varios clientes que ya pueden trabajar independientemente y son PCs o estaciones de trabajo. El proceso cooperativo permite que los sistemas operativos de los clientes puedan interactuar con el de la red. Aquí surge el modelo Cliente/Servidor donde una aplicación se divide en un Front end y un Back end para aprovechar de manera óptima la capacidad de cada elemento.

La definición de sistemas Cliente/Servidor ha sido envuelta lentamente de nuevos productos y estándares que se van agregando. La definición de la arquitectura Cliente/Servidor se expande a medida que los sistemas van evolucionando.

### *1.2.1 Servidores de archivos*

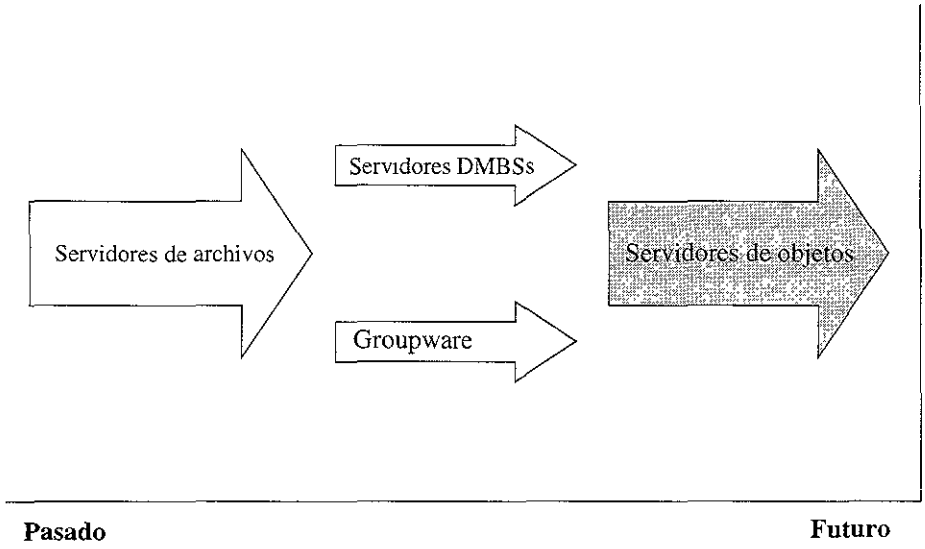



**Figura 1.3 Una linea de tiempo en la evolución de los servidores**

Como se puede observar en la figura 1.3, el concepto Cliente/Servidor ha ido evolucionando.

Inicialmente todos los servidores consistían en servidores de archivos, lo cual permitía compartir archivos a otros usuarios desde un servidor central, por ejemplo con la plataforma DOS esto significa acceder a archivos que se encuentran en un drive virtual el cual es añadido al sistema.

Al compartir archivos, un servidor puede correr en un equipo servidor central y permitir el acceso a sus directorios y archivos. Cuando el usuario requiere una copia de un archivo, el servidor lo envía a la máquina del cliente.

### *1.2.2 Servidores de bases de datos*

A medida que se ha ido generalizando el uso de bases de datos multiusuario y existen servidores –a nivel hardware- cada vez más poderosos, las compañías de software han ido desarrollando mejores programas servidores de bases de datos.

Los servidores de bases de datos corren un programa servidor que administra las tablas de la base de datos en un formato común, todo esto almacenado en la máquina servidor. El cliente envía una petición (una búsqueda o una modificación por ejemplo) al servidor de la base de datos, y el servidor ejecuta la petición. El tipo más común de servidores de bases de datos es el basado en el lenguaje SQL, aunque existen otros que utilizan un diseño propietario, por ejemplo los lenguajes Xbase.

### *1.2.3 Servidores de Groupware*

Con el advenimiento de servidores de bases de datos también han llegado las aplicaciones de grupo (Groupware Applications). El groupware es una categoría no muy generalizada en el campo Cliente/Servidor que describe programas que coordinan grupos de documentos o información en una red. Por ejemplo, si un grupo de usuarios está trabajando de manera conjunta para la publicación de un libro, pueden ser requeridos muchos tipos de documentos, y el grupo podría necesitar compartir la información. Usando groupware el equipo de trabajo puede hacer uso de la red para compartir todos los documentos comunes al equipo (archivos CorelDRAW, MS Word, CAD, etc.) aunque éstos estén en diferentes máquinas.

Actualmente la mayoría de las aplicaciones de groupware son de *documentos centralizados*, estas aplicaciones coordinan archivos almacenados en diferentes equipos en la red.

### *1.2.4 Servidores de objetos*

Muchos especialistas en el mercado de la industria de cómputo concuerdan en decir que los servidores de objetos vendrán a ser el tipo dominante de servidores. Un servidor de objetos trabaja con datos, figuras, objetos de bases de datos, y procedimientos de una manera muy similar entre si. Un cliente puede solicitar una copia o ejecución de un objeto en particular desde el servidor. El objeto contiene la información de si mismo, así, cada objeto "sabe" como responder a una instrucción o petición.

Mientras ésta es una explicación muy sencilla, la intercambiabilidad de objetos es la fortaleza de los servidores de objetos. Un simple servidor de objetos puede manipular servicios multimedia (audio y video), accesos a bases de datos, compartimiento de archivos, correo electrónico, rutinas y cualquier otra pieza de datos la cual pueda ser encapsulada dentro de un objeto.

El mundo de los estándares para objetos puede ser extremadamente complejo, estos estándares están basados en modelos de objetos que definen el formato para el objeto.

Los servidores de objetos no se han popularizado por completo, debido al tráfico en la red que con su implementación se origina. Estos objetos también requieren el desarrollo de ambientes y sistemas operativos que interactuen con ellos.

### *1.2.5 Otros servidores*

Existen algunos otros tipos de servidores disponibles, incluyendo servidores de reportes, de correo, de fechas, de video, de imágenes, de audio, y otros. La mayoría de estos servidores son altamente especializados o actúan como funciones integradas a otros servidores, es decir, son servidores de funciones especializadas que forman parte de un servidor-aplicación global.

## 1.3 División de tareas

"El cliente y el servidor están diseñados a fin de dividir el trabajo y procesamiento de una aplicación"[3]. Por ejemplo, un servidor es idóneo para el procesamiento intenso de información cuando un número muy grande de recursos compartidos tiene que estar levantado en un equipo servidor. Un cliente, por otro lado, puede ejecutar la mayoría de las necesidades específicas de los usuarios. Por ejemplo, realizar una gráfica a partir de una colección de datos.

La división de tareas proporcionada por un sistema Cliente/Servidor es la gran ventaja sobre el procesamiento centralizado. Esto equivale a tener parte del procesamiento de la aplicación completa en la máquina de cada usuario, aunque el requerimiento de procesos especializados y la centralización de datos permanece. La metodología Cliente/Servidor requiere que las responsabilidades sean divididas entre las máquinas del cliente y del servidor.

### *1.3.1 Responsabilidades del servidor*

El servidor mantiene los datos centralizados y responde a los requerimientos de los clientes. Como los datos están centralizados en el servidor, el servidor es tradicionalmente el responsable del respaldo completo de datos y su almacenamiento, el cual puede ser robustecido mediante el uso de una unidad de cinta -el método más común- o alguna otra utilería calendarizada.

Un servidor puede usar la replicación para asegurar la seguridad de sus datos. La replicación permite el acceso a varios lugares donde hay datos para una sincronización y coordinación de la información. Esto permite tener un "espejo" de la información la cual puede estar resguardada en algún lugar geográfico diferente.

El servidor también es el responsable de la coordinación de datos. Últimamente se ha ido incrementando el uso de procedimientos almacenados (stored procedures) en servidores. Los procedimientos almacenados son rutinas que permanecen de manera nativa en el servidor y que pueden ser ejecutados cuando se presenta alguna condición en especial dentro de la base de datos o simplemente cuando son llamados desde un cliente.

Se ha ido generalizando el interés de poner las reglas del negocio a manera de procedimientos almacenados en un servidor central. Por ejemplo, en un sistema tradicional de punto de venta, si una venta en particular es mas alta que el monto de crédito especificado (por ejemplo $5000 pesos) entonces el límite del cliente tiene que ser verificado. Tradicionalmente cada programa cliente verifica que la venta total no sobrepase los $5000 pesos. Si en algún momento se decide que el monto a verificar sea a partir de $ 7000 pesos, todos los programas cliente tienen que ser modificados.

[3] RAHMEL. *Client Server applications with Visual Basic 4*. p. 34.

En este ejemplo, si las reglas del negocio son puestas en línea y pueden ser llamadas desde el servidor para verificar el límite, entonces las reglas del negocio son lo único que es necesario ser cambiado. Un procedimiento almacenado puede ser activado de manera automática cuando una condición en particular ocurre.

### *1.3.2 Responsabilidades del cliente*

El programa cliente es el responsable de la integridad de los datos que son obtenidos desde el servidor. Muchos programas cliente que acceden datos de misión crítica no tienen privilegios de escritura en la base de datos del servidor. Por razones de seguridad e integridad, la menor cantidad de clientes posible tienen acceso a modificar la información de la base de datos. Muchos clientes son simplemente construidos para realizar búsquedas en la base de datos (*queries*).

En las modificaciones enviadas desde el cliente al servidor, la integridad de datos es vitalmente importante. Por ejemplo, si una venta consiste en 10 artículos, y la conexión Cliente/Servidor es rota después de que el sexto artículo fue modificado, el diseño Cliente/Servidor necesita tener la habilidad de reestablecer la transacción o en su caso abortar por completo la transacción en el orden en el que fue hecha (del primer al sexto artículo). La comunicación Cliente/Servidor tiene que ser monitoreada y controlada para asegurar la precisión de los datos. Para asegurar dicha precisión, los servidores de bases de datos usan un método conocido como *transacción plana*, la cual permite al cliente encapsular una transacción que se encuentra entre los comandos `Begin Transaction` (inicio de transacción) y `Commit Transaction` (fin de transacción). Si alguna de las partes de la transacción falla todos los cambios son deshechos desde donde la transacción comenzó.

Existen dos estructuras comunes para un sistema Cliente/Servidor: sistemas de apoyo a toma de decisiones y sistemas de procesamiento de transacciones en línea (OLTP online transaction processing).

La más común es la estructura de apoyo a toma de decisiones. Los sistemas así diseñados obtienen información periódica (supongamos cada quince minutos) de otros sistemas independientes. Mediante la información obtenida, la persona que analiza esta información puede generar un sumario de la información procesada y tomar un juicio hacia una dirección correcta del negocio.

El procesamiento de transacciones en línea (OLTP) es usado en sistemas día con día. Ellos cuentan con una respuesta instantánea (o por lo menos de unos cuantos segundos) y tienen que ser construidos con este objetivo. Un sistema de punto de venta es un excelente ejemplo. Cuando la persona de ventas está vendiendo mercancía en una tienda, el artículo es dado de alta y el precio aparece de forma automática en la pantalla. Entonces los artículos son deducidos del inventario de la tienda. La respuesta de la computadora tiene que ser instantánea y exacta.

## 1.4 Ventajas y desventajas de la arquitectura Cliente/Servidor

La implementación del modelo Cliente/Servidor presenta una serie de características que pueden considerarse como ventajas comparadas con los modelos tradicionales de cómputo. Sin embargo también presenta desventajas.

*Ventajas*:

- **Permite un mejor aprovechamiento de la potencia de cómputo** de los equipos al poder descargar en el cliente parte de la carga de trabajo del servidor.

- **Permite el acceso de un cliente a varios servidores en forma simultánea**, lo que resulta cada vez mas importante en los ambientes heterogéneos actuales.

- **Reduce el tráfico en la red**, ya que solo viajan los requerimientos y las atenciones a ellos.

- **Puede y, de hecho, opera bajo sistemas abiertos**, lo que significa su capacidad de cambio a plataformas con un mínimo de problemas.

- **Permite el uso de interfaces gráficas** de usuario muy versátiles y amigables en los clientes.

- **Aumento de la productividad** y disminución del costo y tiempos de entrenamiento de usuarios en el uso de estaciones de trabajo con interfaz gráfica.

- **Integración de aplicaciones** desarrolladas internamente con programas comerciales.

- **Flexibilidad** en la organización al distribuir datos y aplicaciones en la red, ponerlos mas cerca de los usuarios y poder relocalizar funciones sin necesidad de modificar la aplicación.

- **Mejor aprovechamiento de los recursos de cómputo**, ya que cada tarea se ejecuta en la plataforma mas adecuada.

- **Acceso a la información** cuando y desde donde el usuario la necesite.

- **Facilidad de modificar la operación** de la organización sin necesidad de hacer grandes inversiones y largos planes.

- **Flexibilidad de uso** y migración de plataformas de hardware, herramientas de desarrollo y sistemas operativos.

*Desventajas:*

- **Las aplicaciones Cliente/Servidor suelen ser más complejas** que las tradicionales en la forma de anfitrión/invitado, y exigen más de la red. En ambos casos, las desventajas pueden ser superadas mediante las ventajas anotadas, y ciertamente, todo tiene su costo.

- **Seguridad.** Los expertos en seguridad informática mantienen, que bajo Cliente/Servidor la seguridad es más sencilla de romper. Pero la respuesta a esta aseveración es que la seguridad está sujeta al diseño y medidas de seguridad que se incorporen al ambiente, y a las aplicaciones desarrolladas para operar bajo la arquitectura Cliente/Servidor.

- **Administración.** La administración del servicio de proveedores externos (equipos, programas comerciales, desarrolladores de aplicaciones) puede ser más compleja al implementarse las plataformas de software y hardware disponibles no propietarios, y aunado a la novedad de las tecnologías involucradas el control de calidad del servicio se dificulta.

- **Conflictos.** El cambio organizacional podría general conflictos en las áreas involucradas al tratar de deshacerse nichos de poder y posiciones políticas. La común resistencia al cambio es uno de los principales factores a considerar.

# 2. ODBC

## 2.1 Introducción

La conectividad a bases de datos (ODBC Open Database Connectivity) es un estándar abierto para la conexión datos, que por lo regular son bases de datos bien definidas (DBMS Database Management System) -por ejemplo SQL Server- aunque también se puede acceder con ODBC otras fuentes de datos, por ejemplo archivos Excel o archivos DBF; por tal motivo denominaremos *fuente de datos* a cualquier archivo que contenga información que pueda ser manejable.

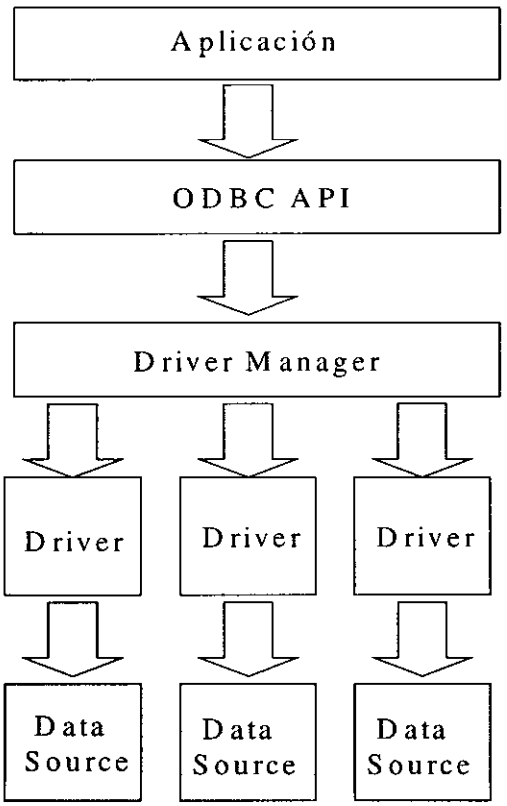



**Figura 2.1 Una visión global de ODBC**

ODBC está diseñado para la máxima *interoperabilidad*, que es, "la habilidad que posee una sola aplicación para acceder a diferentes fuentes de datos usando el mismo código fuente"[4].

Estas aplicaciones llaman a fuentes de datos a través de *drivers*. Un driver ODBC es un archivo DLL que es usado de acuerdo a una fuente de datos en particular y a la cual se desea acceder. Por ejemplo, un driver ODBC para Sybase puede ser usado para acceder a un servidor Sybase.

Como se puede ver en la figura 2.1, una aplicación accede a una fuente de datos (que por lo regular es una base de datos aunque puede ser un archivo Excel o incluso un archivo de texto) vía ODBC haciendo uso de manejadores ODBC los cuales denominaremos drivers.

El uso de drivers separa la aplicación de la fuente de datos de la misma forma que los drivers para impresoras separan los programas procesadores de texto de la impresora que se desee usar. Esto es debido a que los drivers son cargados en memoria en tiempo de ejecución, un usuario solo tiene que adicionar un nuevo driver si desea acceder a una nueva fuente de datos, y no es es necesario modificar el código o recompilar la aplicación.

### *2.1.1 ¿ Porqué fue creado ODBC ?*

Inicialmente, el acceso a bases de datos fue hecho mediante un front end diseñado especialmente para cada base de datos o a través de aplicaciones y/o comandos diseñados para trabajar de manera exclusiva con dicha base de datos. De esta forma, el uso de las computadoras creció y empezó a haber mayor diversidad de software y hardware, y empezaron a desarrollarse y venderse diferentes DBMSs. Las razones para adquirir tal o cual DBMS dependían de cuestiones tales como el precio, la rapidez en el acceso a datos,

[4] *Microsoft ODBC 3.0 Software Development Kit and Programmers Reference.* Vol 1 pp. 9

cual era mejor manejada por su gente de sistemas, cual era la última existente en el mercado, o cual trabajaba mejor como aplicación. Así mismo empezaron a existir diferentes tipos de bases de datos en las empresas porque, las que empezaron con un DBMS en particular, de pronto tuvieron mas de un DBMS.

Este crecimiento de DBMSs se hizo más complejo con la aparición de las computadoras personales. Estas computadoras acarrearon consigo todo un conjunto de herramientas para realizar búsquedas, análisis y el despliegue de los datos, junto con un gran número de bases de datos económicas y fáciles de usar. A partir de entonces una sola empresa frecuentemente tenía todo un ejército de servidores y minicomputadoras así como una gran cantidad de información almacenada en una gran variedad de bases de datos incompatibles, información que era accedida por un vasto número de diferentes herramientas.

El reto final vino con el advenimiento de la computación Cliente/Servidor, la cual busca hacer más eficiente el uso de los recursos de cómputo. Computadoras personales (clientes) proveen tanto front ends gráficos como herramientas comunes tales como hojas de cálculo, gráficas de datos, y reporteadores. Las minicomputadoras y mainframes (los servidores) contienen las DBMSs las cuales hacen uso de los recursos del servidor para trabajar de manera óptima. ¿Cómo se logró que los programas de front end se pudieran conectar a las bases de datos del back end?

Un problema similar sucedió con los vendedores de software. Los vendedores de software realizaron herramientas de explotación de datos para cada DBMS. Esto originó un gran gasto de recursos desarrollando y dando mantenimiento a las rutinas de acceso para cada DBMS.

Así, nació la necesidad de acceder a diferentes fuentes de datos desde una misma aplicación. Se necesitaba una forma interoperable de acceder a datos. Se necesitaba la conectividad abierta a bases de datos.

### *2.1.2 ¿ Qué es ODBC ?*

Existen muchas concepciones erróneas acerca de ODBC. Para usuario final, ODBC representa un icono en el Panel de Control, para un programador, es una librería que contiene rutinas para el acceso a datos, y para muchos otros, ODBC representa la respuesta a los problemas de acceso a fuentes de datos.

Principalmente, ODBC es una plataforma API (llamada ODBC API), Este API es independiente a cualquier fuente de datos o sistema operativo. ODBC API es un conjunto de funciones y procedimientos que están contenidas dentro de librerías (DLLs), estas librerías pueden ser usadas por cualquier desarrollador de aplicaciones para acceder a fuentes de datos. Usando ODBC API, se puede realizar una aplicación que se conecte a una fuente de datos, que realice una búsqueda, y regrese los datos al cliente (la misma aplicación) y finalmente cierre la conexión a la fuente de datos.

En lugar de que un programador tenga que hacer uso de diferentes APIs y lenguajes de acceso a bases de datos propietarios, ODBC provee un API universal el cual puede ser usado para acceder a diferentes fuentes de datos heterogéneas.

ODBC, además de sus librerías API, ofrece toda una serie de componentes los cuales se ven mas adelante, y mediante los cuales se puede desarrollar una aplicación Cliente/Servidor que acceda a cualquier fuente de datos.

Aquí solo se hace mención de la existencia de ODBC API, debido a que, por su facilidad para el desarrollo de aplicaciones usaremos el modelo de objetos Remote Data Object, el cual contiene en sí mismo –encapsula- las funciones ODBC API y el cual se verá de manera detallada en el siguiente capítulo.

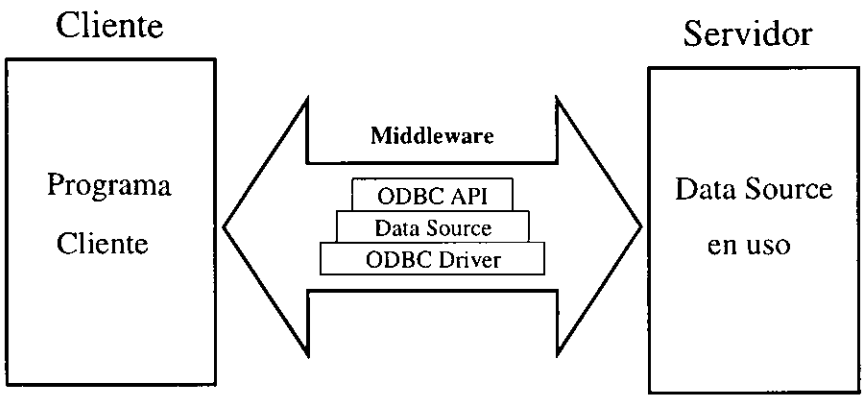



**Figura 2.2 Un diagrama con la arquitectura Cliente/Servidor ODBC**

Regresando a la definición de lo que ODBC significa, podemos decir que el estándar ODBC día a día es mas aceptado en el mundo de la computación. Dicho estándar fue creado por Microsoft y grupos independientes de desarrolladores. ODBC actúa como un middleware entre una aplicación y cualquier fuente de datos mediante un apropiado driver. (Ver figura 2.2).

ODBC ofrece ciertas capacidades tales como

- Transparencia en el sistema de tablas, ya que las tablas son automáticamente modificadas en su información por el driver.

- Cursores movibles (en una o ambas direcciones, dependiendo del tipo de fuente de datos).
- Creación de transacciones.
- Acceso a la fuente de datos de forma asíncrona.
- Manejo dinámico de la información.
- Uso de procedimientos almacenados (stored procedures) para bases de datos que soporten SQL.

ODBC es un estándar basado en SQL que realiza sus transacciones y búsquedas a través de comandos SQL. Estos comandos son provistos mediante el uso de funciones, stored procedures, y tipos de datos complejos (como `date`, `time`, `timestamp` o `binary`).

Los componentes ODBC son los siguientes, y son explicados de manera mas detallada en el apartado 2.2.

- **El driver manager.** Carga y descarga los drivers requeridos por la aplicación. Procesa las llamadas de las funciones ODBC o las pasa de manera directa al driver.
- **Los Drivers.** Procesan las llamadas de las funciones ODBC, envía las sentencias SQL a la fuente de datos, y regresan los resultados a la aplicación. Si es necesario, el driver modifica la sintaxis de las sentencias SQL a fin de que puedan ser entendidas por la fuente de datos.
- **Data Sources.** Un data source contiene los datos necesarios para acceder a una fuente de datos específica, por ejemplo Oracle Server o SQL Server.

Así, de manera general, ODBC está compuesto de archivos DLL, archivos .INI y aplicaciones ejecutables. ODBC de manera general está organizado mediante el uso de *data sources*. Un data source puede ser configurado (creado, borrado o modificado) mediante el adminstrador de ODBC. Una aplicación cliente llama a la interfaz ODBC haciendo uso del

nombre dado a un data source en particular, entonces el sistema ODBC carga el driver apropiado e intenta conectarse a la fuente de datos.

### *2.1.3 ¿ Cómo funciona ODBC ?*

Se puede desarrollar una aplicación que haga uso de ODBC, dicha aplicación podrá acceder a cualquier tipo de fuente de datos siempre y cuando dicha fuente tenga un driver disponible. Al tener una aplicación que halla sido desarrollada usando la interfaz ODBC, pueden ser accedidos diferentes tipos de fuentes de datos sin ninguna necesidad de modificar el código de los programas.

ODBC trabaja haciendo conexiones a fuentes de datos. Cada conexión define un data source y -por lo regular- contiene el nombre default de la base de datos a cargar. La conexión provee la comunicación entre las peticiones SQL y la fuente de datos.

Si la conexión es exitosa, las peticiones pueden ser hechas haciendo uso de ODBC API o algún método diferente (por ejemplo usando RDO, que como se mencionó, se explica en el siguiente capítulo) para requerir información desde la fuente de datos, modificaciones o llamadas a stored procedures. Cada petición realizada es convertida por el driver en un protocolo que la fuente de datos pueda entender y usar. La información enviada desde la fuente de datos es trasladada y manipulada a través del sistema ODBC y regresada a la aplicación cliente en un formato estándar.

El sistema ODBC acepta comandos normales del tipo SQL. Dichos comandos son preparados por el driver de tal forma que la fuente de datos los pueda entender de forma nativa, cabe destacar que cuando la fuente de datos usa de igual forma sentencias SQL la conversión requerida driver/fuente de datos es mínima, cosa que no sucede cuando la fuente

de datos utiliza un lenguaje diferente a SQL, en este caso la conversión requiere de más procesamiento.

## 2.2 Arquitectura

La arquitectura ODBC tiene cuatro componentes: Aplicaciones, el Driver Manager, los drivers y los data sources.

La figura 2.3 muestra la relación entre estos cuatro componentes.

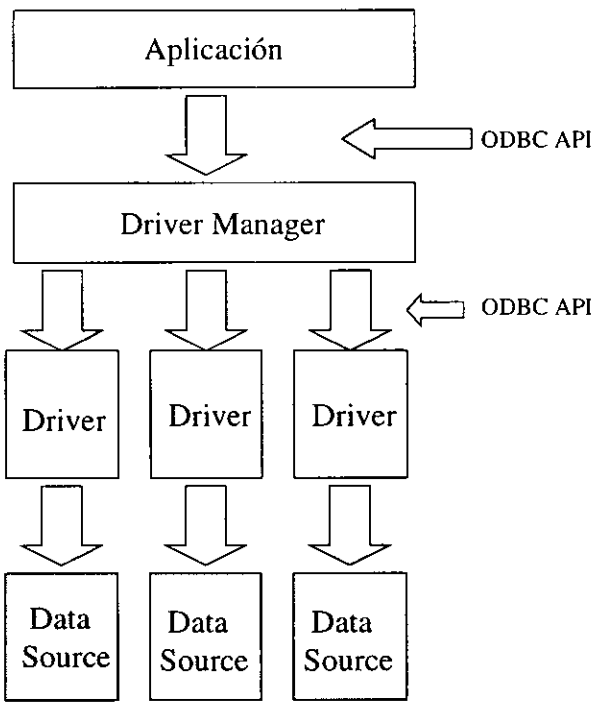



**Figura 2.3 Arquitectura ODBC**

- **Aplicaciones.** Ejecuta procesos y hace llamadas a las funciones ODBC mediante el envío de instrucciones (sentencias SQL) y la obtención de resultados.

- **El driver manager.** Carga y descarga los drivers requeridos por la aplicación. Procesa las llamadas de las funciones ODBC o las pasa de manera directa al driver.

- **Driver.** Procesa las llamadas de las funciones ODBC, envía las sentencias SQL a la fuente de datos, y regresa los resultados a la aplicación. Si es necesario, el driver modifica la sintaxis de las sentencias SQL a fin de que puedan ser entendidas por la fuente de datos.

- **Data Source.** Un data source contiene los datos necesarios para acceder a una fuente de datos específica

Nótese lo siguiente acerca de la figura. Pueden existir múltiples drivers y data sources, lo cual permite que una aplicación pueda, de manera simultanea acceder a diferentes fuentes de datos.

### *2.2.1 Aplicaciones*

Una aplicación es un programa que, mediante ODBC accede a fuentes de datos. Aunque puede haber muchos tipos de aplicaciones posibles, la mayoría cae en tres categorías básicas:

- **Aplicaciones genéricas.** Las aplicaciones genéricas son diseñadas para trabajar con una gran variedad de diferentes DBMSs. Por ejemplo las hojas de cálculo para generar alguna estadística, las cuales usan ODBC para importar datos los cuales serán

posteriormente analizados, o por ejemplo, algún procesador de texto que use ODBC para obtener la libreta de direcciones de correo electrónico de alguna base de datos.

Lo que toda aplicación genérica tiene en común es que, es altamente interoperable entre DBMSs y que necesita usar ODBC de una forma relativamente genérica.

- **Aplicaciones verticales.** Las aplicaciones verticales ejecutan un simple tipo de tarea, como dar de alta artículos a un inventario o obtener reportes de existencias, este tipo de aplicaciones trabajan con un esquema de base de datos que es controlado por el desarrollador de la aplicación. Para una unidad de negocio pequeña podría hacer uso de una fuente de datos simple como dBASE, mientras que una unidad más grande podría hacer uso de Oracle.

  La aplicación usa ODBC de tal forma que no está atada a una fuente de datos en particular y funciona de igual forma con cualquier fuente de datos, siempre y cuando se conserve el mismo esquema de base de datos. De esta forma, el desarrollador de la aplicación puede vender esta aplicación de manera independiente de la fuente de datos. Las aplicaciones verticales son interoperables cuando son desarrolladas, pero en ocasiones son modificadas de acuerdo a las necesidades del negocio, en donde se les incluye funciones no interoperables una vez que el usuario a elegido una fuente de datos en particular.

- **Aplicaciones personalizadas.** Las aplicaciones personalizadas son construidas para realizar una tarea específica en una compañía en particular. Por ejemplo, una compañía con muchas divisiones departamentales puede tener diferentes DBMSs en cada departamento, aunque al final solo se obtenga un reporte único general. ODBC es usado en este caso por su interfaz de comandos común, lo cual facilita el trabajo a los programadores que tienen que acceder a diferentes DBMSs. Estas aplicaciones

generalmente no son interoperables y son diseñadas para una fuente de datos y un driver en particular.

Hay un número de tareas que son comunes a cualquier aplicación, no importa como ellas hagan uso de ODBC. Así, el flujo de cualquier aplicación es:

- Seleccionar un data source y conectarse a él.
- Enviar una petición mediante sentencias SQL.
- Obtener los resultados de la petición.
- Procesar errores.
- Dar fin a la transacción (`commit`) o deshacer la transacción (`rollback`).
- Desconectarse del data source

El data source, no es la fuente de datos en sí. Es el registro que apunta al lugar donde la fuente de datos se encuentra.

Debido a que la mayor parte de las tareas realizadas en el acceso a fuentes de datos son sentencias SQL, la tarea principal de las aplicaciones que usan ODBC es enviar sentencias SQL y obtener los resultados (si existen) generados por estas sentencias. Otras tareas que una aplicación puede realizar comúnmente son, obtener información general de la fuente de datos, tales como su estructura de tablas, reglas, defaults, etc.

### *2.2.2 El Driver Manager*

El driver manager es una librería que se encarga de administrar la comunicación entre las aplicaciones y los drivers.

El driver manager fue construido principalmente como una ventaja para los desarrolladores de aplicaciones, el driver manager resuelve una cantidad de problemas común a cualquier aplicación. Entre estos problemas se encuentran determinar cual driver se necesita cargar para una fuente de datos específica, la carga y descarga de los drivers, y la llamada a las funciones de los drivers.

Consideremos que pasaría si una aplicación llamara directamente a las funciones del driver, mientras la aplicación estuvo ligada directamente a un driver en particular, la aplicación tendría que construir una tabla de apuntadores a las funciones del driver y llamar dichas funciones a través del puntero. Si usáramos el mismo código para usar mas de un driver esto podría originar un nivel de complejidad. La aplicación tendría primero que establecer un puntero para las funciones adecuadas en el driver correcto, y entonces llamar a la función que se necesita a través del puntero.

El driver manager resuelve este problema por medio de la provisión de un solo lugar para llamar cada función. La aplicación es ligada al driver manager, entonces llama a las funciones ODBC del driver manager y no directamente a las funciones del driver. La aplicación identifica el driver objetivo y el data source a través de un *handle* de la conexión. Cuando la aplicación carga (por medio del driver manager) un driver, el driver manager construye una tabla de punteros a las funciones del driver. El driver manager usa el handle de la conexión el cual lo recibe a través de la conexión a fin de encontrar la dirección de la función en el driver y llamar esta función a través de dicha dirección.

Por la mayor parte, el driver manager solo pasa las llamadas de las funciones desde la aplicación al driver. De cualquier forma, el driver manager también implementa algunas funciones (SQLDataSources, SQLDrivers y SQLGetFunctions) y ejecuta el chequeo básico de las ejecuciones. Por ejemplo, el driver manager checa que los handles no hagan referencia a un puntero nulo, que las funciones sean llamadas en el orden correcto y que los argumentos pasados a las funciones sean válidos.

El rol final -y el mayor del driver manager- es la carga y descarga de drivers. La aplicación carga y descarga solo el driver manager, cuando la aplicación quiere usar un driver en particular solo llama a una función de conexión (SQLConnect, SQLDriverConnect o SQLBrowseConnect) del driver manager y especifica el nombre en particular de un data source, por ejemplo "Clientes" o "SQL Server". Por medio este nombre, el driver manager busca la información que el data source contiene acerca del nombre y la ubicación del driver a usar, por ejemplo SQLSRVR.DLL. El driver manager carga entonces el driver (asumiendo que no ha sido cargado), guarda la dirección de cada una de las funciones del driver y llama a la función de conexión, la cual se inicializa a sí misma y se conecta a la fuente de datos.

Cuando la aplicación ha terminado de usar el driver, hace una llamada a la función SQLDisconnect en el driver manager. El driver manager hace la llamada de esta función en el driver a su vez, donde el driver se desconecta de la fuente de datos. El driver manager mantiene el driver en memoria en caso de que sea necesaria una reconexión. El driver manager descarga al driver solo cuando la aplicación libera la conexión usada por el driver o usa la conexión para un driver diferente y no hay más conexiones para el driver usado.

### *2.2.3 Los drivers*

Los drivers son librerías que implementan las funciones en el ODBC API. Cada uno es construido para trabajar con una fuente de datos en particular. Por ejemplo un driver para Oracle no puede acceder a datos de un DBMS de Informix. Los drivers exponen las capacidades de los DBMSs.

Los drivers son específicos para cada fuente de datos. Estos drivers generalmente también son provistos por la compañía que desarrolla la fuente de datos. Por ejemplo, Microsoft

provee los drivers para sus bases de datos SQL Server y Access, así como para otras fuentes de datos que no son precisamente bases de datos tales como archivos DBF o Excel. Así mismo Informix provee drivers para sus bases de datos en sus versiones Standard Enginee, OnLine y New Era. Y en general existen drivers casi para todo tipo de fuentes de datos.

**Tareas de los drivers.**

- Conectarse y desconectarse de un data source.
- Buscar errores no verificados por el driver manager.
- Inicializar transacciones; ésto es transparente a la aplicación.
- Enviar sentencias SQL a la fuente de datos para su ejecución. El driver debe transformar (si es necesario) la sentencia SQL de ODBC a una sintaxis reconocida por la fuente de datos.
- Enviar y recibir datos procedentes del data source, incluyendo la conversión de los datos (en caso de ser necesario) de acuerdo a las especificaciones de la aplicación.

**Arquitectura de los drivers.**

La arquitectura de los drivers cae en dos categorías dependiendo sobre que tipo de software se procesen las sentencias SQL.

- **Drivers File-based.** El driver accede físicamente a los datos de manera directa. En este caso, el driver actua como driver y como data source, esto es, el driver procesa las llamadas a ODBC y los comandos SQL. Por ejemplo, los drivers para dBASE son File-based porque dBASE no provee un mecanismo de gestión de base de datos como tal que el driver pueda usar. Es importante hacer notar que los desarrolladores de drivers tipo File-based tienen que desarrollar sus propios mecanismos de gestión de base de datos.

- **Drivers DBMS-based.** El driver accede a los datos físicos de manera indirecta a través del gestor de la fuente de datos. En este caso, el driver procesa solo las llamadas hacia ODBC; el cual pasa las sentencias SQL al gestor de la base de datos para su proceso. Por ejemplo, los drivers para Oracle son drivers tipo DBMS-based debido a que Oracle posee un gestor para la administración de la base de datos del cual hace uso el driver. El lugar donde reside dicho gestor es inmaterial. Este puede residir en la misma máquina que el driver o en una máquina diferente en la red.

**Drivers File-based**

Los drivers del tipo File-based son usados con fuentes de datos como dBASE, el cual no provee un mecanismo de gestión de datos del cual pueda hacer uso el driver. Este tipo de drivers acceden a los datos de manera directa, y tienen que implementar un mecanismo de gestión para procesar las sentencias SQL, dicho mecanismo soporta instrucciones SQL a un nivel mínimo necesario.

Comparando los drivers tipo file-based con los DBMS-based, los primeros son mas difíciles de ser implementados debido a que se necesita crear un mecanismo de gestión de datos, aunque son menos complicados debido a que no existen protocolos de red, de igual forma son menos poderosos debido al tiempo invertido en su desarrollo en comparación con aquellos que son desarrollados por compañías de bases de datos (drivers DBMS-based).

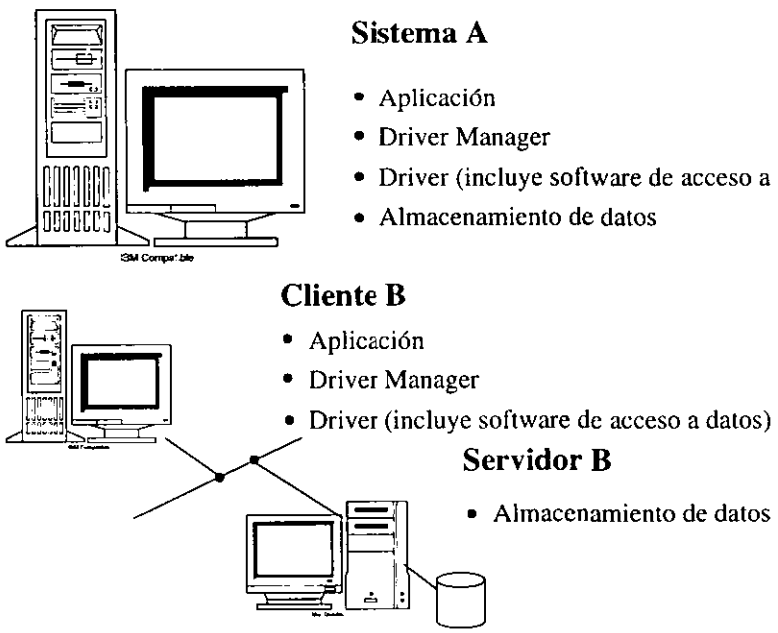



**Figura 2.4 Drivers tipo File-based.**

La figura 2.4 muestra dos configuraciones diferentes para drivers file-based, una en la cual los datos residen de manera local y la otra en la que los datos se encuentran en un servidor de archivos en la red.

### Drivers DBMS-based

Los drivers del tipo DBMS-based son usados con fuentes de datos como Oracle o SQL Server, DBMS que proveen mecanismos para la administración y uso de los datos, mecanismos de los cuales los drivers pueden hacer uso; de esta forma, un driver DBMS-based no accede de manera directa a los datos, lo hace a través del gestor o mecanismo que administra la base de datos.

Los drivers DBMS-based actúan como clientes en una arquitectura Cliente/Servidor, donde los datos fungen el rol de servidor. Generalmente, el cliente (el driver) y el servidor (la fuente de datos) residen en diferentes máquinas, aunque ambas pueden residir en un mismo equipo corriendo bajo un sistema operativo multitarea.

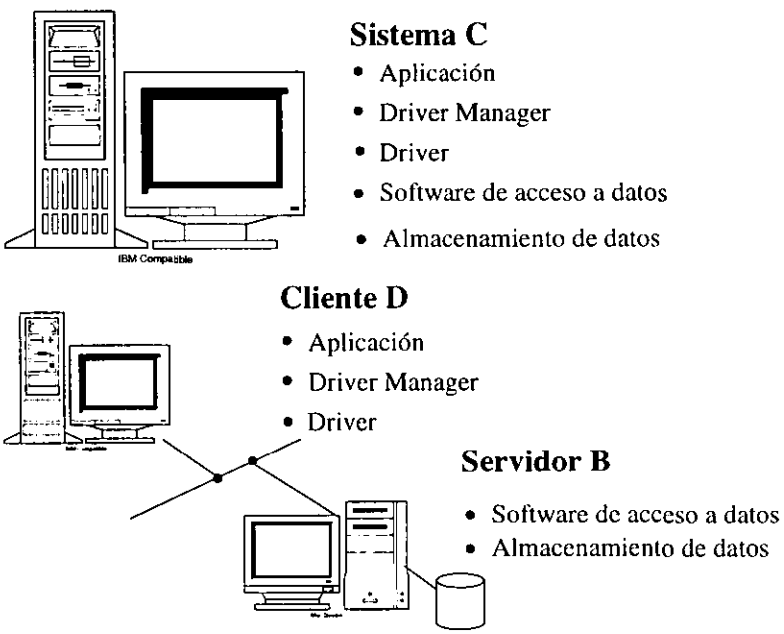



**Figura 2.5 Drivers tipo DBMS-based.**

La figura 2.5 muestra estas dos formas de configuración para drivers DBMS-based: corriendo bajo un mismo equipo y en diferentes equipos.

### *2.2.4 Data Sources*

Un data source es simplemente un archivo que hace referencia a una fuente de datos, dicha fuente de datos puede ser un archivo o una base de datos en particular.

El propósito de un data source es "reunir toda la información técnica necesaria para acceder a datos –el nombre del driver, la dirección de red (IP), el software de red, y mas- en un lugar sencillo y oculto al usuario final"[5]. El usuario debe poder visualizar desde una aplicación, solo una lista –por ejemplo, inventarios, cartera, ventas, etc.- de recursos a datos, escogiendo uno de estos recursos, la aplicación se conectará, sin mostrar donde se encuentran los datos o cómo se accede a estos.

*Los data source*: El término data source es usado para especificar cada registro que existe en el administrador ODBC y que hace referencia a una fuente de datos. Cada data source actúa a manera de middleware (véase la figura 2.6) entre el ODBC API y la fuente de datos.

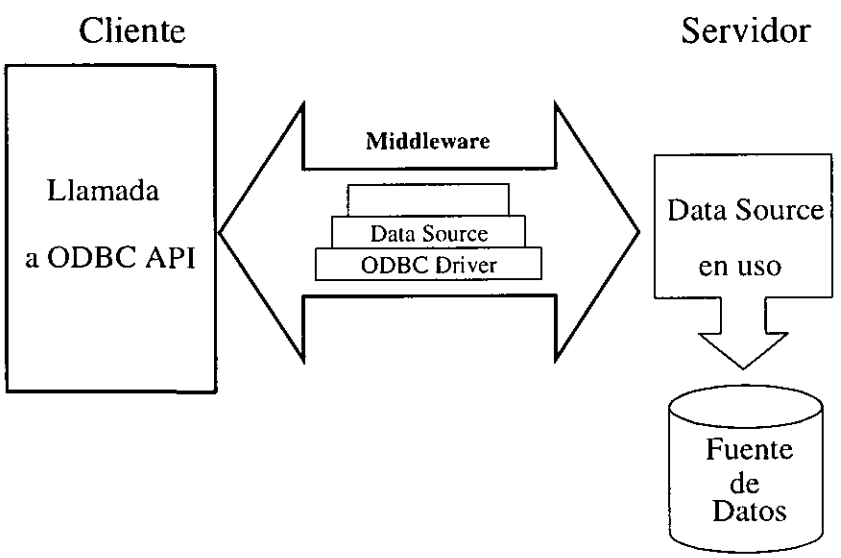



**Figura 2.6 El Data Source como middleware**

[5] *Microsoft ODBC 3.0 Software Development Kit and Programmers Reference.* Op. Cit. pp. 44

### Tipos de data sources

Existen dos tipos de data sources, data sources de usuario y data sources de sistema. Aunque ambos contienen información similar acerca de la fuente de datos, la diferencia radica en la forma en que la información es almacenada. A causa de estas diferencias, estos dos tipos de data sources son usados de diferente manera.

### Data sources de usuario

Los data sources de usuario (Machine Data Sources) son almacenados en el sistema con una cuenta de usuario definida. Asociada con el data source, está toda la información que driver manager y el driver necesitarán para conectarse a la fuente de datos. Por ejemplo, para un data source Xbase, podría necesitarse el nombre del driver Xbase, la ruta completa del directorio que contiene los archivos Xbase y algunas opciones que le indiquen al driver como usar esos archivos (modo de lectura, tipo de acceso, etc.). Para un data source de Oracle, se podría necesitar el nombre del driver, el servidor donde el DBMS reside, el usuario y el password.

### Data sources de sistema

Los data sources de sistema (File Data Sources) son almacenados en un archivo y proporcionan la información que será usada de forma repetida por un simple usuario o compartida por un grupo de usuarios. Cuando un data source de sistema es usado, el driver manager realiza la conexión a la fuente de datos usando la información que se encuentra en el archivo .DSN (Data Source Name, por sus siglas en inglés). Este archivo puede ser manipulado como cualquier otro archivo.

## Usando data sources

Los data sources son generalmente creados para el usuario final mediante un programa llamado Administrador ODBC, dicho administrador solicita al usuario el driver que será usado. El driver despliega entonces una ventana de diálogo la cual solicita la información necesaria para realizar la conexión a la fuente de datos. Después que el usuario ha proporcionando toda esta información, el driver la almacena en el sistema.

Finalmente, la aplicación llama al Driver Manager y le envía el nombre del data source de usuario o la ruta que contiene el data source de sistema. Cuando el dato enviado es un data source de usuario, el driver manager busca en el sistema (operativo) hasta encontrar el driver usado por el data source. El driver manager carga entonces el driver y le envía el nombre del data source, el driver usa este nombre para encontrar la información que necesita para la conexión a la fuente de datos. Al final, el driver se conecta a dicha fuente de datos, típicamente mediante la validación de un usuario.

Cuando el dato enviado desde la aplicación hacia el driver manager es un data source de sistema, el driver manager abre el archivo y carga el drive especificado en el data source y, de igual forma, el driver realiza una validación de usuario.

Existen algunas fuentes de datos para las cuales no es necesaria una validación de usuario, estas son, por lo regular, fuentes que utilizan drivers file-based, como es de suponer, esto generalmente es debido a que dichas fuentes de datos no poseen un administrador de base de datos que por lo regular, es el encargado de la seguridad en el acceso a datos.

## Usando el Administrador ODBC

El Administrador ODBC es la aplicación mediante la cual se puede configurar cada data source. El Administrador ODBC es una aplicación usada para crear y dar mantenimiento a data sources. Este administrador existe en dos versiones para 16- y 32-bits dependiendo de la versión de sistema operativo que se tenga.

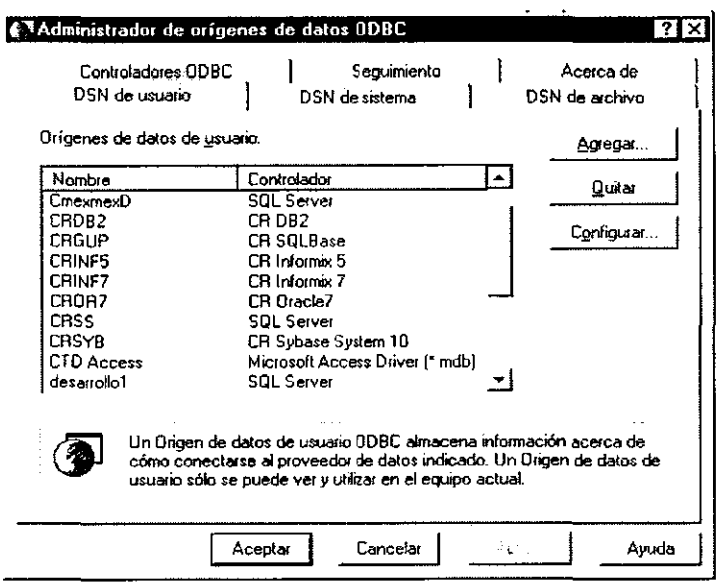



**Figura 2.7 El Administrador ODBC**

Cuando el administrador ODBC es ejecutado, éste despliega todos los registros data source que están disponibles en el sistema. (véase la figura 2.7). Cada conexión a una fuente de datos tiene su propio registro en el administrador ODBC. Cuando un programa accede a una fuente de datos, lo hace mediante el uso del nombre dado al registro data source.

La información contenida en cada registro es mostrada si se selecciona y se hace click en el botón "Configurar". (Ver figura 2.8). La información incluye el nombre del data source, el nombre del servidor que contiene la fuente de datos, información de dicho servidor (por

ejemplo dirección IP), el nombre de la base de datos a acceder cuando la conexión haya sido establecida, etc.

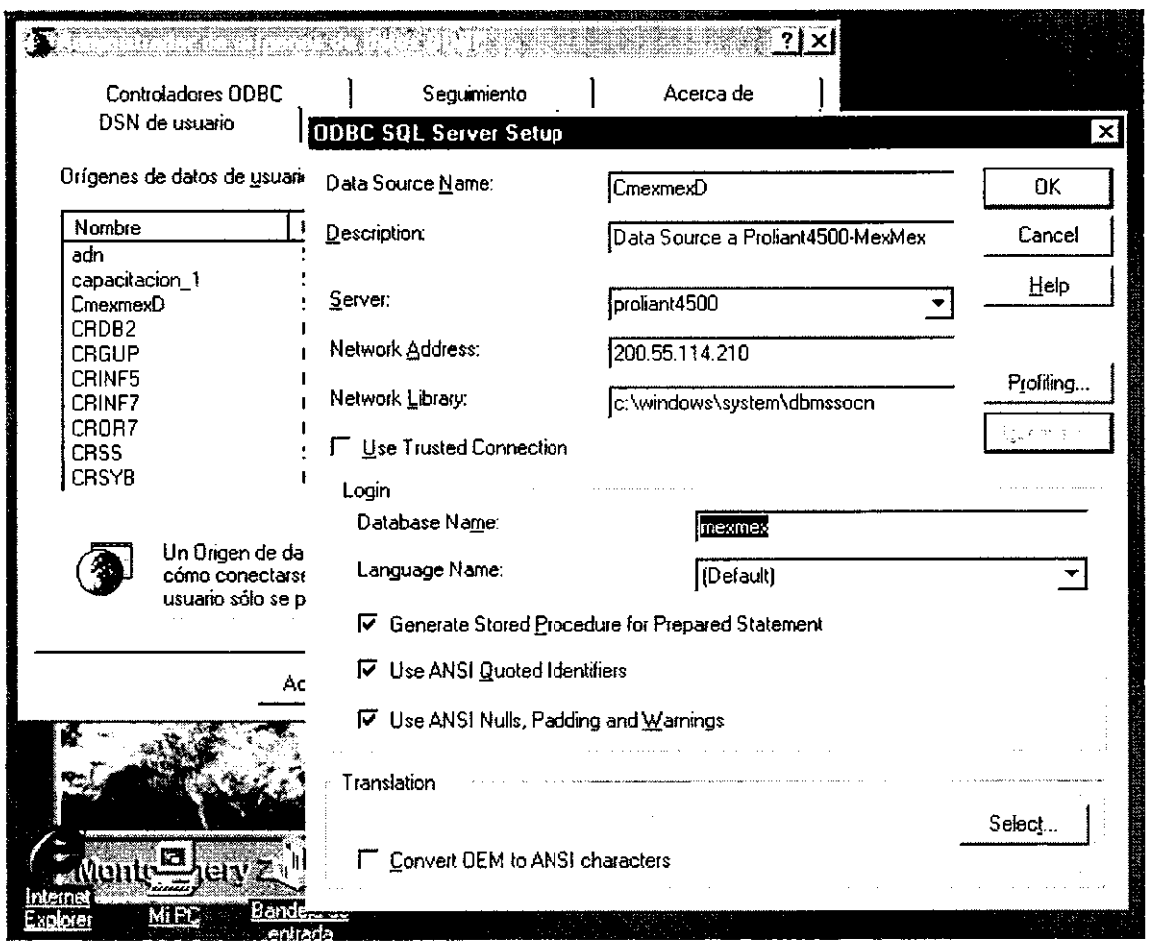



**Figura 2.8 Las propiedades de un Data Source**

La información de cada registro depende de la fuente de datos a usar, por ejemplo, una conexión a SQL Server o Informix requiere mas datos que una conexión a una base de datos Access donde solo se necesita especificar el lugar físico donde se encuentra dicha base de datos.

Cuando se añade un registro al administrador ODBC, la aplicación muestra una ventana con los drivers existentes (aunque se pueden añadir nuevos drivers al sistema y cada driver es proporcionando por la compañía que desarrolló la fuente de datos, por ejemplo Oracle, Informix, SQL Server, etc.) para que se escoga uno de estos drivers asociado al nuevo data source. Vease figura 2.9.

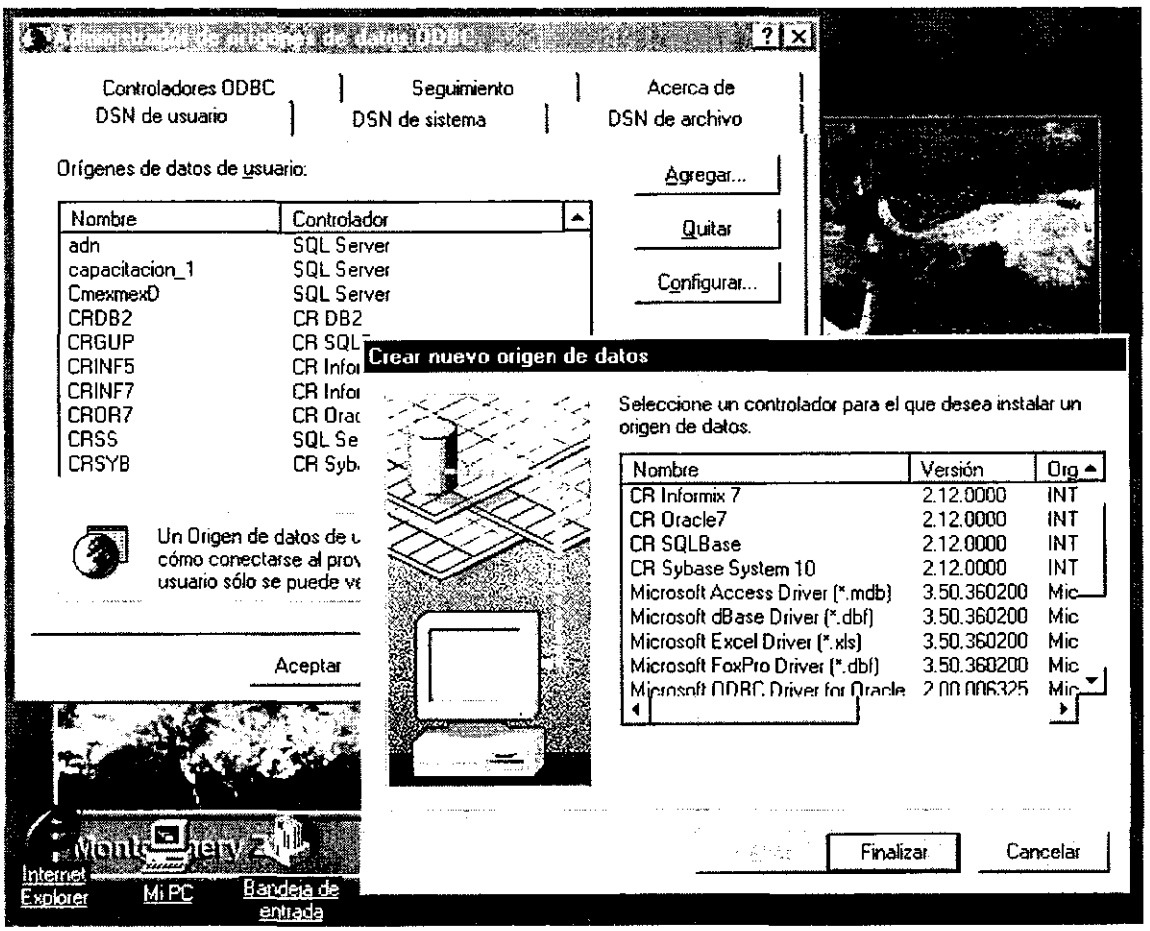



**Figura 2.9 Agregando un Data Source**

## 2.3 La solución ODBC

La pregunta final es entonces, ¿ Cómo estandariza ODBC el acceso a fuentes de datos ?. Existen dos requerimientos de arquitectura:

- Las aplicaciones deben ser capaces de acceder a múltiples fuentes de datos usando el mismo código fuente sin necesidad de recompilar los programas.
- Las aplicaciones deben ser capaces de acceder a múltiples fuentes de datos simultáneamente.

Hay una pregunta más debido a la realidad del mercado, ¿ Cuáles características de los DBMSs debe exponer ODBC ? ¿ Sólo las características que son comunes en todos los DBMSs, o cualquier característica disponible en todo DBMS ?

ODBC resuelve estos problemas de la siguiente manera:

- **ODBC es una interfaz de llamada por proceso**. Para resolver el problema de cómo las aplicaciones pueden acceder a múltiples fuentes de datos, ODBC usa un driver para cada fuente de datos que soporta ODBC. El driver implementa las funciones ODBC API. Para usar un driver diferente, la aplicación no necesita se recompilada, en lugar de esto, la aplicación simplemente carga el nuevo driver y llama a las funciones contenidas en él. Para acceder a múltiples fuentes de datos de manera simultánea, la aplicación carga los drivers necesarios de manera paralela.

- **ODBC posee una gramática estándar SQL.** Esta gramática está basada en la especificación X/Open SQL CAE. La aplicación puede enviar peticiones SQL a la fuente de datos usando la gramática estándar de ODBC, si la sentencia SQL tiene una gramática diferente a la que es reconocida por la fuente de datos, el driver se encarga de

convertir la sentencia antes de enviarla a la fuente de datos. Estas diferencias entre sentencias SQL cada vez son más raras a medida que las fuentes de datos están usando la gramática estándar SQL.

- **ODBC provee un Driver Manager que administra el acceso simultáneo a múltiples fuentes de datos.** Cuando una aplicación necesita un driver en particular, ésta primero solicita un apuntador al driver mediante el cual manejará la conexión, y entonces, la misma aplicación pide al Driver Manager que el driver en cuestión sea cargado. El Driver Manager carga el driver y regresa el resultado de la operación a la aplicación avisando si la carga fue exitosa o no. De esta forma, la aplicación puede pedir la carga de otro driver al Driver Manager, el cual se encarga de mantener el control de cada driver en memoria, esto facilita la administración de los drivers cargados a un mismo tiempo.

- **ODBC expone un número importante de características y propiedades aunque no requiere que cada driver soporte todas estas características.** Si ODBC expusiera solo las características más comunes a todos los drivers, su uso no sería muy común. La razón por la que hoy en día existe una gran cantidad de DBMSs es debido a que cada una ofrece características en particular. Si ODBC tuviera todas las características que están disponibles en cada DBMS, sería imposible para los drivers implementar todas estas características.

  En lugar de esto, ODBC expone un número importante de características –más de las que son soportadas por la mayoría de los DBMSs- lo que origina que los drivers implementen solo una parte de dichas características. El resto de las características son implementadas por los drivers solo si son soportadas por la fuente de datos (DBMS) a la que se está accediendo. De esta forma las aplicaciones pueden ser diseñadas para hacer uso de las características mas simples de las DBMSs, de igual forma dichas

aplicaciones tienen, mediante ODBC, la capacidad de checar por una característica en particular.

De esta forma, una aplicación puede determinar que características soporta un driver de un DBMS en particular. ODBC provee funciones que obtienen información general acerca del driver y el DBMS así como una lista de las funciones que dicho driver soporta. ODBC también posee diferentes niveles para la gramática SQL, los cuales especifican que tipo de instrucciones son soportadas por cada driver en particular.

Es importante destacar que ODBC define una interfaz común para cada una de las características que expone. De esta forma ODBC es quien recibe los comandos a realizar y no el driver de manera directa. Una de las ventajas de esto es que las aplicaciones no necesitan ser modificadas cuando las características soportadas por un DBMS son actualizadas, en lugar de esto, cuando se instala un driver que ha sido actualizado, la aplicación automáticamente usará las nuevas características (o las características actualizadas).

# 3. EL MODELO DE OBJETOS RDO

Visual Basic provee una variedad de formas para el acceso a datos. Además del acceso secuencial a archivos y archivos binarios, Visual Basic soporta un amplio rango de interfaces para el acceso a fuentes de datos.

Visual Basic posee una gama completa de herramientas para realizar cualquier aplicación cliente que necesite acceder a alguna fuente de datos. Esto da la posibilidad de implantar de manera rápida cualquier tipo de aplicación Cliente/Servidor. Visual Basic también soporta la escalabilidad de código y controles, es decir si se tiene una aplicación realizada en Visual Basic 3 está puede posteriormente ser abierta desde Visual Basic 4 o 5 y los cambios que Visual Basic realiza son mínimos, casi siempre de manera transparentes, ya que cualquier propiedad o método permanece como en la versión anterior mas las nuevas propiedades y métodos adicionados con la nueva versión.

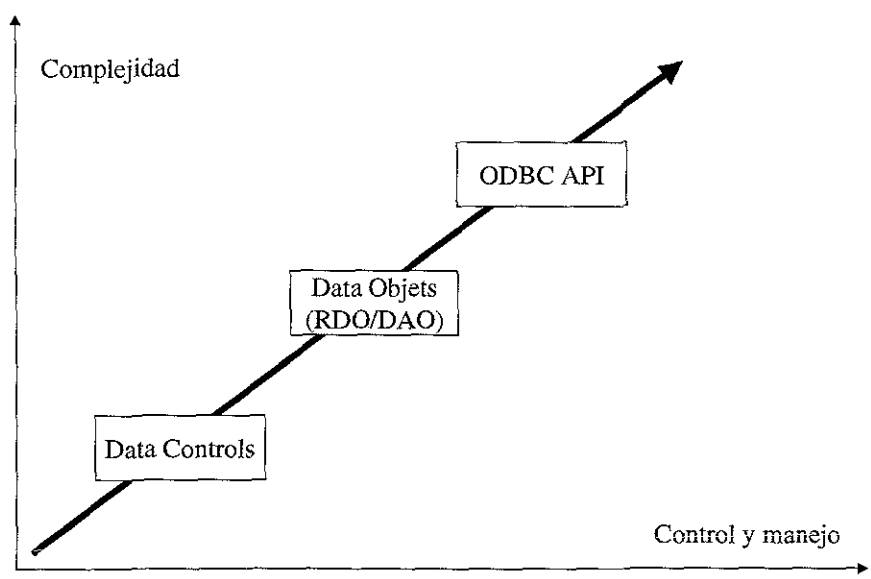



**Figura 3.1 El avance del acceso a datos**

Para tener la posibilidad de realizar una aplicación de manera rápida, Visual Basic incluye varias formas estándar de acceder a fuentes de datos: el Data Control, Data Access Objects (DAO), VB-SQL, Remote Data Objects (RDO) y la conectividad ODBC API. Con el incremento de la complejidad se incrementa de igual forma el control y poder que se tiene sobre la forma de acceso (véase figura 3.1).

El objetivo del presente capítulo es presentar el modelo de objetos RDO, por tal motivo, los métodos restantes solo se mencionan de forma somera. Como se puede ver en la gráfica y, a medida que se irá viendo en el presente capítulo, RDO no posee un nivel muy alto de complejidad lo cual lo hace de fácil uso, así mismo, el control que mantiene sobre los datos es de tipo regular a alto.

## 3.1 ¿ Qué es RDO ?

RDO es miembro de la familia de modelos de programación Cliente/Servidor, una familia que continua en evolución, RDO es un modelo implantado vía una delgada capa de código

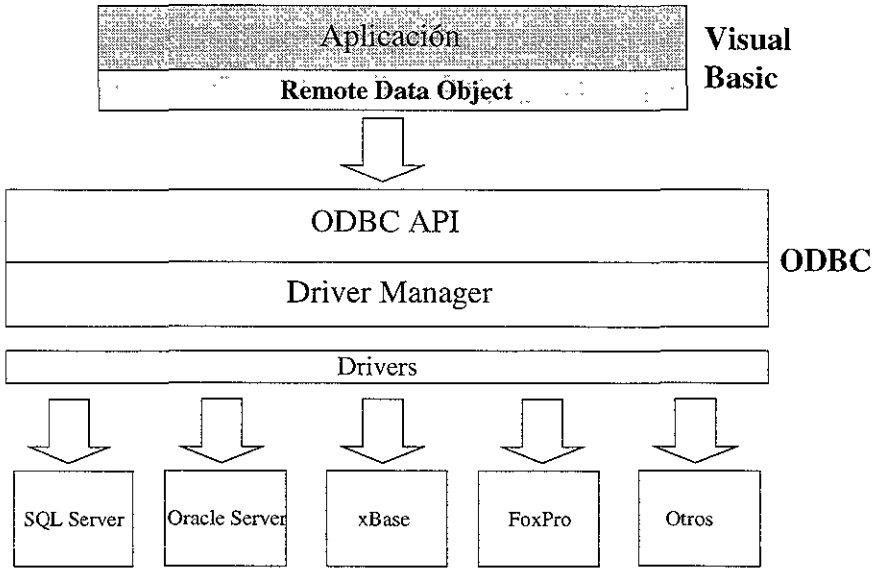



**Figura 3.2 Accesando a datos mediante RDO**

sobre ODBC API, el Driver Manager y un driver ODBC específico (como se observa en la figura 3.2). El Driver Manager es quien establece las conexiones, obtiene los datos desde la fuente de datos y ejecuta los procedimientos (sentencias SQL o stored procedures) usando el mínimo de recursos del cliente.

Visual Basic 4.0 Enterprise Edition implantó el modelo de objetos RDO como un rasgo nuevo. RDO provee una manera simple y eficaz de acceder a fuentes de los datos mediante ODBC dentro de un programa Visual Basic. Aunque es transparente a ambos usuario y diseñador. En una palabra, pretende ser mas amigable al desarrollador.

**Ventajas de RDO**

Además de RDO, hay otras maneras de acceder a datos por medio de ODBC dentro de Visual Basic: Jet Engine/Data Access Objects (Jet/DAO), VB-SQL, y ODBC API. RDO une los rasgos más buenos de las otras técnicas; como resultado, es tan fácil usar como DAO pero más poderoso y eficaz que VB-SQL u ODBC API. Esta combinación de características dan a RDO la mejor opción para una conexión a fuentes de datos por medio de ODBC. Examinemos algunas ventajas específicas.

*Uso de Controles*

Como Jet/DAO, RDO permite usar controles tales como el Remote Data Control (RDC) a partir de una estructura de resultados (rdoResultSet).

*Memoria*

Jet/DAO está diseñado para trabajar con fuentes de datos ISAM como dBASE o Paradox. Estos archivos son datos puros y no poseen un gestor o administrador de base de datos para hacer búsquedas, ordenar, y dar mantenimiento a los datos, por esta razón Jet provee un

gestor cuando es usado con este tipo de fuentes de datos. Los data sources proveen sus propios gestores, esto hace que el Jet sea redundante. Como RDO fue diseñado para usarse mediante ODBC, no necesita este tipo de gestores, por consiguiente requiere menos recursos de memoria y es más eficiente que Jet.

*Operaciones asíncronas*

Cuando se está ejecutando una búsqueda con Jet/DAO, la aplicación que realiza tal proceso queda suspendido mientras la búsqueda no halla sido completada. RDO, como VB-SQL y ODBC soporta ejecuciones asíncronas: la búsqueda es enviada a la fuente de datos y la aplicación continua con mas procesos, en algún momento los resultados de la búsqueda son enviados como respuesta a la aplicación, ésta es notificada del resultado de la búsqueda, procesa los datos de la búsqueda y continua procesando.

*Otras características*

RDO soporta otros rasgos poderosos, disponibles sólo debajo de VB-SQL u ODBC. Estos incluyen cursores, parámetros tipo OUTPUT y resultados a partir de stored procedures, múltiples sets de resultados, así como un número limitado de renglones regresados por una petición.

*Requisitos para usar RDO*

Para desarrollar una aplicación RDO, se debe usar VB 4.0 o superior (32-bit Enterprise Edition). Se debe tener drivers ODBC instalados en la máquina y una fuente de los datos disponible. Se requiere también un conocimiento del lenguaje SQL para implantar las sentencias.

### 3.1.2 El modelo de objetos RDO

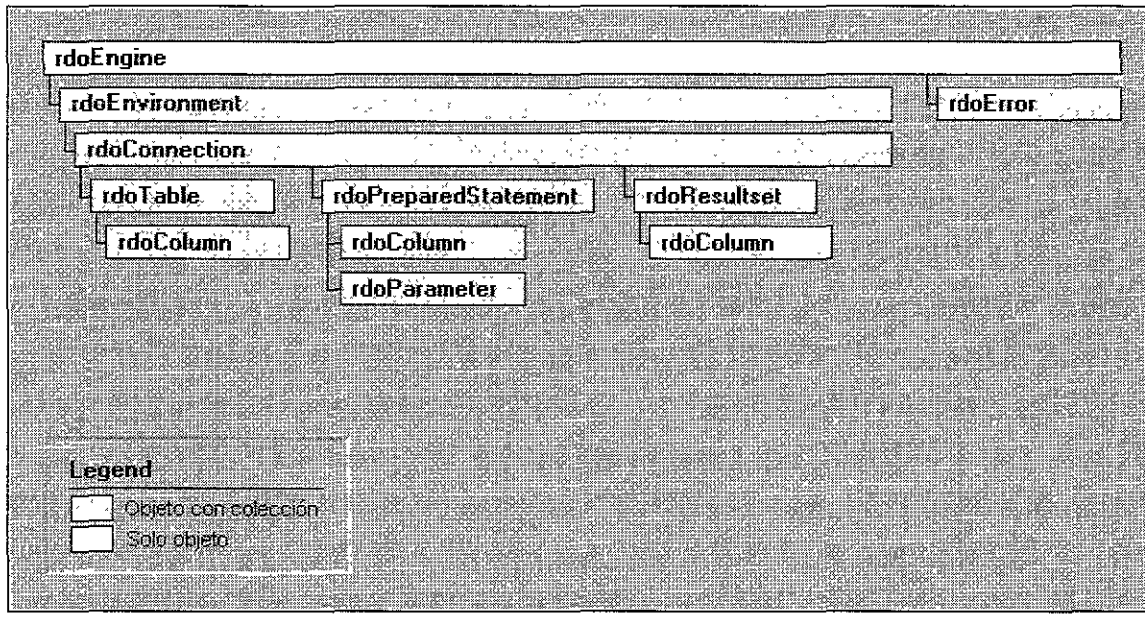



**Figura 3.3 El modelo de objetos RDO**

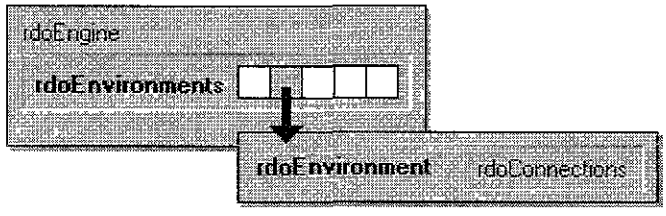
La figura 3.3 ilustra el modelo de objetos del Remote Data Object. A continuación se presenta cada una de sus partes.

#### RdoEngine


El objeto rdoEngine representa el Data Source o la fuente de datos de manera global, como modelo de máximo nivel jerárquico, rdoEngine es un contenedor de más objetos (los cuales se ven adelante), a partir de este objeto se encuentran los demás elementos (objetos) del modelo RDO.

RdoEngine es la representación del gestor de la fuente de datos, y en general, de cualquier fuente de datos manejada por el driver manager. Como lo muestra la figura 3.3, rdoEngine no pertenece a ninguna colección. Una colección es un grupo de componentes relacionados bajo un mismo nombre (el del grupo).

### RdoEnvironment(s)




Un objeto rdoEnvironment representa una colección de conexiones a una fuente de datos, esto significa que puede realizar transacciones simultáneas usando cada una de las conexiones que se encuentren dentro del mismo rdoEnvironment.

La colección rdoEnvironments, es un arreglo en el que se encuentra cada rdoEnvironment instanciado dentro de un rdoEngine. Esto se logra mediante el método **rdoCreateEnvironment** que posee el rdoEngine.

El primer rdoEnvironment es creado cuando se hace referencia por primera vez a la clase RDO dentro de Visual Basic.

Cada rdoEnvironment tiene la capacidad de manejar sus conexiones de manera transaccional, es decir, si se aplica el método `CommitTrans` o `RollBackTrans` desde un rdoEnvironment, éste a su vez, aplica dicha instrucción a cada una de las transacciones pendientes en cada conexión. Obviamente, la transacción comienza a partir de la declaración `BeginTrans`.

Para hacer referencia a cada rdoEnvironments dentro de la colección se tienen tres formas:

```
rdoEnvironments("nombre")
```

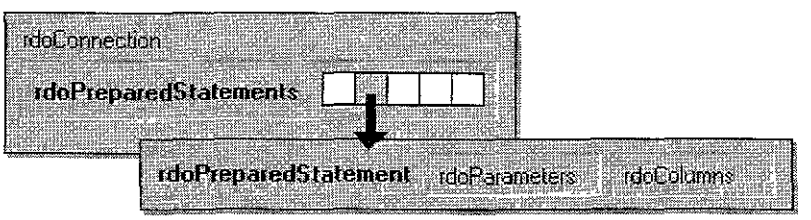
```
rdoEnvironments!nombre
```

y

```
rdoEnvironments(n)
```

Donde n es el número asociado a cada rdoEnvironment, comenzando desde cero.

### RdoConnection(s)




Un objeto del tipo rdoConnection representa una conexión abierta a alguna fuente de datos, accediendo a ésta como una base de datos específica. Una vez abierto un rdoConnection no se puede cambiar de base de datos dentro de la misma conexión, se tendría que instanciar otro rdoConnection. Una colección de rdoConnections representa a todas las conexiones hechas dentro de un mismo rdoEnvironment.

Una vez realizada la conexión, se puede manipular la base de datos usando los métodos y propiedades que rdoConnection provee. Por ejemplo:

- Se puede usar el método **Execute** para realizar alguna afectación a los datos (`update`, `delete`, `insert`, o alguna transacción completa por medio de stored procedures), en donde no sea necesario que la fuente de datos regrese algún resultado. Como **Execute** no regresa ningún resultado, se puede usar la propiedad **RowsAffected** para saber el número de renglones que fueron afectados.

- Usar el método **OpenResultSet** para crear un nuevo objeto rdoResultSet, el cual se explica más adelante.

- Usar el método **CreatePreparedStatement** para crear un nuevo objeto rdoPreparedStatement, el cual se verá de igual manera adelante.

- Usar el método **Close** para cerrar la conexión ocurrente. Cuando un rdoConnection es cerrado, todos los objetos creados a partir de este rdoConnection (rdoResultSets, rdoTables, rdoPreparedStatements, etc.) son destruidos.

- Usar los métodos **BeginTrans**, **CommitTrans** y **RollBackTrans** para la implantación de afectaciones transaccionales dentro de la aplicación.

- Determinar ciertas propiedades para la conexión tales como **LoginTimeOut** que determina el tiempo máximo para lograr la conexión y **QueryTimeOut** la cual determina el tiempo máximo para terminar una ejecución.

De la misma manera que para rdoEnvironment (y en general para cualquier colección existente dentro del modelo RDO y por lo cual no se ejemplificará en lo subsecuente) se puede hacer referencia a cada rdoConnection dentro de la colección rdoEnvironments:

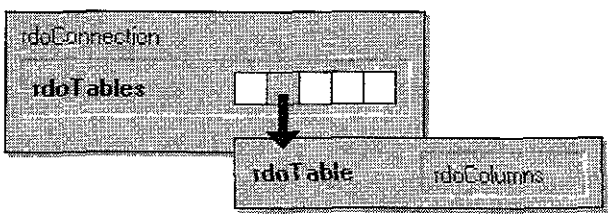
```
rdoConnections("nombre")
```

```
rdoConnections!nombre
```

y

```
rdoConnections(n)
```

## rdoTable(s)

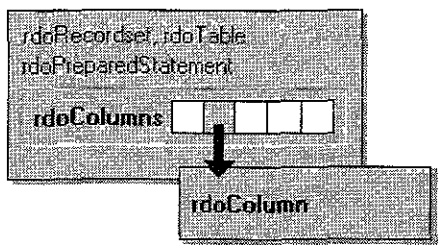



Un rdoTable representa la definición de una tabla de una base de datos en el modelo de objetos RDO, la colección rdoTables contiene cada rdoTable de un rdoConnection.

El manejo de una tabla de una base de datos se logra a través del mapeo de la tabla en cuestión por medio del objeto rdoTable, el cual provee de métodos y propiedades para la manipulación de dicha tabla. Por ejemplo:

- Examinar las propiedades de las columnas de la tabla mediante el objeto rdoColumn.
- Crear un rdoResultSet usando el método **OpenResultSet** a partir de un rdoTable.
- Obtener el nombre de la tabla a partir de la propiedad **Name**.
- Usar la propiedad **RowCount** para obtener el numero de registros de la tabla.
- Usar la propiedad **Type** para determinar el tipo de tabla que se está mapeando.

### rdoColumn(s)

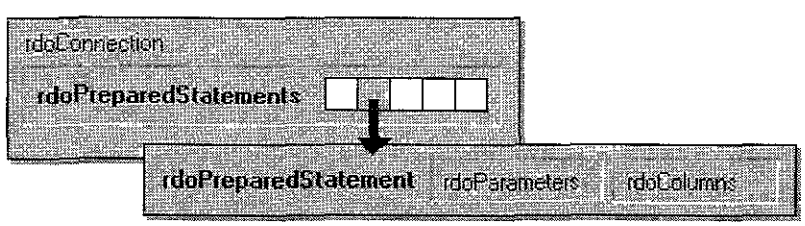



El objeto rdoColumn contiene cada columna obtenida desde la fuente de datos ya sea a partir de un rdoRecordSet, un rdoTable o un rdoPreparedStatement. De igual forma existe la colección rdoColumns la cual contiene cada rdoColumn obtenido.

El objeto rdoColumn es usado para obtener las características de una columna tales como tipo de columna (**Type**), longitud en bytes (**Size**), atributos (**Attributes**), etc.

El nombre de cada rdoColumn es asignado a partir de el nombre físico de la columna en la tabla o asignado a en la sentencia SQL y puede ser obtenido mediante la propiedad **Name**.

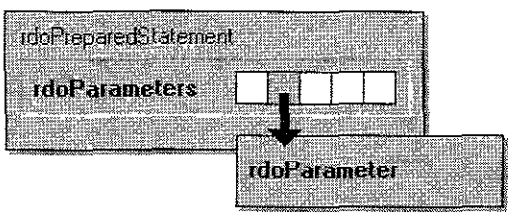
### rdoPreparedStatement(s)




Un rdoPreparedStatement es una sentencia SQL precompilada por medio de la cual se puede:

- Asignar la sentencia SQL que se desea ejecutar usando la propiedad **SQL**.
- Usar la propiedad **Type** para determinar que tipo de afectación se realizará: Modificación, Selección o la ejecución de un procedimiento.
- Usar la propiedad **QueryTimeOut** para determinar el tiempo máximo de ejecución.
- Usar **MaxRows** para determinar cuantos registros se desean obtener como máximo.
- Determinar por medio de **RowsAffected** cuantos registros fueron modificados.

### rdoParameter(s)




Un objeto del tipo rdoParameter representa cada parámetro asociado a un rdoPreparedStatement. Este objeto también forma parte de una colección: rdoPreparedStatements.

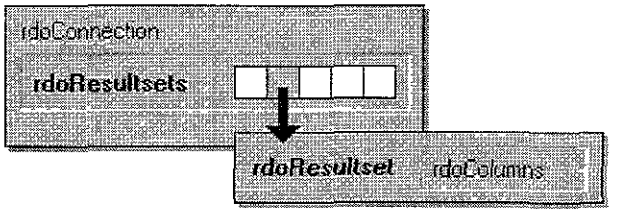
Por medio de cada rdoParameter se puede asignar los párametros necesarios para realizar una búsqueda o afectación de datos a través de rdoPreparedStatement. Por ejemplo:

- Asignar la propiedad **Direction** para determinar si el parámetro será del tipo `input` u `output`.

- Asignar el tipo de parámetro por medio de la propiedad **Type**, la cual es similar al Type de un rdoColumn.

- Asignar el valor que se pasará a la fuente de datos mediante la propiedad **Value**.

### rdoResultSet(s)




Un objeto rdoResultSet representa los resultados arrojados de una búsqueda o afectación a una fuente de datos. Este objeto también pertenece a una colección, la colección rdoResultSets.

Para crear un rdoResultSet se logra con el método **OpenResultSet** que provee rdoConnection. Para que un rdoResultSet pueda ser creado, es necesario que la petición hecha a la fuente de datos arroje una estructura compuesta de al menos una columna, los registros arrojados pueden incluso ser cero, pero si el resultado no arroja ninguna columna Visual Basic generará un error cuando se intente leer dicho **OpenResultset**.

Un objeto rdoResultSet es un apuntador a una estructura de datos compuesta por columnas y registros, es posible "navegar" a través de los registros por medio de los métodos **MoveFirst**, **MoveLast**, **MovePrevious** y **MoveNext**.

De igual forma se puede determinar el tipo de rdoResultSet deseado mediante el argumento *type*, el cual es opcional; esto significa que, si no especifica dicho argumento, el default es keyset-type. Existen cuatro tipos de rdoResultSets:

- Forward-only: Usando este tipo, se crea un cursor el cual accede a una estructura de registros, los cuales pueden ser incluso modificados, pero solo existe un sentido para

navegar a través de los datos: hacia adelante. Es decir, una vez que rdoResultSet se encuentra en un registro, es imposible aplicar un método **MovePrevious**. Este tipo de cursores por lo regular es usado cuando la aplicación no necesita regresar a una posición anterior, por ejemplo para un "barrido" de registros.

- Static-type: Este tipo de rdoResultSet genera una "copia" de los datos existentes en la fuente de datos, dichos datos pueden incluso ser modificados, pero esto depende de los drivers que soporten realizar modificaciones para este tipo de rdoResultSets.

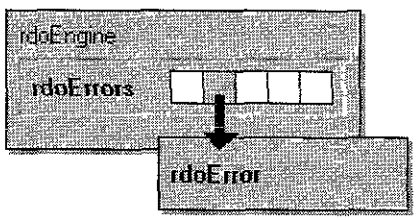
- Keyset-type: Este tipo de rdoResultSet es dinámico, se puede navegar en él hacia ambas direcciones y los registros son modificables, además que se puede agregar o borrar registros, con la restricción que el cursor permanece de forma fija. Si se desean ver los cambios hechos a los registros es necesario aplicar el método **Refresh**.

- Dynamic-type: Este tipo de rdoResultSet es dinámico, se puede navegar en él hacia ambas direcciones y los registros son modificables, además que se puede agregar o borrar registros, este tipo de cursor no permanece de forma fija y se ven reflejados los nuevos registros borrados y/o agregados.

Existen otros métodos y propiedades para este objeto tales como:

- **AbsolutePosition,** el cual sirve para conocer en que registro se encuentra apuntando en cursor en determinado momento.
- **BOF y EOF,** éstas son propiedades booleanas para conocer si el apuntador se encuentra al principio o al final del cursor.
- **RowCount,** permite conocer el número total de registros existentes en el cursor.
- **AddNew,** método que permite adicionar nuevos registros.
- **Delete,** permite borrar registros.
- **Edit,** que, junto con **Update**, permite modificar registros.

**rdoError(s)**




El objeto rdoError pertenece a la colección llamada rdoErrors, cada rdoError contiene la información acerca de errores ocurridos en el modelo de objetos RDO, cada operación incluida en RDO puede generar uno o más errores. Cada vez que una nueva operación RDO genera un nuevo rdoError, la colección es "limpiada" y el o los nuevos rdoErrors son añadidos a la colección. Por supuesto, si una operación no genera ninguna falla durante el tiempo de ejecución, no se generarán nuevos errores.

El objeto rdoError posee métodos y propiedades que pueden ser usados para verificar cual fue el error ocurrido, "limpiar" dicho error, y ver su descripción.

Existe la propiedad **Number**, la cual arroja un valor entero que indica el último error generado, esto es de gran ayuda en la aplicación porque permite corregir el error en el momento que sucede y continuar el proceso de la aplicación sin interrumpir al usuario de la aplicación. La propiedad **Description** permite conocer la descripción técnica del error, aunque muchas veces es necesario mostrar el error al usuario de una forma clara en base a sus conocimientos, esto implica mostrar un mensaje que describa la causa del error de manera entendible.

# 4. SOCKETS

## 4.1. ¿ Qué es un socket ?

En todas las áreas de la tecnología se tiende siempre a crear modelos simplificados o niveles que sean transparentes a usuarios de aplicaciones superiores. Con esta premisa surgió el paradigma "socket" popularizado por la Berkeley Software Distribution (BSD) de la Universidad de California, en Berkeley. Este socket, o enchufe, consiste en un conjunto de ordenes para gestionar la transmisión de datos en cualquier aplicación o programa, pero a diferencia de lo ocurrido hasta entonces donde cada programador hacia las suyas propias, se trata de un conjunto de órdenes standard común para todos los usuarios del entorno para el que es creado, en el caso de Berkeley Software Distribution se trata del entorno UNIX.

Para poder entender lo que es un Socket se hace necesario el definir ciertos conceptos. Al hablar del interfaz de Sockets, se nombran *llamadas al sistema*, *descriptores de entrada/salida* y, por supuesto, *sockets*. En las siguientes subsecciones vamos a establecer una introducción a cada uno de ellos.

### Llamadas al sistema

Las llamadas al sistema constituyen los puntos de entrada al sistema operativo. A través de estas llamadas es posible solicitar una serie de *servicios* que únicamente puede ejecutar el núcleo (kernel) del sistema.

Cuando un proceso genera una llamada al sistema, pasa a modo kernel. Desde el punto de vista del programador las llamadas al sistema se ven como funciones API, las cuales

devuelven un valor entero negativo en caso de error, o un valor positivo si se ejecutaron sin complicaciones. Este valor positivo tendrá distintos significados según la llamada al sistema en concreto.

### Descriptores de entrada/salida

Los descriptores son variables, generalmente números enteros, que utiliza el sistema para identificar elementos de entra/salida. Cuando un proceso abre un fichero, recibe por parte del sistema un descriptor que le permitirá acceder posteriormente al recurso que identifica. Esta política se sigue también con los sockets, con la idea de mantener una interfaz similar a otras clases de entrada/salida. Cuando un proceso de usuario crea un socket, recibe por parte del sistema un descriptor que le permitirá acceder al canal abierto para realizar operaciones sobre él.

### Socket

Un socket es definido como un punto virtual de comunicación que permite intercambiar datos con otro proceso de usuario (aplicación). Este último proceso puede estar situado en la misma máquina o en otra conectada a la red. Al igual que, los números de puerto en los protocolos de transporte permiten la utilización de un canal de comunicación físico por varios procesos sin entorpecerse, los sockets realizan la misma función a nivel programación.

El programador ve el socket como un lugar en el que puede escribir y del que puede leer, como si de un fichero se tratase (salvando lógicamente las diferencias de tratamiento que no pueden ser escondidas). Por supuesto, un socket deberá estar asociado a un número de puerto y a un protocolo específico del nivel de transporte, y el programador lo utilizará

mediante un **descriptor** (ver descriptores de entrada/salida) y realizando **llamadas al sistema**.

Por lo tanto, como ideas fundamentales establecemos que un socket es un canal que pueden abrirse (*open*) con una dirección de destino; por él pueden enviarse datos (*send,write*),y recibirse (*recv,read*), y también puede cerrarse el canal (*close*).

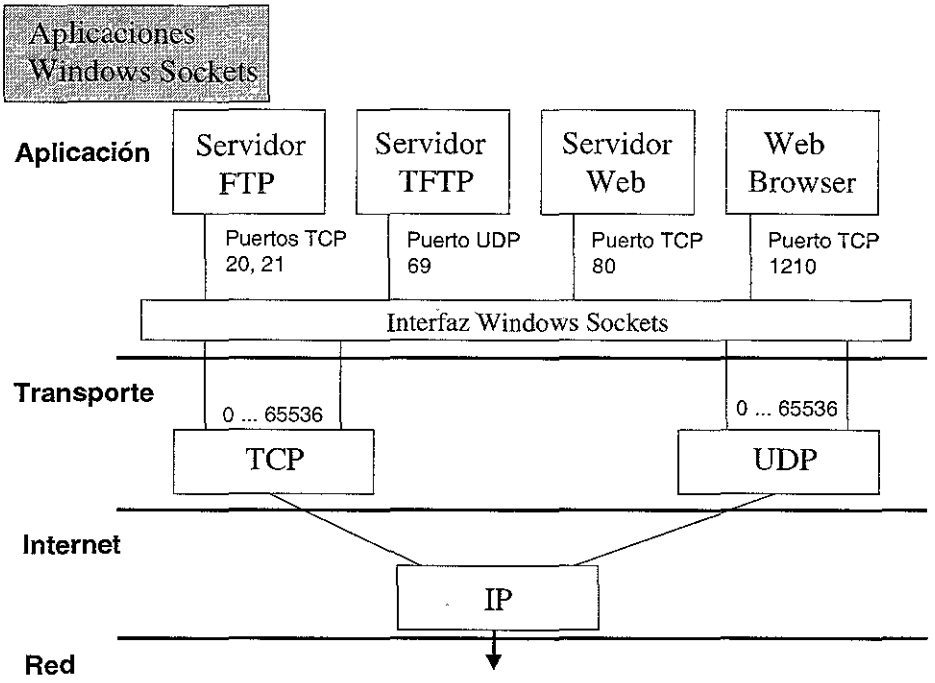



**Figura 4.1 Aplicaciones Windows Sockets**

Una aplicación crea un socket mediante la definición de tres elementos: la dirección IP del lugar donde se encuentra el socket en la red, el tipo de protocolo a usar y el puerto a través del cual se va a conectar.

## 4.2 Introducción a Windows Sockets

Cuando hablamos de Windows Sockets nos referimos a una interfaz de red, programado para usar en Microsoft Windows. Este interfaz posee el característico estilo de las rutinas socket de Berkeley y las extensiones especificas de Windows diseñadas para permitir al programador sacar ventaja de la forma de conducir los mensajes que tiene Windows.

En su origen Windows Sockets fue diseñado para su utilización con el protocolo de transporte TCP/IP. Sin embargo, las extensiones actuales de Windows Sockets permiten su utilización con protocolos de transporte distintos de TCP/IP. La implantación de Windows Sockets en Windows NT Server y Windows NT WorkStation ofrece un servicio de comunicación entre procesos independiente del transporte.

La especificación Windows Sockets sirve para proporcionar una sencilla API (Interfaz de Programación de Aplicaciones, herramienta que mejora y potencia los servicios de Windows) con la que los programadores pueden desarrollar aplicaciones y utilizar los diversos sistemas de software de red tanto comerciales como estándar. Además, en el contexto de una versión particular de Microsoft Windows, Sockets define un modo de conectarse tal que una aplicación escrita para el Windows Sockets API pueda trabajar con un protocolo correspondiente facilitado por cualquier vendedor de software de red. Con este conjunto de órdenes el programador de aplicaciones no tiene porque preocuparse del nivel de red, tan solo sabe que haciendo uso de las llamadas a subrutinas ya hechas y siguiendo el standard establecido, la transmisión de datos se realizara de forma eficaz y correcta.

Entre los Sockets existentes en la actualidad tenemos algunos que permiten la transmisión de datos tanto en una red de área local como por línea telefónica, de forma que se crea una conexión virtual entre el ordenador remoto que accede por módem y la red.

Trabajando de esta forma como si estuviésemos conectados físicamente a la red, pudiendo recibir y mandar datos que tenemos en nuestro ordenador y haciendo uso de los programas que más se adapten a nuestras necesidades y gustos, siempre que sean compatibles con Windows Sockets.

La API de Windows Sockets proporciona una interfaz estándar para aquellos protocolos que tienen esquemas de direccionamiento diferentes. Se desarrolló también como ayuda en el proceso de estandarización de una API que sirviese para todas las plataformas de sistemas operativos. Los protocolos que soportan Windows Sockets son TCP/IP y NWLink (IPX/SPX).

Windows Sockets está formado por los siguientes elementos

- Wsock32.dll, que comparte el espacio de direcciones de la aplicación Windows Sockets en modo usuario.

- Emulador Windows Sockets que proporciona el nivel de conversación Windows Sockets entre las aplicaciones Windows Sockets y los protocolos anteriores.

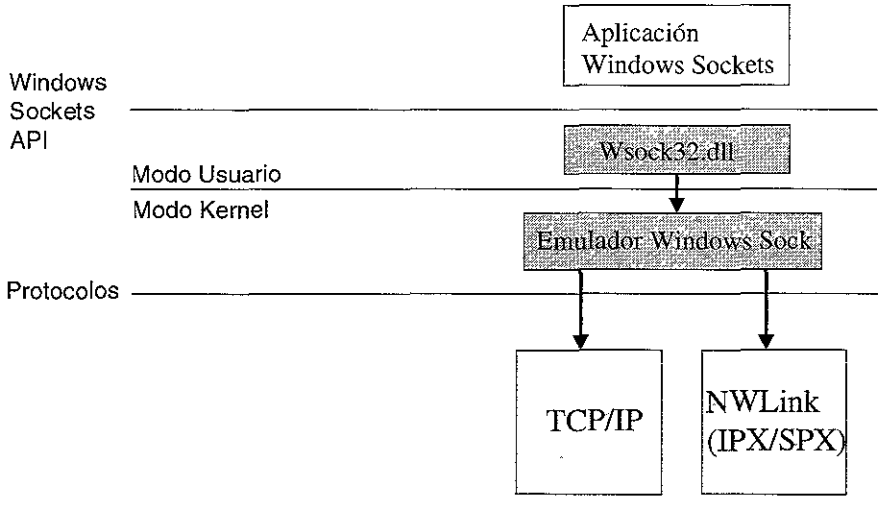



**Figura 4.2 Interfaz de programación de Windows Sockets**

Observando la figura anterior (4.2), se puede ver que la interfaz Windows Sockets es una API usada para enviar y recibir datos por una red. Diseñada originalmente como una interfaz de nivel superior para pilas de transporte de red TCP/IP, el API de Windows Sockets proporciona una interfaz Windows estándar para muchos transportes con diferentes esquemas de direccionamiento incluyendo protocolos como los que se observan en la figura.

Windows Sockets especifica una interfaz de programación basada en la interfaz Sockets. Incluye un conjunto de extensiones diseñadas para aprovechar la naturaleza orientada a mensajes de Windows, de esta forma Windows Sockets es una especificación abierta y estándar.

Windows Sockets es una interfaz entre las aplicaciones y los protocolos, y trabaja como una tubería bidireccional de entrada y salida de datos de aplicación. Windows Sockets está implantado como una biblioteca de enlace dinámico (Winsock32.dll) que hace que las aplicaciones y el servicio de transporte en los protocolos se encuentren dinámicamente unidos en tiempo de ejecución.

## 4.3 El control WinSock

Visual Basic provee un control denominado **Microsoft WinSock Control** el cual es usado para, mediante sockets, hacer uso de los servicios de TCP y UDP. Mediante este control, no es necesario conocer detalles muy técnicos de TCP o realizar llamadas de bajo nivel al API de Windows Sockets.

Mediante este control se puede crear una aplicación servidor y una aplicación cliente las cuales puedan comunicarse entre si, y realizar el envío y recepción de datos (**SendData** y **GetData**).

### *4.3.1 Métodos*

- **Connect**, este método (que para Visual Basic es, en realidad un evento), permite abrir la conexión por medio de sockets.

- **Close**, este método permite cerrar la comunicación por medio de sockets.

- **Bind,** permite especificar el puerto local y la dirección IP que se usarán para la comunicación.

- **Listen,** este método permite crear un socket (está restringido su uso sólo para TCP); una vez creado, el socket está listo para atender llamadas de lectura/escritura.

- **SendData**, permite realizar un envío de datos a un equipo remoto.

- **GetData**, permite obtener información a partir de un socket el cual es almacenado en una variable de la aplicación. También se puede especificar el tipo de dato a recibir así como la longitud máxima. La sintaxis es :

  objeto.**GetData** variable, [tipo,] [Longitud]

- **PeekData**, este método, al igual que **SendData** permite leer datos desde un socket, con la diferencia que los datos no son removidos de la información contenida en el socket,

es decir, con **PeekData** se puede leer el mismo dato cualquier número de veces y el dato permanece en el socket hasta no ser leído/removido por **SendData**.

### *4.3.2 Propiedades*

- **BytesReceived**, esta propiedad permite obtener el valor del número de bytes recibidos en la ultima invocación del método **SendData**.

- **LocalHostName**, permite conocer el nombre de la máquina local.

- **LocalIP**, permite conocer la dirección IP de la máquina local con el formato xxx.xxx.xxx.xxx

- **LocalPort**, esta propiedad, a diferencia de las anteriores que son read-only, permite tanto obtener como asignar el número de puerto local que será usado en la máquina local para Lectura/Escritura. Si el número de puerto es igual a cero, el control WinSock asignará un valor *random* el cual será usado a lo largo del tiempo de ejecución.

- **Protocol**, mediante esta propiedad se puede obtener o asignar el protocolo a usar en la aplicación un valor igual a cero asignará el protocolo TCP (valor por default) mientras que un valor igual a uno asignará el protocolo UDP para la aplicación.

- **RemoteHost**, permite obtener y asignar cual será la máquina remota a la que el WinSock enviará y recibirá datos. El valor asignado puede ser con formato URL (por ejemplo, FTP://ftp.microsoft.com) o IP (por ejemplo 198.105.232.1).

- **RemoteHostIP**, permite obtener la dirección IP de la máquina remota.

- **RemotePort**, permite obtener y asignar el número de puerto remoto al cual la máquina local se conectará.

- **State**, mediante esta propiedad, se puede conocer el estado actual del control WinSock a partir de la siguiente tabla:

| Constante VB | Valor | Descripción |
|---|---|---|
| sckClosed | 0 | Conexión no abierta |
| sckOpen | 1 | Conexión abierta |
| sckListening | 2 | Listo (Ready o Listen) |
| sckConnectionPending | 3 | Conexión pendiente |
| sckResolvingHost | 4 | Buscando Host |
| sckHostResolved | 5 | Host encontrado |
| sckConnecting | 6 | Conectando |
| sckConnected | 7 | Conectado |
| sckClosing | 8 | Cerrando conexión |
| sckError | 9 | Error |

# 5. PROPUESTA

El continuo avance tecnológico en el área computacional, así como la exigencia de más recursos dentro de dicha área, hacen necesario el planteamiento de nuevos modelos para el acceso a bases de datos.

Este capítulo pretende mostrar un nuevo modelo para acceder a bases de datos mediante la administración de conexiones y la petición de transacciones desde un controlador de conexiones, incrementando la velocidad de respuesta así como aprovechando mejor los recursos disponibles de la base de datos, realizando la comunicación Cliente/Servidor mediante el uso de sockets y manteniendo conexiones abiertas disponibles para la atención de transacciones.

No es la realización de un sistema en particular, sino mas bien el desarrollo de un modelo el cual quiere presentarse como prototipo, y el cual podría aplicarse de manera general en el acceso a bases de datos dentro del área de desarrollo de sistemas.

## 5.1 El modelo actual

Desde la aparición del modelo Cliente/Servidor, las compañías y los desarrolladores han buscado día a día la manera de optimizar el acceso a bases de datos. Se han creado estándares como ODBC, para acceder de manera universal a casi cualquier fuente de datos, se han buscado mejores técnicas para agilizar el acceso y el manejo de volúmenes muy grandes de registros, se ha buscado la forma de que un gestor de una base de datos posea la capacidad de "decidir" que índice usara en determinado tipo de búsqueda.

Sin embargo, existe una parte dentro de todo este avance tecnológico la cual no sufrido cambios relevantes: el modo de acceso a una fuente de datos.

### *5.1.1 Presentación*

El acceso común a una fuente de datos dentro de la arquitectura Cliente/Servidor es semejante a una conversación entre dos personas, por lo cual se le puede denominar "una conversación Cliente/Servidor" y está representado de la siguiente forma:

1. **Una aplicación realiza una petición de conexión a la fuente de datos.** En un sistema Cliente/Servidor normalmente, el cliente (que es en este caso la aplicación del usuario) es la parte encargada de solicitar una conexión a una fuente de datos, por lo regular esto se hace mediante una previa validación de usuario (login) y contraseña (password). El cliente envía su clave de usuario y contraseña, en algunas ocasionas envía también el nombre de la base de datos a la que pretende conectarse.
2. Existen fuentes de datos que no poseen un administrador de base de datos, tal es el caso de archivos DBF o Excel, donde no es necesario enviar una clave de usuario y contraseña, en este caso el acceso normalmente es permitido siempre y cuando la tabla que se desea abrir no este siendo usada de manera exclusiva por otra aplicación cliente.

3. **Validación de acceso en la fuente de datos.** Si la fuente de datos posee un gestor que administre el acceso, dicho gestor realizará una validación para determinar quien es el cliente que pretende realizar la conexión, esto, como se mencionó, se realiza mediante la validación de la clave de usuario y contraseña. En el caso de las fuentes de datos sin gestor de administración, el acceso por lo regular es concedido siempre y cuando no

exista alguna otra aplicación que esté utilizando el mismo recurso de una manera exclusiva.

4. **Conexión.** Si la validación fue exitosa o el recurso no está siendo usado por otra aplicación de manera exclusiva, el acceso es permitido.

Hablando en términos de programación y dependiendo del tipo de aplicación que se tiene como cliente, se puede obtener un valor entero positivo o igual a cero (*handle*) el cual indica que la conexión fue exitosa; también se puede decir que, si un error no ocurre, la conexión es tomada como exitosa, esta forma es conocida como conexión con manejo de errores. A continuación se muestra estos dos ejemplos, aunque, puede haber otros estilos de conexión dependiendo del software que se use para este fin, el primero es una conexión el C y el segundo es una conexión en Visual Basic:

*Conexión por medio del manejo de handle:*

```
int hnd;    /* hnd es el handle para la conexión */

hnd = source.connect(user, password);
if (hnd < 0)
   printf("No conectado\n");
else
   printf("Conexión exitosa\n");
```

*Conexión por medio de manejo de error:*

```
On Error GoTo no_conexion

source.connect user, password
Msgbox "Conexión exitosa"
Return

No_conexion:
Msgbox "No conectado"
```

5. **La aplicación está conectada a la fuente de datos.** Una vez que la conexión ha sido exitosa, nuestra aplicación puede realizar transacciones de búsqueda o afectación en los datos, siempre y cuando el usuario (en caso de que haya sido necesario uno) posea los permisos necesarios para realizar esta operación.

   A partir de este momento la fuente de datos actúa como un servidor de datos y la aplicación se mantendrá conectada al servidor hasta que ella misma solicite su desconexión.

6. **Petición de datos desde la aplicación cliente.** La aplicación envía alguna instrucción a la fuente de datos ya sea ésta una sentencia SQL o algún otro tipo de sentencia que la fuente de datos pueda interpretar. Cuando la aplicación solicita ciertos datos del servidor (búsqueda o *query*) o solicita la afectación en los datos (agregar, borrar o modificar), esto lo realiza mediante el envío de comandos al servidor de datos.

7. **Validación de peticiones.** El servidor de datos valida dos aspectos: la sintaxis y la seguridad de los datos: la sintaxis se refiere a verificar que una instrucción hecha por la

aplicación no se encuentre mal escrita y el gestor de la base de datos pueda interpretar perfectamente la instrucción; la seguridad en los datos se refiere al hecho de verificar, antes de realizar una operación sobre los datos, los permisos que posee el usuario de la aplicación. Finalmente, si estos dos aspectos son válidos, el servidor "acepta" la ejecución de la instrucción enviada por la aplicación cliente.

8. **Ejecución de peticiones.** Una vez hecha la validación de la petición, el servidor de datos ejecuta el comando solicitado por la aplicación. Esta ejecución consiste en una modificación a los datos (ya sea con una inserción, borrado o modificación), la selección de determinada información contenida en el servidor o ambos. El tiempo de ejecución por parte del servidor dependerá de la complejidad de la afectación así como de la velocidad de procesamiento del servidor.

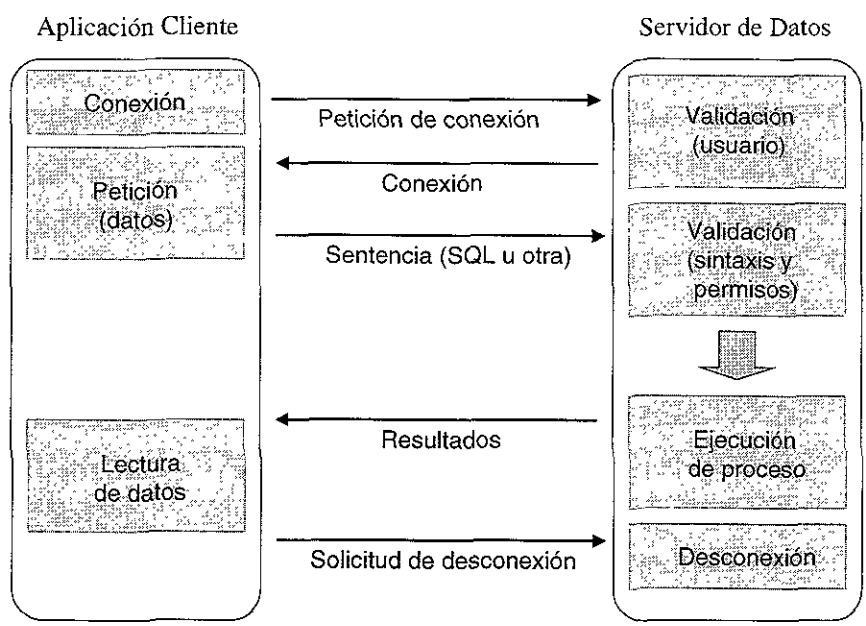



**Figura 5.1 Diagrama de flujo Cliente-Servidor de datos.**

9. **Envío de resultados.** Este punto no existe en todos los casos de conversaciones Cliente/Servidor, el envío de resultados se presenta solo cuando la aplicación cliente así lo solicitó, por ejemplo en los casos de búsquedas. Cuando una aplicación hace una

petición al servidor de datos, esta petición puede incluir que sean regresados ciertos registros como resultados -condición que no es forzosa-, dichos resultados son enviados en forma de columnas y registros.

A partir de este punto, la aplicación esta lista para ejecutar otra operación o solicitar la desconexión de la fuente de datos.

10. **Solicitud de desconexión.** Si otra petición fue hecha por parte de la aplicación, el flujo de la operación se repite a partir del paso seis, en caso contrario, el servidor cierra la conexión y la operación Cliente/Servidor queda concluida. Una aplicación una vez desconectada puede volver a conectarse cuantas veces sea necesario –y esto dependiendo del estilo de la aplicación- regresando, si lo desea al punto uno.

Como se puede ver, la forma de acceder a los datos desde una aplicación es casi siempre la misma: conexión, ejecución, envío de los resultados y desconexión.

El avance tecnológico –como se mencionó- en el área de aplicaciones Cliente/Servidor para el acceso a bases de datos ha originado una mejora considerable del lado del cliente (front end) mediante mejores componentes de acceso y manejo de datos remotos, y del lado del servidor (back end) mediante la optimización de búsquedas de información, uso de procedimientos almacenados (stored procedures), etc. Todo este avance no ha ido a la par con el estilo de acceder a una fuente de datos desde una aplicación.

Pongamos un ejemplo ficticio, la empresa Eastern Union es una empresa dedicada a la realización de transferencias de dinero a 1000 ciudades diferentes en todo el mundo por medio de concesiones a compañías locales de cada país –como Elektra, Bital y Telecomm en México-.

Eastern Union posee un servidor central de base de datos –digamos SQL Server- en Estados Unidos el cual atiende a 480 usuarios concurrentes. En dicho servidor se encuentra la información de todos los envíos disponibles (no cobrados), así como información tal como nombre del remitente, del beneficiario y destino del envío.

Cada concesionario se conecta a la base de datos central por medio de comunicación satelital como lo muestra la siguiente figura.

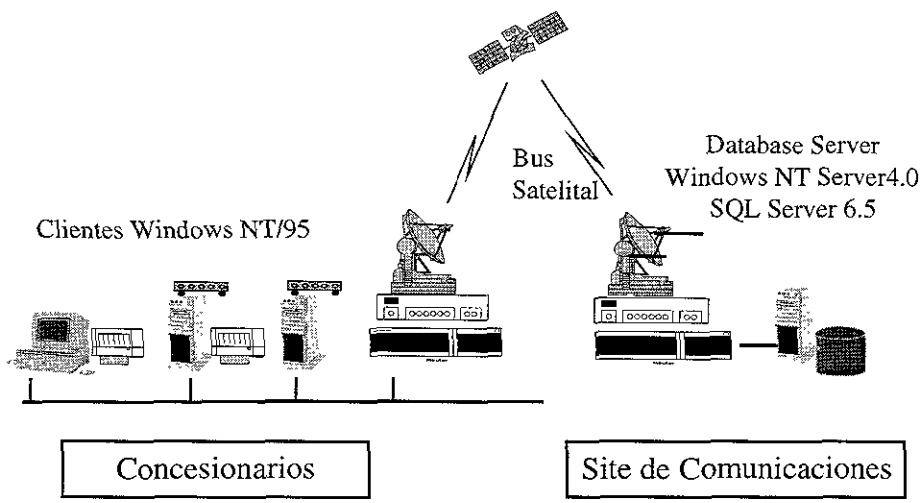



Figura 5.2 Un esquema global de Eastern Union

Supongamos que el tiempo promedio para realizar la conexión (satelital) a la base de datos es de 1.5 minutos mientras que el tiempo promedio de cada transacción (ya sea al enviar o pagar una transferencia) una vez establecida la conexión, es de 5 segundos.

A primera vista, se puede observar que, el *tiempo real de transacción* representa un 5.5 por ciento del tiempo total de la operación o también se puede decir que el servidor ocupa el 94.5 por ciento por cada operación realizando operaciones de conexión/desconexión.

Mediante el ejemplo expuesto se puede concluir que:

- Muchos sistemas de base de datos requieren de un proceso completo de sistema operativo por conexión. Para aplicaciones con cientos de usuarios esto genera una lista muy grande de *hosts* concurrentes conectados al servidor.

- Establecer una conexión a una fuente de datos frecuentemente resulta muy lento. Con una cantidad grande de usuarios conectándose y desconectándose el sistema (del servidor) degrada su desempeño de una manera severa.


**Figura 5.3 Tiempo por operación de Eastern Union**

*5.1.2 Ventajas y desventajas*

La implantación del modelo convencional de conversación Cliente/Servidor para acceso a fuentes de datos presenta una serie de características que pueden considerarse como ventajas, ya que permiten tener seguridad y buen manejo en general de aplicaciones Cliente/Servidor. Sin embargo también presenta ciertas desventajas que a continuación se enumeran.

*Ventajas*

- **Seguridad**. El esquema actual para el acceso a bases de datos ha sido, a lo largo del tiempo, mejorado con el propósito de robustecer la seguridad de los datos, que por cierto, es uno de los objetivos fundamentales de las bases de datos.

- **Conexiones individualizadas**, mediante el esquema anterior, cada aplicación cliente es tratada como un individuo aparte, esto permite personalizar el acceso a datos mediante la creación de grupos que poseen diferentes jerarquías sobre los datos, por ejemplo: administradores, supervisores, operadores, etc.

- **Acceso de aplicaciones cliente a múltiples fuentes de datos.** Permite el acceso de una aplicación cliente a varios servidores de datos de forma simultánea, lo que resulta cada vez más importante en los ambientes heterogéneos actuales.

- **Flexibilidad** en la organización al distribuir datos y aplicaciones en la red, ponerlos mas cerca de los usuarios y poder relocalizar datos sin necesidad de modificar la aplicación. El esquema Cliente/Servidor para bases de datos posee una arquitectura tan flexible que solo es necesario indicarle a la aplicación cliente "el donde" se encuentran los datos.

- **Acceso a la información cuando y desde donde la aplicación la necesite**. A la aplicación cliente solo le es necesario disparar una operación de conexión y otra de solicitud de datos, para obtener la información en el momento que la misma aplicación lo necesite.

*Desventajas*

- **Tiempo de operación**. Sin lugar a dudas, esta es la desventaja principal del modelo actual de acceso a fuentes de datos, la mayor parte del tiempo necesario para realizar una transacción Cliente/Servidor a una fuente de datos es tiempo ocupado en la conexión y autentificación de la aplicación cliente hacia el servidor.

- **Aprovechamiento no óptimo de recursos.** No se permite el buen aprovechamiento de la potencia de cómputo de los equipos (servidores). Cuando existe una alta concurrencia de conexiones y desconexiones de las aplicaciones cliente, se origina un incremento en la carga de trabajo del servidor, ocasionando una degradación en su desempeño y por ende, haciendo más lentas las conversaciones Cliente/Servidor.

- **Se incrementa el tráfico en la red.** Por cada conexión existente se genera una línea de comunicación que es constante mientras la aplicación no solicite su desconexión, dicha conexión a su vez requiere de un proceso completo de sistema operativo el cual la mantiene y respeta a lo largo de toda su existencia.

- **Conexiones morosas.** Cada conexión a su vez, permanece "colgada" a la fuente de datos mientras no sea solicitada la desconexión por parte de la aplicación cliente. Esto genera en ocasiones que la fuente de datos procese de manera más lenta las peticiones hechas por los clientes, lo cual significa que, por cada cliente que es añadido a la colección de conexiones, las demás aplicaciones clientes "sufren las consecuencias".

## 5.2 El modelo propuesto

El presente proyecto trata de resolver un problema actual en el área computacional: la reducción de costos tanto en horas/hombre, de tráfico de información en redes, así como de tiempo de procesamiento y respuesta en modelos Cliente/Servidor para acceso a bases de datos.

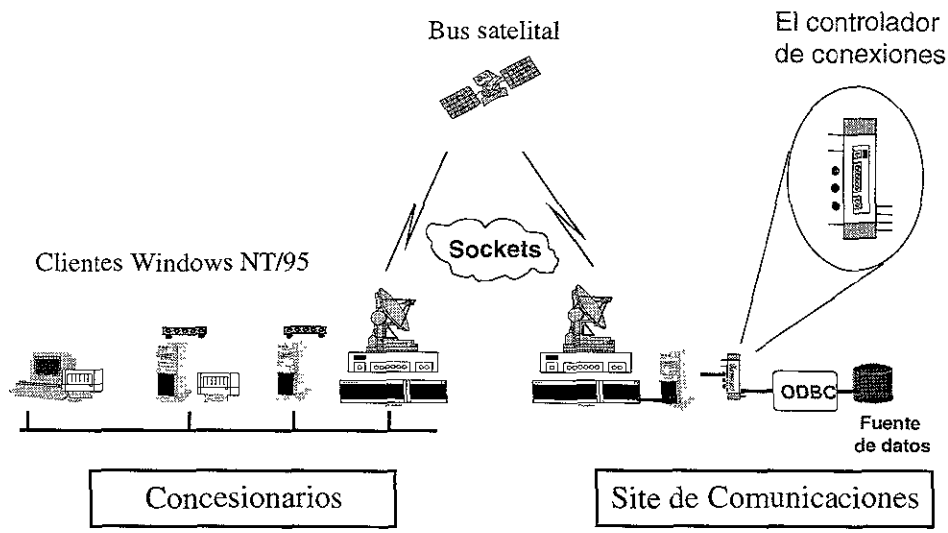



Figura 5.4 El modelo propuesto

Este proyecto pretende ofrecer un nuevo modelo para acceder a fuentes de datos mediante la administración de conexiones y la petición de transacciones desde un controlador de conexiones, incrementando la velocidad de respuesta así como aprovechando mejor los recursos disponibles del servidor (hardware) y de la fuente de datos, realizando la comunicación Cliente/Servidor mediante el uso de sockets y manteniendo conexiones abiertas disponibles para la atención de transacciones.

Pensando en esto, el presente proyecto expone la posibilidad de tener un servidor de conexiones configurable, en cuanto a la base de datos a administrar (la cual puede ser desde un simple DBF hasta una biblioteca AS400), y el número de conexiones máximas concurrentes, mediante el uso de uno de los objetos mas robustos diseñados para el acceso a bases de datos en ambientes Windows NT/95: RDO (Remote Data Object).

Incluso es posible desarrollar un cliente de datos orientado a un lenguaje y plataforma particular, el cual sea abastecido por un controlador de conexiones que trabaja bajo otra plataforma, esto sugiere posibilidades de adaptibilidad y portabilidad.

### *5.2.1 Presentación*

Los objetivos por los cuales, nace la posibilidad de este proyecto son los siguientes:

- La reducción de tiempos de procesamiento y respuesta en modelos Cliente/Servidor para acceso a datos.
- La reducción de tráfico de información en redes.
- La reducción de costo horas/hombre en el desarrollo de aplicaciones Cliente/Servidor para acceso a datos.

Veamos cada uno de los puntos.

**La reducción de tiempos de procesamiento y respuesta en modelos Cliente/Servidor para acceso a datos.** Como se concluye del apartado 5.1, la mayor parte del tiempo usado en una operación Cliente/Servidor es utilizado para proceso de conexión y validación de usuarios. Esto origina pérdidas en tiempo de operación de cualquier empresa, lo que origina a su vez pérdidas de dinero. Se crea la necesidad de plantear un nuevo esquema mediante el

cual no sea necesario pasar por este punto (el de la validación) cada vez que una aplicación cliente solicita información de una fuente de datos.

Con el logro de esta reducción de tiempo, se obtendrá una reducción en el tiempo global de procesamiento de una aplicación Cliente/Servidor, haciendo mas rápido el esquema de conversación entre el cliente y el proveedor de datos, esto se traduce en una mejor respuesta del esquema general Cliente/Servidor.

**La reducción de tráfico de información en redes.** Usando el modelo convencional, por naturaleza del mismo modelo, se tiene la necesidad de tener una conexión permanente por cada usuario conectado –por medio de su aplicación- a la fuente de datos, mientras dicha aplicación no solicite su desconexión, el servidor de datos tiene la obligación de no cerrar por ninguna causa la conexión, ya que ello generaría inminentemente un fallo en la aplicación cliente.

Si hablamos que la conexión es permanente mientras el cliente no "desee" desconectarse, estamos hablando que se necesita tener un servidor de datos con una capacidad media/alta para que pueda brindar una atención eficiente a cada usuario conectado. Esto origina costos muy altos: memoria, capacidad de procesamiento de los procesadores, rapidez del medio de comunicación, capacidad funcional del software usado como base de datos, etc.

Se crea otra necesidad: la necesidad de no mantener conexiones permanentes por cada aplicación cliente apuntando al servidor de datos, con la implantación de un modelo Cliente/Servidor que no utiliza conexiones permanentes se está hablando de la posibilidad de reducir el tráfico en la red, e incluso ayudar a mejorar el desempeño del servidor de datos mediante el no apilamiento de hosts en el servidor.

Cada host conectado a un servidor origina un proceso exclusivo en el sistema operativo que lo atiende, así como la completa atención y protección por parte de dicho sistema, es decir

se genera un proceso permanente en memoria que atiende al momento que el cliente requiere cualquier petición –válida o no- solicitada por dicho cliente. Si se logra omitir las conexiones permanentes dentro del modelo aquí expuesto, se habla de una considerable mejora en todo el esquema Cliente/Servidor incluyendo al medio mismo –la red- que participa como middleware.

Con la implantación de un controlador de conexiones como éste, se ofrecen mas ventajas y facilidades tanto al desarrollador de software como al usuario final y a la empresa: la transparencia entre aplicaciones, bases de datos, e incluso sistemas operativos.

**La reducción de costo horas/hombre en el desarrollo de aplicaciones Cliente/Servidor para el acceso a datos.** Dependiendo del software que se use para crear aplicaciones, el tiempo de desarrollo varia en gran manera, en base a las características de cada tipo de software se puede crear aplicaciones de manera rápida y a veces no tan rápida. De igual forma existe una varianza –ligera o brusca- entre los diferentes estilos para acceder a una fuente de datos incluso desde un mismo lenguaje o herramienta de desarrollo, por ejemplo, hablando de Visual Basic existen diferentes formas de acceder a datos, se puede usar DAO, RDO, ADO y otros; los desarrolladores de aplicaciones muchas veces tienen que invertir cierto tiempo para determinar cual sería la herramienta de desarrollo y el estilo que mas conviene a tal o cual aplicación.

Uno de los objetivos del presente proyecto es sugerir un estándar para el acceso a datos: un controlador de conexiones dinámico que pueda acceder a cualquier fuente de datos siempre y cuando dicha fuente de datos pueda ser accedida a través de ODBC. Logrando esto, se puede hablar de una significativa reducción de tiempo y costo en el desarrollo de aplicaciones, esto facilitaría la implantación de aplicaciones Controlador/Cliente de datos escondiendo la complejidad del modelo permitiendo a los desarrolladores centrarse sobre sus aplicaciones.

*Planteamiento*

De manera general, se expone el planteamiento para la implantación del modelo propuesto, el diseño deberá reunir las siguientes características:

El modelo propuesto, debe incluir tanto al cliente de datos como al servidor de datos, al que denominaremos en lo consecuente el controlador de conexiones. El modelo propuesto, deberá ser capaz de atender a diferentes usuarios (aplicaciones) a la vez, esto se realiza mediante la atención en el controlador de conexiones, las cuales una vez abiertas no se cerrarán hasta que el administrador –del controlador- lo solicite.

El controlador de conexiones debe poder conectarse a cualquier fuente de datos que posea un drive para ODBC. El servidor de datos, para poder ser parte de un sistema abierto, debe poder abrir conexiones de cualquier fuente de datos, siempre y cuando dicha fuente de datos posea un drive propio ODBC. De esta manera, el controlador tendrá una máxima interoperabilidad, es decir, la habilidad de acceder a diferentes fuentes de datos usando el mismo código fuente. El controlador solo preguntará al administrador de la fuente de datos que drive se usará para abrir conexiones.

Cada conexión atiende una petición de alguna aplicación cliente, la ejecuta y envía los resultados sean estos resultados exitosos o no; por tal motivo el controlador deberá ser capaz de soportar instrucciones erróneas por parte de los clientes, cuando este caso suceda, el controlador deberá enviar a la aplicación cliente el mensaje completo obtenido de la fuente de datos –*permission denied*, por ejemplo-.

Cuando una petición halla sido contestada, el controlador debe desconectar al cliente a fin de permitir que esa conexión quede libre y pueda ser usada por cualquier otra aplicación que en ese momento solicite información. De igual manera, un cliente, al ser desconectado,

debe poder reconocer la información enviada por el controlador para determinar si su petición fue ejecutada exitosamente o no; en caso de una petición no exitosa, el desarrollador de la aplicación cliente puede mandar un mensaje más cordial al usuario indicándole que su solicitud no fue llevada a cabo.

Mediante la independencia entre las conexiones y las aplicaciones se logra eliminar el esquema de aplicaciones que estén "colgadas al sistema" de manera permanente, mejorando el tráfico en la red y disminuyendo el tiempo que tarda el servidor de datos en atender cada aplicación cliente.

El protocolo de comunicación que se propone para comunicar al cliente con el servidor de datos es TCP/IP, esto es debido a que TCP/IP es una familia de protocolos diseñados para la interconexión de equipos de cómputo, independientemente de su arquitectura y el sistema operativo que ejecuten, además de ser un estándar de facto debido a la expansión de Internet. Se intenta implantar un modelo abierto, independiente a cualquier plataforma, usando TCP/IP el modelo no queda limitado a un sistema operativo en particular.

Se usarán sockets como medio de comunicación entre el cliente y el servidor de datos, por supuesto, un socket deberá estar asociado a un número de puerto y a un protocolo específico del nivel de transporte –en este caso TCP/IP-, y el administrador del controlador lo asignará cuando el controlador sea puesto en marcha. Al igual que, los números de puerto en los protocolos de transporte permiten la utilización de un canal de comunicación físico por varios procesos sin entorpecerse, los sockets realizan la misma función a nivel de programación.

Se deberá poder especificar el número de conexiones que el controlador atenderá así como el número del puerto lógico que usara para comunicarse con sus clientes a través de la red, estos dos parámetros (el número de conexiones y la especificación del puerto) deben especificarse en el controlador justo antes de que el controlador abra conexiones a alguna

fuente de datos, esto es con el fin de brindar más dinamismo al modelo. Por ejemplo podrían estar corriendo dos instancias del mismo controlador en un equipo de cómputo, un controlador atendiendo a aplicaciones que desean conectarse a una base de datos de ventas a través del puerto 1020 usando 10 conexiones y el segundo controlador atendiendo a otro departamento, el de contabilidad por ejemplo, a través del puerto 1040 con solo 5 conexiones; ambos departamentos dentro del mismo corporativo.

A cada conexión abierta por el controlador le será asociado un socket, el cual será el encargado de "escuchar" cualquier petición de alguna aplicación cliente; cuando un cliente solicite información, el socket se encargará de aceptar la requisición y quedará conectado en ese momento al cliente, el socket enviará el mensaje que contiene la petición al controlador. Se recomienda **englobar** en una sola función los procesos de conexión y envío de la sentencia SQL, esto con el fin de optimizar el tiempo de respuesta del modelo, por ejemplo se podría implantar una función llamada `envia` la que a su vez incluya los procedimientos `conecta` y `ejecuta`, el desarrollador de aplicaciones solo conocerá la función `envia`:

```
Private Sub Envia(SQLStatement)
    On Error GoTo No_conexion

    ApCliente.Conecta
    ApCliente.Ejecuta SQLStatemen, ApCliente.Usuario, _
            ApCliente.Password
    Exit Sub


No_conexion:
    MsgBox "Error al conectarse al controlador"
End Sub
```

El controlador se encargará de enviar la petición a la fuente de datos de manera asíncrona, es decir, realiza el envío de la petición y continua atendiendo otras requisiciones de las aplicaciones, cuando la fuente de datos haya ejecutado la sentencia requerida por la aplicación cliente enviará los resultados al controlador, y éste a su vez por medio del socket –que durante todo el proceso controlador/fuente de datos no se ha desconectado del cliente- enviará los resultados erróneos o no a la aplicación.

A diferencia del modelo actual, el modelo propuesto separa la validación sintáctica y la validación de seguridad: el controlador de conexiones será el encargado de validar la cuestión de seguridad mediante la validación de la clave de usuario enviada por la aplicación cliente comparándola con su clave propia usada al momento de conectarse a la fuente de datos (o que tal vez no necesitó y solo la tiene almacenada en memoria); la cuestión sintáctica –se recomienda- debe ser validada por la fuente de datos, el motivo principal es la varianza de los lenguajes que son capaces de interpretar cada fuente de datos, incluso existen diferentes "dialectos" derivados del mismo lenguaje SQL, además que se tendría que desarrollar un compilador sintáctico muy complejo con la capacidad suficiente de ser multilenguaje, o multidialecto en caso de usarlo solo para fuentes de datos que usen SQL.

Cuando el socket haya enviado el total de los datos a la aplicación cliente, disparará un proceso de desconexión, desligándose así del cliente y posibilitándose para la atención de cualquier otra aplicación que en ese momento intente solicitar información.

*Problemas que deben contemplarse en la interacción cliente/servidor*

1. Si una aplicación cliente muere deja en el controlador memoria ocupada. El controlador debe utilizar alguna forma de TIMEOUTS para desconectar los threads (conexiones) de los clientes no activos.
2. Una aplicación cliente no debe utilizar un IDConversación de otro cliente (Seguridad).

*Seguridad*

Antes de que el controlador sea puesto en marcha, se le deberá especificar un usuario y una contraseña, datos que serán usados para abrir las conexiones con la fuente de datos; en caso de tratarse de fuentes de datos que no posean un gestor que administre la seguridad de la fuente, los datos serán de cualquier manera almacenados en la memoria del controlador, ya que dichos datos serán usados para validar la autenticidad de aquellas aplicaciones cliente que intenten conectarse al controlador.

De igual forma, no debe olvidarse que es mucho más recomendable enviar absolutamente todos los datos encriptados en todos los niveles de conversación entre la aplicación cliente y el controlador de conexiones, esto es con el fin de mantener un nivel apropiado de seguridad en el modelo propuesto. Es recomendable encriptar las claves de usuario y contraseña que viajan de las aplicaciones cliente hacia el controlador a fin de disminuir la probabilidad de fraudes en la red.

Como se está determinando la validación por medio de una comparación entre las respectivas claves del controlador y las aplicaciones cliente, se hace necesario nunca prescindir de dichas claves de usuario y contraseña en el controlador aún cuando la fuente de datos a la que el controlador haga conexión no solicite dichos datos. Esto es necesario porque una vez arrancadas las conexiones desde el controlador, las aplicaciones podrían

intentar conectarse y para dichas aplicaciones es necesario realizar un proceso de validación.

A continuación se enumera la secuencia de una conversación común Cliente/Servidor en base al modelo propuesto:

1. **Una aplicación realiza una petición de conexión a la fuente de datos.** Una aplicación solicita una conexión del controlador mediante un socket, para lo cual es necesaria la definición de tres elementos: la dirección IP del lugar donde se encuentra el socket en la red -la dirección del controlador-, el tipo de protocolo a usar –en este caso TCP/IP- y el puerto a través del cual se va a conectar.

2. **Validación de acceso en el controlador.** En este caso, como el controlador esta supliendo al gestor de la fuente de datos, se optimiza el hecho de acceder a fuentes de datos que no posean un gestor propio, mediante esta sustitución se brinda seguridad al modelo.

   Cuando una aplicación cliente intente conectarse al controlador deberá enviar una clave de usuario y su contraseña, el controlador validará estos datos contra su propia clave de usuario y contraseña, si esta validación es exitosa, el controlador verifica qué conexión tiene disponible para atender a la aplicación concurrente. Ante una petición de un cliente que no puede ser atendida, el controlador podría:

   2.1 Bloquear al cliente hasta que pueda atenderle, esto significa una forma de atención síncrona.

   2.2 El controlador se apunta la petición y le dice al cliente que le contesta cuando pueda, desbloqueando al cliente (forma de atención asíncrona).

2.3 Contestarle que no puede. El cliente se provee de algún método para validar si la conexión fue exitosa o no, y tener algún algoritmo para controlar el número de intentos de conexión con un tiempo de espera, como intervalo entre cada intento.

3. **Conexión.** Si la validación fue exitosa y existe alguna conexión que pueda atender a la aplicación, el acceso es permitido.

   Dependiendo del estilo como se diseñe la aplicación cliente, se pueden programar funciones o procedimientos similares a los de los modelos convencionales para validar si la conexión fue exitosa o no: manejo de handles y manejo de errores.

4. **La aplicación está conectada a la fuente de datos y solicita datos.** Una vez que la conexión ha sido exitosa, nuestra aplicación inmediatamente solicita una transacción de búsqueda o afectación en los datos, siempre y cuando –como en el modelo actual- el usuario (en caso de que haya sido necesario uno) posea los permisos necesarios para realizar esta operación.

   A partir de este momento la fuente de datos actúa como un servidor de datos y la aplicación se mantendrá conectada al servidor hasta que halla sido atendida por el controlador, el cual, una vez atendida la petición procederá a desconectar a la aplicación cliente para brindar el servicio a otras aplicaciones. A diferencia del modelo actual, en el modelo propuesto, nuestra aplicación cliente no permanecerá conectada hasta que ella misma desee desconectarse, en vez de esto, el controlador procederá a desconectarla una vez que haya sido atendida su solicitud.

5. **Validación de peticiones.** El controlador procede a enviar la sentencia (petición) a la fuente de datos; en caso de cualquier error sintáctico, el controlador deberá ser capaz de detectar que un error ha ocurrido, mismo que enviará a la aplicación cliente como respuesta a su solicitud. La validación sintáctica no es validada directamente por el

controlador, sino que es cedida a la fuente de datos –por las razones ya expuestas acerca de la diferencia en las versiones de lenguajes en las fuentes de datos-, el controlador solo espera el resultado de la petición así como la noticia acerca del éxito de la petición.

6. **Ejecución de peticiones.** Una vez hecha la validación de la petición, y si ésta fue exitosa, la fuente de datos ejecuta el comando solicitado por el controlador. El tiempo de ejecución por parte de la fuente dependerá de la complejidad de la afectación así como de la velocidad de procesamiento del servidor donde se esté procesando dicha información.

7. **Envío de resultados y desconexión.** Una vez que la fuente de datos ha procesado el requerimiento que le fue enviado, le informa al controlador que la petición fue realizada, el controlador, a su vez, por medio del socket que en ese momento tiene asociado a la aplicación cliente le envía los resultados del requerimiento, cuando el total de los datos le han sido enviados a la aplicación cliente se procede a la desconexión de la aplicación cliente.

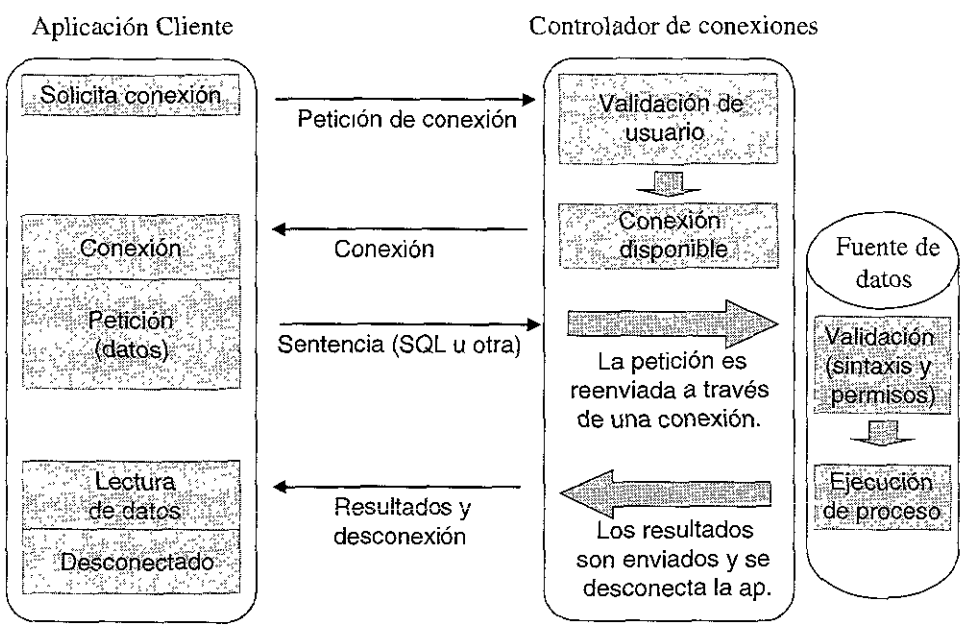



**Figura 5.5 Diagrama de flujo del modelo propuesto.**

Como se puede observar, la secuencia de la conversación Cliente/Servidor entre el modelo actual y el propuesto es similar a excepción de dos puntos, que son de hecho, la base del modelo propuesto:

a) **La validación de seguridad en el controlador.** A diferencia del modelo actual, donde el proceso de validación/conexión a la fuente de datos es sumamente lenta en comparación con el tiempo de procesamiento de datos, en el modelo propuesto el tiempo de validación es casi nulo; esto es debido al hecho de que solo se realiza una comparación de cadenas entre las claves de usuario y contraseñas contra las claves que el controlador usó para abrir las conexiones a la fuente de datos.

b) **La desconexión automática por parte del controlador de las aplicaciones.** Una vez enviada la respuesta por parte del controlador hacia la aplicación cliente, el mismo controlador se encarga de desconectar a la aplicación. Esto proporciona al modelo propuesto cierto dinamismo al no mantener clientes ligados de manera permanente al servidor (controlador), además de –como se ha dicho- mejorar la calidad del tráfico en la red.

### *5.2.2 Ventajas y desventajas*

La implantación del modelo Cliente/Servidor propuesto para acceso a fuentes de datos presenta una serie de características que pueden considerarse como ventajas, ya que permiten economizar de manera general tiempos en la comunicación Cliente/Servidor, así como un buen manejo en general de las aplicaciones. Sin embargo también presenta ciertas desventajas que a continuación se observan.

*Ventajas:*

- **Optimización de tiempo de acceso a bases de datos** (de más del 50%), esto originando reducción de costos en cuanto al tiempo horas/hombre, tiempo de procesamiento tanto del cliente como del servidor, tiempo de uso de la red (que en caso de ser satelital el costo en dinero es sumamente elevado), y, en general tiempo de transacción.

- **Administración de conexiones y puertos**, esto es mediante la asignación de un número máximo de conexiones así como su distribución entre los clientes potenciales de datos y la asignación de puertos para la comunicación Cliente/Servidor a través del medio.

- **La generalidad de su uso**, mediante la construcción de un modelo que ha sido construido sobre la plataforma ODBC, lo cual garantiza su implantación en diferentes bases de datos tales como Informix, SQL Server, Access, Oracle, SQLBase, Sybase, Paradox; así como manejadores ISAM tales como DBase, Excel, Foxpro e incluso archivos de texto y bibliotecas AS400, lo que le permite ser un modelo abierto.

- **Estandarización.** La tendencia actual es unificar el acceso a bases de datos mediante el estándar ODBC, por tal motivo las empresas desarrolladoras de manejadores de bases de datos cada vez están mas convencidas de la necesidad de ésta unificación.

- **Facilidad de uso.** El controlador de conexiones encapsula recursos que se acceden a través de operaciones que invocan los clientes por medio de sockets. Esto facilita la implantación de aplicaciones Controlador/Cliente de datos escondiendo la complejidad del modelo permitiendo a los desarrolladores centrarse sobre sus aplicaciones.

- **Es un modelo multiusuario.** El uso de *threads* en el controlador permite el acceso concurrente de varios clientes, esto significa, tener un servidor multiusuario con la capacidad de atender de manera simultánea a varios clientes. Así queda establecido que las operaciones de los clientes no deben interferirse entre si y cada una es tratada de manera independiente, respetando así una de las características del modelo Cliente/Servidor para acceso a datos actual. El efecto de realizar una operación pedida por un cliente no debe verse interferido por las operaciones pedidas por otros clientes.

- **Es un sistema abierto.** Puede y, de hecho, opera bajo sistemas abiertos, lo que significa su capacidad de cambio a plataformas con un mínimo de problemas.

*Desventajas:*

- El **uso imprescindible de una plataforma ODBC** que soporte el modelo aquí expuesto. Bien se podría prescindir de la plataforma ODBC, pero esto ocasionaría tener un modelo que solo sirviera para acceder a una fuente de datos en particular, si se deseara acceder a otra fuente de datos, lo mas probable es que se necesitara reprogramar algunas funciones internas del controlador en base requerimientos de la nueva fuente de datos.

- **El uso de sockets**, el cual permite la extensibilidad en el envío de datos por cualquier medio, siempre y cuando se encuentren los nodos cliente y servidor dentro de los protocolos TCP/IP o UDP/IP. Esto limita nuestro modelo para su uso bajo otros protocolos de comunicación.

- **Seguridad.** Los expertos en seguridad informática mantienen, que bajo Cliente/Servidor la seguridad en el acceso a datos es sencilla de romper. Pero la respuesta a esta aseveración es que la seguridad está sujeta al diseño y medidas de

seguridad que se incorporen al ambiente, y a las aplicaciones desarrolladas para operar bajo la arquitectura Cliente/Servidor.

- **Uso de un controlador por cada fuente de datos a la que se desee acceder.** Debido a que el diseño del controlador fue hecho para acceder solo a una fuente de datos, si se tuvieran diferentes fuentes de datos, se tendrían que correr tantas instancias del mismo controlador como fuentes de datos se deseara acceder a través del modelo propuesto.

  Otra solución para este problema sería rediseñar el controlador a fin de que este permita adicionar nuevas fuentes de datos en tiempo de ejecución, con sus parámetros correspondientes (número de conexiones, puerto, usuario, etc.), es decir, multidimensionarlo a fin de soportar el control de múltiples fuentes de datos a la vez.

# 6. DESARROLLO Y LIBERACION DEL MODELO PROPUESTO

"Nuestra capacidad de creación de software de computación no está a la par con la evolución del hardware. Se necesita una revolución industrial en el software.

Es probable que esta revolución provenga de las técnicas orientadas a objetos, combinadas con herramientas CASE, generadores de código, programación visual y desarrollo basado en depósitos (repository-based development). La meta es maximizar la reutilización de código, así como construir y almacenar objetos complejos."

La implantación de un modelo similar al propuesto en el capítulo anterior requiere de ciertas características mínimas básicas para su óptima funcionalidad; así mismo es necesario tratar de englobar o "encapsular" las funciones inherentes al modelo, esto con el propósito de simplificar su desarrollo y brindar modularidad a cada una de sus partes.

En el presente capítulo se desarrollarán las partes necesarias mínimas para la puesta a punto de un controlador de conexiones, así como su respectiva aplicación cliente.

## 6.1 Conceptos básicos

Para entrar a detalle en el desarrollo del controlador, es necesario primeramente dejar por asentado ciertos conceptos que serán imprescindibles para el desarrollo del modelo.

Una de las preocupaciones actuales más urgentes de la industria de la computación es la de crear software y sistemas corporativos más pronto y de más bajo costo.

Para hacer un buen uso del poder cada vez mayor de las computadoras, se necesita un software de mayor complejidad. Aparte de más complejo, también es necesario que dicho software sea más confiable. La alta calidad es esencial en el desarrollo de software, ya que una calidad pobre es un desperdicio de dinero y tiempo.

Las técnicas orientadas a objetos permiten que el software se construya a partir de objetos de comportamiento específico. Los propios objetos se pueden construir a partir de otros, que a su vez pueden estar transformados por otros objetos. Esto nos recuerda una maquinaria compleja, construida por partes, subpartes, sub-subpartes, etc.

Con el creciente progreso en el campo del desarrollo de software, todas las cosas comienzan ha ser vistas desde la perspectiva de la orientación a objetos. Ahora, casi "automágicamente", todas las cosas son objetos. Los controles también lo son, y como tales, corresponden a instancias de una clase definida en alguna parte, que "encapsula" su estructura y los métodos que le son aplicables.

## Definiciones básicas

*Objeto*

Las personas tienen una idea clara de lo que es un objeto: conceptos adquiridos que nos permiten sentir y razonar acerca de las cosas del mundo. Un objeto podría ser real o abstracto, por ejemplo una organización, una factura, una figura en un libro, una pantalla de usuario, un avión, un vuelo de avión, etc.

Un objeto es cualquier cosa, real o abstracta. Este es creado como una instancia de un tipo de objeto. Cada objeto tiene una identidad única que los distingue e independiente de cualquiera de sus características. Cada objeto ofrece una o más operaciones.

Cuando las personas preguntan a un experto en objetos que es OBJETO, muy posiblemente se encontrarán con la siguiente respuesta: "Un objeto es cualquier cosa que usted vea, un libro, una persona, una puerta, una computadora, etc."

**Figura 6.1 Simples objetos**

Probablemente cualquier persona puede quedar mas confundida que antes de la respuesta, puesto que un libro, una persona, una puerta, una computadora realmente no aclara nada sobre el profundo concepto que hay detrás de lo dicho anteriormente.

Como se explicó un objeto es cualquier cosa, pues bien, si es cualquier cosa imaginemos un televisor. En este caso es algo real, pero nos pudimos haber imaginado algo abstracto (Ej. Un número complejo). Si tenemos dos televisores exactamente idénticos, estos serán dos objetos (no necesitamos distinguirlos con un código o una marca, no, solo son dos objetos diferentes). Cada uno en distintos lugares del espacio y no importa lo que le suceda al otro seguirán siendo dos objetos independientes. Es todo lo que necesitamos saber para comprender que un televisor es un objeto.

Internamente el televisor tiene algunos circuitos básicos de funcionamiento, un receptor, un amplificador, un filtro, etc. posiblemente no sabemos como están construidos estos circuitos, desde el punto de vista de objetos, esto está *encapsulado*. Solo sabemos de la existencia de estos porque al encender el televisor, este da una imagen –en caso de que el televisor funcione-, podemos sintonizar programas, etc. Esta es la forma de enviar un mensaje a los circuitos para que estos se ajusten y dejen pasar únicamente la señal del programa deseado. La comunicación con los objetos se hace a través de mensajes.

El modelo del televisor (con sus circuitos) puede ser utilizado para diseñar un nuevo aparato de recepción, el principio de funcionamiento de los televisores es único, de manera que puede haber una reutilización de su tecnología para crear nuevos aparatos.

Por último un televisor puede ser usado como monitor de computadora, proyector de cine de un VHS, etc. en término de objetos esto se conoce como *polimorfismo*. Pero algún experto leyendo esto dirá, el televisor como objeto no es polimórfico, el concepto de polimorfismo se aplica únicamente a las operaciones del objeto.

*Análisis y diseño orientado a objetos.*

En el análisis y diseño orientados a objetos (OO), interesa el comportamiento del objeto. Si se construye software, los módulos de software OO se basan en los tipos de objetos. El software que implanta el objeto contiene estructuras de datos y operaciones que expresan dicho comportamiento. Las operaciones se codifican como métodos. La representación en software OO del objeto es entonces una colección de tipos de datos y objetos.

Entonces, dentro del software orientado a objetos, un objeto es cualquier cosa, real o abstracta, acerca de la cual almacenamos datos y los métodos que controlan dichos datos.

Un objeto puede estar compuesto por otros objetos. Estos últimos a su vez también pueden estar compuestos por otros objetos. Esta intrincada estructura es la que permite construir objetos muy complejos.

*Tipos de Objeto, métodos y propiedades*

Los conceptos que poseemos se aplican a tipos determinados de objetos. Por ejemplo, empleado se aplica a los objetos que son personas empleadas por alguna organización.

Algunas instancias de empleado podrían ser Juan Pérez, José Martínez, etc. En el análisis orientado a objetos, estos conceptos se llaman tipos de objetos; las instancias se llaman objetos. Así, un tipo de objeto es una categoría de objeto, mientras que un objeto es una instancia de un tipo de objeto.

En el mundo de las bases de datos existen los tipos de entidad, por ejemplo cliente o empleado. Existen muchas instancias de cada tipo de entidad (como Juan Pérez o José Martínez para el tipo de entidad empleado). Del mismo modo, en OO se define tipos de objetos e instancias de tipo de objeto.

Sin embargo, el término objeto tiene diferencias fundamentales con el término entidad, ya que la entidad sólo se refiere a los datos, mientras que objeto se refiere a los datos y a los métodos mediante los cuales se controlan a los propios datos. En OO, la estructura de datos y los métodos de cada tipo de objeto se manejan juntos. No se puede tener acceso o control de la estructura de datos excepto mediante los métodos que forman parte del tipo de objeto.

Existe otra forma de poder acceder a los datos no usando métodos, esto es a través de *propiedades*. Una propiedad es la representación del dato de un objeto que ha sido expuesto, existen propiedades que solo son de lectura y otras escritura, también existen

propiedades que son tanto de lectura como de escritura; esto se refiere al hecho de poder modificar el dato (en el caso de escritura) o sólo poder leerlo (lectura).

Los métodos especifican la forma en que se controlan los datos de un objeto. Los métodos en un tipo de objeto sólo hacen referencia a la estructura de datos de ese tipo de objeto. No deben tener acceso directo a las estructuras de datos de otros objetos. Para utilizar la estructura de datos de otro objeto, deben enviar un mensaje a éste. El tipo de objeto empaca juntos los tipos de datos y su comportamiento.

Un objeto entonces es una cosa cuyas partes están representadas por tipos de datos y su comportamiento por métodos.

Un método asociado con el tipo de objeto factura podría ser aquel que calcule el total de la factura. Otro podría transmitir la factura a un cliente. Otro podría verificar de manera periódica si la factura ha sido pagada y, en caso contrario, añadir cierta tasa de interés.

Una propiedad podría ser el número de factura, el cual no puede ser modificado porque es asignado automáticamente por el sistema y por lo tanto es una propiedad de lectura. La tasa de interés podría ser una propiedad de lectura y escritura porque puede tanto leerse (para saber la tasa de interés que está siendo aplicada) como reescribirse (en caso de un aumento o disminución el la tasa de interés).

*Encapsulado*

Define el empaquetamiento de operaciones y datos dentro del objeto, tal que los datos son únicamente accesibles a través de su interfaz. El empaque conjunto de datos y métodos se llama encapsulado. El objeto esconde sus datos de los demás objetos y permite el acceso a los datos mediante sus propios métodos. Esto recibe el nombre de ocultamiento de información. El encapsulamiento evita la corrupción de los datos de un objeto. Si todos los programas pudieran tener acceso a los datos de cualquier forma que quisieran los usuarios, los datos se podrían corromper o utilizar de mala manera. El encapsulado protege los datos del uso arbitrario y no pretendido.

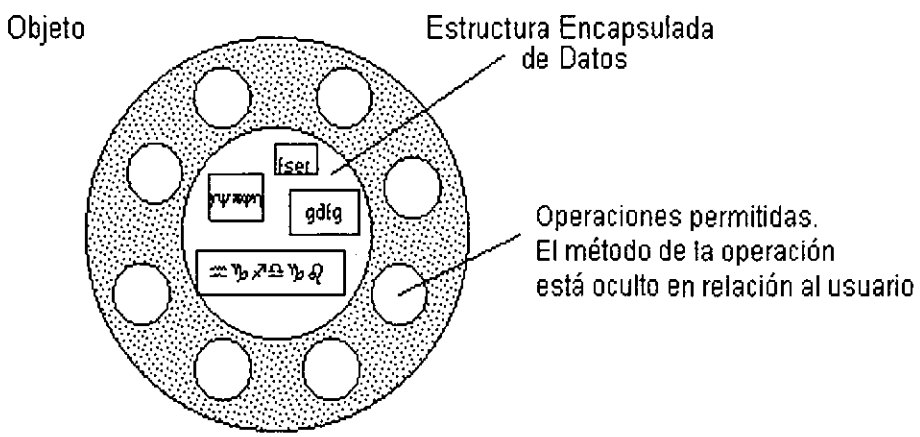



**Figura 6.2 Encapsulamiento de datos**

El encapsulado oculta los detalles de su implantación interna a los usuarios de un objeto. Los usuarios se dan cuenta de las operaciones que puede solicitar del objeto, pero desconocen los detalles de cómo se lleva a cabo la operación. Todos los detalles específicos de los datos del objeto y la codificación de sus operaciones están fuera del alcance del usuario. De igual forma, solo son expuestos aquellos datos que el usuario puede modificar a través de propiedades.

Así, encapsulado es el resultado (o acto) de ocultar los detalles de implantación de un objeto respecto de su usuario.

El encapsulado, al separar el comportamiento del objeto de su implantación, permite la modificación de ésta sin que se tengan que modificar las aplicaciones que lo utilizan.

*Mensajes y comunicación entre objetos*

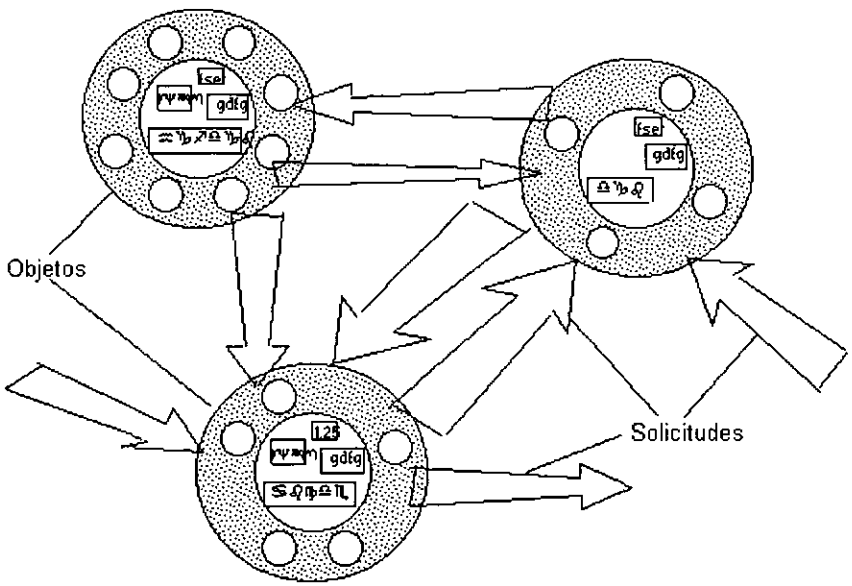



**Figura 6.3 La comunicación entre objetos**

En sistemas orientados a objetos, toma la forma de un envío de requerimientos por parte de un objeto a otros objetos.

Para que un objeto haga algo, se le envía una solicitud. Esta hace que se produzca una operación. La operación ejecuta el método apropiado y, de manera opcional, produce una respuesta. El mensaje que constituye la solicitud contiene el nombre del objeto, el nombre de una operación y, a veces, un grupo de parámetros.

La programación orientada a objetos es una forma de diseño modular en la que con frecuencia el mundo se piensa en términos de objetos, operaciones, métodos y mensajes que se transfieren entre tales objetos. Un mensaje es una solicitud para que se lleve a cabo la operación indicada y se produzca el resultado.

Los objetos pueden ser muy complejos, puesto que pueden contener muchos subobjetos, éstos a su vez pueden contener otros, etc. La persona que utilice el objeto no tiene que conocer su complejidad interna, sino la forma de comunicarse con él y la forma en que responde.

*Clase*

El término clase se refiere a la implantación en software de un tipo de objeto. El tipo de objeto es una noción de concepto. Especifica una familia de objetos sin estipular la forma en que se implanten. Los tipos de objetos se especifican durante el análisis OO.

Así, una clase es una implantación de un tipo de objeto. Especifica una estructura de datos y los métodos operativos permisibles que se aplican a cada uno de sus objetos.

*Herencia*

Un tipo de objeto de alto nivel puede especializarse en tipos de objeto de bajo nivel. Un tipo de objeto puede tener subtipos. Por ejemplo, el tipo de objeto persona puede tener subtipos estudiante y empleado. A su vez, el tipo de objeto estudiante puede tener como subtipo estudiante de pregrado y estudiante de posgrado, mientras que empleado puede tener como subtipo a académico y administrativo. Existe de este modo una jerarquía de tipos, subtipos, subsubtipos, etc.

Una clase implanta el tipo de objeto. Una subclase hereda propiedades de su clase padre; una sub-subclase hereda propiedades de las subclases; etc. Una subclase puede heredar la estructura de datos y los métodos, o algunos de los métodos, de su superclase. También tiene sus métodos e incluso tipos de datos propios.

*Abstracción*

Define la relación entre un grupo de objetos tales que, un tipo de objeto representa un conjunto de características las cuales son compartidas por otros tipos de objetos.

*Reusabilidad*

Define la habilidad de reusar objetos y clases de objetos con la implantación de un sistema.

## 6.2 El servidor de datos

El servidor de datos –que es el controlador- es la parte del modelo que se encargará de acceder a la fuente de datos y atender, por medio de sockets a los clientes de datos (las aplicaciones cliente).

A continuación, se observa la lista de las características que debe poseer el servidor, las cuales fueron expuestas en el capítulo anterior.

- Ser multiusuario: capacidad de atender múltiples requerimientos a un mismo tiempo.

- Poder conectarse a cualquier fuente de datos, siempre y cuando dicha fuente de datos posea un manejador especifico para ODBC.

- Deberá ser capaz de soportar errores, producto de peticiones erróneas por parte de las aplicaciones cliente.

- Deberá desconectar a las aplicaciones cliente, una vez que su solicitud halla sido atendida.

- Deberá trabajar bajo el protocolo TCP/IP.

- Se deberá comunicar con los clientes a través de sockets.

- El número de puerto lógico para la comunicación por sockets podrá ser especificado por el administrador del controlador.

- El número de conexiones podrá ser especificado de igual forma por el administrador.

- El controlador se encargará de enviar la petición a la fuente de datos de manera asíncrona, es decir, realiza el envío de la petición y continua atendiendo otras requisiciones de las aplicaciones.

- El controlador de conexiones será el encargado de validar la cuestión de seguridad mediante la validación de la clave de usuario enviada por la aplicación cliente.

- El controlador delegará la responsabilidad de la cuestión sintáctica a la fuente de datos, esto por razones de varianza en las distintas fuentes de datos.

### *6.2.1 Propiedades, métodos y eventos*

A fin de simplificar la construcción del servidor de datos –el controlador-, se detallará su diseño en base a un análisis orientado a objetos, es decir, usando propiedades, métodos y eventos; encapsulando cada una de las funciones que el servidor realizará las cuales serán mostradas a través de métodos. Denominaremos "controlador" al objeto principal que fungirá como el controlador de conexiones.

*Propiedades*

**DSN**. Propiedad de lectura/escritura, permitirá asignar u obtener el Data Source Name de la fuente de datos a la que el controlador se conectará.

**Usuario**. Propiedad de lectura/escritura, permitirá asignar u obtener el nombre del usuario que el controlador usará para abrir conexiones, es decir, para registrarse en la fuente de datos.

**Password**. Propiedad de lectura/escritura, permitirá asignar u obtener el password asociado a la propiedad **Usuario** en la apertura de conexiones.

**Conexiones**. Propiedad de lectura/escritura, esta propiedad será utilizada para especificarle al controlador el número de conexiones que deberá abrir.

**Puerto**. Propiedad de lectura/escritura, esta propiedad permitirá asignar el número de puerto lógico mediante el cual, las aplicaciones cliente podrán ser atendidas por el controlador.

En base a lo anterior, la primer tarea del controlador debe ser la asignación –y validación- de las propiedades anteriores, si alguna de ellas es nula el controlador no debe comenzar el proceso de atención de aplicaciones cliente. A continuación se observa una pantalla ejemplo donde el controlador solicita los datos necesarios para su arranque al usuario/administrador:

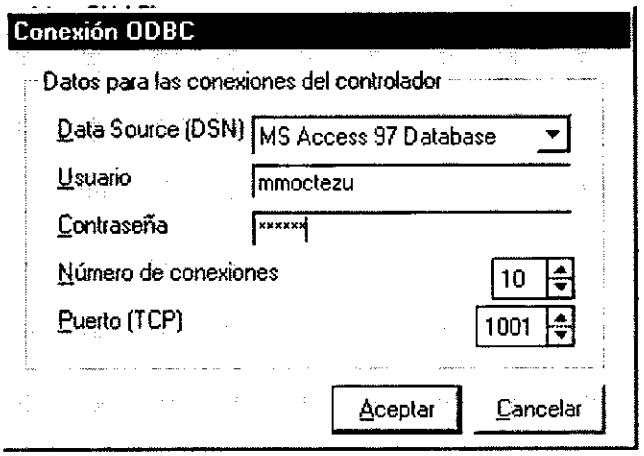



**Figura 6.4 Las propiedades iniciales del controlador**

Se sugiere inicializar estas propiedades con estándares predefinidos (defaults). Por ejemplo utilizar el usuario de sesión de red, 10 conexiones, puerto 1001, etc.

*Métodos y eventos*

**CreaConexiones.** Una vez asignadas las propiedades mencionadas, el controlador se conectará a la fuente de datos seleccionada, abriendo las *n* conexiones solicitadas, dato que se encuentra almacenado en la propiedad **conexiones**; de igual forma, el controlador creará *n* sockets, los cuales estarán disponibles para atender a las aplicaciones cliente. Deberá

existir un socket más, al que llamaremos "listen socket", el cual será el encargado de estar "escuchando" cualquier requisición de los clientes, dicho socket usará como puerto el número asignado a la propiedad **puerto**. A continuación se muestra el código fuente correspondiente a **CreaConexiones**, obsérvese la variable `tcpServer`, dicha variable es una instancia de la clase `WinSock`, y es el objeto encargado de crear los sockets. Veamos este método:

```
Private rdoEn As rdoEnvironment

'Se instancia a tcpServer como un objeto WinSock
Private tcpServer As New WinSock

'Arreglos dinámicos en base al número de conexiones
Private rdoSvrConnection() As New rdoConnection
Private rdoSvrResultSet() As rdoResultset
Private bExecuting() As Boolean

Public Sub CreaConexiones()
  Dim i as Integer

  'Si ocurre cualquier error
  On Error GoTo err_making:

  Set rdoEn = rdoEnvironments(0)
  'Redimensionar las estructuras
  ReDim rdoSvrConnection(Conexiones)
  ReDim rdoSvrResultSet(Conexiones)
  ReDim bExecuting(Conexiones)
  ReDim tRequested(Conexiones)
  ReDim tcpServer(Conexiones)
  For i = 0 To Conexiones - 1
```

```
        'sección que carga y habilita el socket para cada conexión
        Load tcpServer(i)
        tcpServer(i).LocalPort = 0

        'sección que abre cada conexión
        With rdoSvrConnection(i)
            .Connect = "UID=" + Usuario + ";PWD=" + _
                       Password + "DSN=" + DSN + ";"
            .LoginTimeout = 0
            .EstablishConnection rdDriverNoPrompt, False
        End With
        'Se agrega cada conexión a la colección rdoConnections:
        rdoEn.rdoConnections.Add rdoSvrConnection(i)
    Next i

    'Se carga el listen-socket que atenderá las llamadas
    Load tcpServer(Conexiones)
    If tcpServer(Conexiones).State <> sckClosed Then _
        tcpServer(Conexiones).Close

    'Se le asigna el número de puerto y se pone en modo "Listen"
    tcpServer(Conexiones).LocalPort = Puerto
    tcpServer(Conexiones).Listen
    'se habilita el timer de requisiciones para las apps cliente
    RequestTimer.Enabled = True
    Exit Sub

'En caso de error:
err_making:
    Dim Response As Integer
    Beep
```

```
  Response = MsgBox("Ha ocurrido un error: " & _
         Err.Description & ". Desea Continuar ?", _
         vbYesNo + vbCritical + vbDefaultButton2, Me.Caption)
  If Response = vbNo Then
      End
  Else
      Resume
  End If
End Sub
```

**CierraConexiones**. Este método es la contraparte del método anterior: se encarga de cerrar las conexiones a la fuente de datos y descargar –o destruir- cada socket que fue creado. Veamos:

```
Public Sub CierraConexiones()
  'cierra el listen-socket
  Unload tcpServer(Conexiones)

  'cierra los demás sockets y las conexiones
  For i = Conexiones To 1 Step -1
      'Cierra cada conexión
      rdoEn.rdoConnections(i - 1).Close
      'se cierra y descarga cada socket
      tcpServer(Conexiones - i).Close
      Unload tcpServer(Conexiones - i)
  Next i

  'se deshabilita el timer de requisiciones
  RequestTimer.Enabled = False
End Sub
```

Existen eventos, los cuales son los encargados de atender cada llamada de las aplicaciones cliente y recibir los datos enviados desde la aplicación cliente; estos eventos son **ConnectionRequest** y **DataArrival**, los cuales –como se mencionó en el capítulo cuatro- son métodos provistos directamente por el control WinSock y son disparados de manera automática cuando una aplicación cliente invoca su propio método **Connect** y cuando la aplicación cliente envía alguna petición y el controlador procede a leer dicha petición en base a una estructura "sentencia_SQL | aplicación_nombre | usuario | password", respectivamente. Veamos:

```
'evento que se dispara al recibir una requisición de conexión
Private Sub tcpServer_ConnectionRequest(Index As Integer, _
  ByVal requestID As Long)
  'Busca la primer conexión disponible para su posible conexión
  For i = 0 To MaxConnections - 1
      If tcpServer(i).State <> sckConnected Then
          tcpServer(i).Close
          'La conexión de la aplicación cliente es aceptada
          tcpServer(i).Accept requestID
          Exit For
      End If
  Next i
End Sub
Private Sub tcpServer_DataArrival(Index As Integer, _
  ByVal bytesTotal As Long)
  Dim strData As String, SQLStatement As String, _
      AppName As String, User As String, Password As String
  Dim nParam As Integer
    'Almacena los datos recibidos desde el cliente en strData, _
    los datos vienen encriptados y con la siguiente secuencia: _
    Sentencia_SQL|Nombre_de_Aplicación_Cliente|Usuario|Password:
```

```
  tcpServer(Index).GetData strData
  'Los datos son desencriptados usando la función uncrypt la _
   cual puede ser implantada fácilmente
  strData = Trim(uncrypt(strData))

  nParam = 0
  SQLStatement = ""
  AppName = ""
  User = ""
  Password = ""
  'Los datos vienen separados por el caracter "|"
  For i = 1 To Len(strData)
      If Mid(strData, i, 1) = "|" Then
          nParam = nParam + 1
          GoTo Continue
      End If

      Select Case nParam
          Case 0
              SQLStatement = SQLStatement + Mid(strData, i, 1)
          Case 1
              AppName = AppName + Mid(strData, i, 1)
          Case 2
              User = User + Mid(strData, i, 1)
          Case 3
              Password = Password + Mid(strData, i, 1)
      End Select
Continue:
  Next i

  On Error GoTo SendError:
```

```
    'si el usuario o password no coinciden se envía error al _
     cliente mediante el método Err.Raise e inmediatamente se _
     procede a su desconexión (SendError):

    If UCase(User) <> UCase(strUser) Or _
       UCase(Password) <> UCase(strPassword) Then _
        Err.Raise 28000, , "Login Failed"

    'Se asigna el resultado de la petición al objeto _
     rdoSvrResultSet(Index) mediante el método OpenResultset _
     nótese que es invocado de forma asíncrona (rdAsyncEnable):

    Set rdoSvrResultSet(Index) = _
    rdoEn.rdoConnections(Index).OpenResultset(SQLStatement, _
    rdOpenDynamic, rdConcurValues, rdAsyncEnable)

SendError:
    If Err.Number <> 0 Then      'Ocurrió un error
        tcpServer(Index).SendData crypt("ERROR," + _
           Trim(Str(Err.Number)) + "," + Trim(Err.Description))
        tcpServer(Index).Close
    Else
         'Se levanta una bandera booleana la cual indica que _
          la conexión Index está actualmente atendiendo una _
          petición:
        bExecuting(Index) = True
    End If
End Sub
```

Por último se encuentra el método **Procesa_Peticiones**. Este método es realmente el motor del controlador de conexiones, dicho método debe ser disparado mediante algún evento que se pueda controlar en base a cierto lapso de tiempo, es decir, cada *t* tiempo.

**Procesa_Peticiones** debe realizar un chequeo para determinar que peticiones enviadas por las aplicaciones cliente han sido atendidas completamente por la fuente de datos, una vez que cierta petición halla sido atendida por la fuente de datos **Procesa_Peticiones** debe enviar la respectiva respuesta a la aplicación cliente.

Existen diferentes formas para lograr que **Procesa_Peticiones** sea disparado cada *t* lapso de tiempo, véase por ejemplo el siguiente pseudocódigo:

```
while true do
   Procesa_Peticiones
   sleep t_tiempo
done
```

En el presente proyecto, en vez del pseudocódigo anterior se usará un *timer*, un timer es simplemente un control que puede ejecutar cierto código en intervalos regulares de tiempo, esto mediante el disparo de un evento propio, dicho control posee –como objeto que es- una propiedad donde se puede especificar el intervalo de tiempo en el que dicho evento será disparado.

Supongamos que tenemos un objeto llamado `MyTimer` proveniente de la clase `timer` y le asignamos el intervalo de tiempo de un segundo (en milisegundos):

```
MyTimer.Interval = 1000    'El intervalo es cada segundo
```

MyTimer posee un evento llamado timer, en dicho evento se puede escribir el código que se pretende ejecutar cada *t* tiempo:

```
Private Sub MyTimer_timer()
     Procesa_Peticiones    'Ejecuta Procesa_Peticiones
End Sub
```

Como se mencionó, **Procesa_Peticiones** es el motor del controlador de conexiones ya que se encarga de:

- Verificar si la fuente de datos ha terminado de procesar la información de aquellas conexiones que se encuentran atendiendo aplicaciones cliente.

- Garantizar la transmisión total de los datos. Los datos son enviados junto con un valor numérico que indica el número de total de bytes transmitidos por el controlador, dicho dato es concatenado al final del mensaje completo.

- Enviar el resultado (previamente encriptado) de la petición a la aplicación cliente cuando la conexión haya sido atendida por la fuente de datos. **Procesa_Peticiones** enviará los datos a la aplicación cliente en base a la siguiente estructura:

| | | | |
|---|---|---|---|
| Nombre de columna 1 | Nombre de columna 2 | ... | Nombre de columna n |
| Tipo de dato de columna 1 | Tipo de dato de columna 2 | ... | Tipo de dato de columna n |
| Longitud en bytes de columna 1 | Longitud en bytes de columna 2 | ... | Longitud en bytes de columna n |
| Datos de columna 1 en renglón 1 | Datos de columna 2 en renglón 1 | ... | Datos de columna n en renglón 1 |
| Datos de columna 1 en renglón 2 | Datos de columna 2 en renglón 2 | ... | Datos de columna n en renglón 2 |
| ... | ... | ... | ... |
| Datos de columna 1 en renglón n | Datos de columna 2 en renglón n | ... | Datos de columna n en renglón n |

**Tabla 6.1 Estructura de datos enviada por el controlador.**

Los tipos de datos están basados en la siguiente tabla, la cual es provista por el modelo de objetos RDO. Las descripciones en la tabla se toman como se encuentran en su versión original de los archivos de ayuda de RDO:

| Constant | Value | Description |
|---|---|---|
| RdTypeCHAR | 1 | Fixed-length character string. Length set by Size property. |
| rdTypeNUMERIC | 2 | Signed, exact, numeric value with precision p and scale s (1 p 15; 0 s p). |
| rdTypeDECIMAL | 3 | Signed, exact, numeric value with precision p and scale s (1 p 15; 0 s p). |
| rdTypeINTEGER | 4 | Signed, exact numeric value with precision 10, scale 0 (signed: -231 n 231-1; unsigned: 0 n 232-1). |
| rdTypeSMALLINT | 5 | Signed, exact numeric value with precision 5, scale 0 (signed: -32,768 n 32,767, unsigned: 0 n 65,535). |
| rdTypeFLOAT | 6 | Signed, approximate numeric value with mantissa precision 15 (zero or absolute value 10-308 to 10308). |
| rdTypeREAL | 7 | Signed, approximate numeric value with mantissa precision 7 (zero or absolute value 10-38 to 1038). |

| | | |
|---|---|---|
| rdTypeDOUBLE | 8 | Signed, approximate numeric value with mantissa precision 15 (zero or absolute value 10-308 to 10308). |
| rdTypeDATE | 9 | Date — data source dependent. |
| rdTypeTIME | 10 | Time — data source dependent. |
| rdTypeTIMESTAMP | 11 | TimeStamp — data source dependent. |
| rdTypeVARCHAR | 12 | Variable-length character string. Maximum length 255. |
| rdTypeLONGVARCHAR | -1 | Variable-length character string. Maximum length determined by data source. |
| rdTypeBINARY | -2 | Fixed-length binary data. Maximum length 255. |
| rdTypeVARBINARY | -3 | Variable-length binary data. Maximum length 255. |
| rdTypeLONGVARBINARY | -4 | Variable-length binary data. Maximum data source dependent. |
| rdTypeBIGINT | -5 | Signed, exact numeric value with precision 19 (signed) or 20 (unsigned), scale 0; (signed: -263 n 263-1; |

| | | |
|---|---|---|
| | | unsigned: 0 n 264-1). |
| rdTypeTINYINT | -6 | Signed, exact numeric value with precision 3, scale 0; (signed: -128 n 127, unsigned: 0 n 255). |
| RdTypeBIT | -7 | Single binary digit. |

**Tabla 6.2 Tipos de datos soportados por el controlador.**

A continuación, veamos el evento **Procesa_Peticiones**:

```
Private Sub Procesa_Peticiones()
  Dim strResultData As String, strAux As String, _
      strHeader As String
  Dim j As Integer, TipoDato As Integer, nRows As Integer
  Dim tAnswer as Date

  For i = 0 To MaxConnections - 1
      'Verifica si la conexión "i" está siendo ocupada por una _
       aplicación y que su solicitud esté siendo procesada:

      If tcpServer(i).State = sckConnected _
      And bExecuting(i) Then

          'Si la fuente de datos continua procesando dicha _
          solicitud (StillExecuting) entonces el evento _
          permite que se continúe procesando:

          If rdoEn.rdoConnections(i).StillExecuting Then _
```

```
    GoTo Continue  'continua ejecutando query

strResultData = ""
'tiempo de respuesta de fuente de datos
tAnswer = Timer
nRows = 0

On Error Resume Next
'Realiza el barrido de datos en base a la tabla 6.1
Do Until rdoSvrResultSet(i).EOF
   For j = 0 To rdoSvrResultSet(i).rdoColumns.Count - 1
        TipoDato = _
         rdoSvrResultSet(i).rdoColumns.Item(j).Type
        If TipoDato = rdTypeCHAR Or _
           TipoDato = rdTypeVARCHAR Or _
           TipoDato = rdTypeLONGVARCHAR Then
                strResultData = strResultData + _
                 Trim(rdoSvrResultSet(i)(j)) + "|"
        ElseIf TipoDato = rdTypeTIME Or _
               TipoDato = rdTypeDATE Or _
               TipoDato = rdTypeTIMESTAMP Then
                  strResultData = strResultData + _
               Trim(CStr(rdoSvrResultSet(i)(j))) + "|"
        Else   'cualquier otro tipo
                 strResultData = strResultData + _
                 Trim(Str(rdoSvrResultSet(i)(j))) + "|"
        End If
    Next j
    rdoSvrResultSet(i).MoveNext
    nRows = nRows + 1
Loop
'agrega el número de renglones obtenidos
```

```
strResultData = Trim(strResultData) + Trim(Str(nRows))

'se envía tiempo de respuesta y número de columnas
strHeader = Format(tAnswer - tRequested(i), _
    "####0.0000") + "|" + _
    Trim(Str(rdoSvrResultSet(i).rdoColumns.Count))

'se agrega el nombre de cada campo:
For j = 0 To rdoSvrResultSet(i).rdoColumns.Count - 1
    If rdoSvrResultSet(i).rdoColumns.Item(j).Name = ""
    Then
        'garantiza la asignación de nombre por _
         columna cuando éste es nulo:
            strAux = "c" + Trim(Str(j))
    Else
            strAux =  _
            rdoSvrResultSet(i).rdoColumns.Item(j).Name
    End If
    strHeader = strHeader + "|" + strAux
Next j

'se agrega el tipo de dato del campo:
For j = 0 To rdoSvrResultSet(i).rdoColumns.Count - 1
  strHeader = strHeader + "|" + _
  Trim(Str(rdoSvrResultSet(i).rdoColumns.Item(j).Type))
Next j

'se agrega la medida en bytes de cada campo:
For j = 0 To rdoSvrResultSet(i).rdoColumns.Count - 1
  strHeader = strHeader + "|" + _
  Trim(Str(rdoSvrResultSet(i).rdoColumns.Item(j).Size))
Next j
```

```
            'Por último, se concatena al mensaje completo el _
             número total de bytes que se transmitirán _
             (excluyendo este mismo número):

            strHeader = Trim(strHeader) + "|"
            strResultData = strResultData + "|" + _
                Trim(Str(Len(strHeader + strResultData)))

            'envía el resultado a al cliente en paquetes de 8192 _
             bytes -medida que puede ser cambiada-:
            For j = 1 To Len(strHeader + strResultData) Step 8192
                tcpServer(i).SendData crypt(Mid(strHeader + _
                     strResultData, j, j + 8191))
            Next j
            'se libera conexión:
            bExecuting(i) = False
            tcpServer(i).Close
        End If
Continue:
  Next i
End Sub
```

### *6.2.2 Requerimientos y limitaciones*

Para la construcción de un controlador de conexiones como el propuesto, es necesario tener presente ciertos requerimientos mínimos así como las posibles limitaciones que un modelo como el presente presenta.

- Como se mencionó en el apartado *5.2.2 Ventajas y desventajas*, el controlador solo funcionará para aquellas fuentes de datos que posean un manejador para ODBC propio, esto representa una limitación para su uso sólo con aquellas fuentes de datos que en efecto posean su propio manejador de ODBC.

- En cuanto a las conexiones por sockets, se podría incluso tener la comunicación entre las aplicaciones cliente y el servidor usando el protocolo UDP en vez de TCP, dicho protocolo se encuentra disponible para la comunicación por sockets con el control WinSock, pero es importante mencionar que el esquema de conversación podría cambiar, esto debido principalmente al hecho de que para UDP no se requiere una conexión explícita, esto quiere decir que no se hablaría de un esquema Cliente/Servidor sino más bien una conversación de "igual a igual".

- Se puede cambiar la estructura propuesta en la tabla 6.1, siempre y cuando se garantize que las aplicaciones cliente también han sido modificadas para recibir los datos en una forma diferente. De igual forma se podría cambiar los valores asignados para cada tipo de dato (tabla 6.2), aunque no se recomienda esto, debido a las excepciones en tipos de datos que ciertas fuentes pueden presentar, de hecho, es mas garantizable el uso de la propiedad **Type** que el modelo de objetos RDO provee para cada **rdoColumn**.

- En general, cualquier procedimiento aquí propuesto, ya sea método o evento puede ser modificado de acuerdo a las necesidades particulares de cada aplicación con el

propósito de su mejora. El modelo aquí propuesto es mas bien un modelo "estándar" que posee un mínimo de seguridad en cuanto a datos, sintaxis, autentificación, etc., se recomienda optimizar los esquemas de encriptación de datos así como de autentificación de las aplicaciones cliente.

## 6.3 El cliente de datos

El cliente de datos es aquella aplicación cliente que desea manejar y procesar datos obtenidos de la fuente de datos a través del controlador. El cliente de datos deberá ser capaz de "entender" la estructura de datos enviada por el controlador como respuesta a su petición.

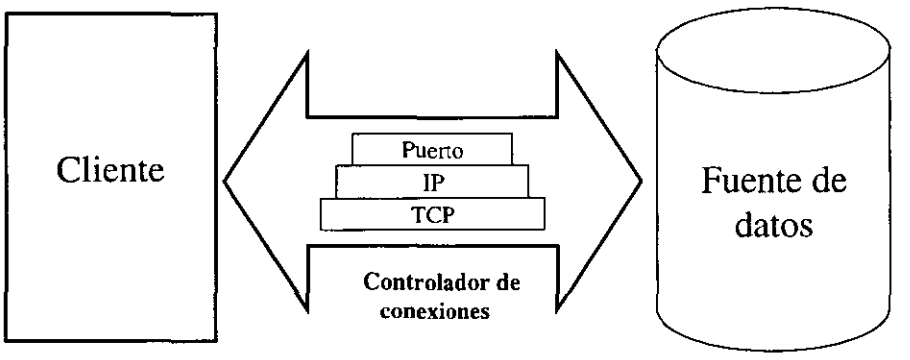



**Figura 6.6 La aplicacion cliente accesando datos**

Como se mencionó en el capítulo anterior, el cliente de datos, conociendo la dirección IP del controlador, el puerto lógico de comunicación, una clave de usuario y su contraseña podrá realizar peticiones al controlador, el cual se encargará de remitir dicha petición a la

fuente de datos, una vez dada la respuesta por la fuente de datos, el controlador reenviará dicha respuesta al cliente y procederá a realizar su desconexión.

### *6.3.1 Propiedades, métodos y eventos*

Siguiendo la metodología aplicada para el servidor de datos, el cliente de datos será diseñado de la misma manera, es decir, con propiedades, métodos y eventos. La implantación del cliente de datos es más sencilla comparada con la implantación del servidor.

*Propiedades*

**Usuario**. Propiedad de lectura/escritura, permitirá asignar u obtener el nombre del usuario que la aplicación cliente usará y enviará al controlador cada vez que pretenda obtener datos a través del controlador.

**Password**. Propiedad de lectura/escritura, permitirá asignar u obtener el password asociado a la propiedad **usuario**, ambas propiedades serán validadas en el controlador de conexiones.

**HostRemoto**. Esta propiedad especificará la dirección IP donde se encuentra corriendo el controlador de conexiones. Esta propiedad mapea directamente la propiedad **RemoteHost** provista por el control Winsock (capítulo 4).

**PuertoRemoto**. Esta propiedad va acompañada con la anterior y le especificará a la aplicación cliente el número de puerto lógico que el controlador de conexiones está usando para la conversación Cliente/Servidor. Esta propiedad mapea de manera similar a la propiedad **RemotePort** también provista por Winsock.

*Métodos y eventos*

De manera análoga al controlador de conexiones, la aplicación cliente deberá tener sendos métodos **CreaConexion** y **CierraConexión** los cuales controlan al objeto `tcpClient` que es una instancia de la clase `WinSock`. El método **CierraConexion** que aquí se presenta no será usado por la aplicación cliente ya que será el mismo controlador de conexiones el encargado de realizar esta tarea una vez atendida la petición de la aplicación cliente, solo es mostrado de manera informacional.

```
Public Sub CreaConexion()
  tcpClient.RemoteHost = HostRemoto
  tcpClient.RemotePort = PuertoRemoto
  tcpClient.Bind 0, tcpClient.LocalIP
  tcpClient.Close
  'Se realiza la conexión:
  tcpClient.Connect
End Sub
```

```
Public Sub CierraConexion()
  tcpClient.Close
End Sub
```

Ahora veamos el método **EnviaDatos**, el cual se encarga de enviar los datos al controlador de forma encriptada y en base al orden de parámetros establecido en el evento **DataArrival** del controlador:

`Sentencia_SQL | Nombre_de_Aplicación_Cliente | Usuario | Password`

```
Private Sub EnviaDatos(SQLStatement As String)
        tcpClient.SendData crypt(SQLStatement + "|" + _
                App.Name + "|" + Usuario + "|" + Password)
End Sub
```

Como se mencionó en el capítulo anterior, es conveniente encapsular los métodos **CreaConexión y EnviaDatos** en un solo procedimiento a fin de agilizar la conversación Cliente/Servidor, por ejemplo:

```
Public Sub Ejecuta(SQLStatement As String)
    CreaConexion
    EnviaDatos(SQLStatement)
End Sub
```

De esta forma, el método `Ejecuta` se encargará de establecer la conexión con el controlador y enviar la petición `SQLStatement`.

Al igual que el controlador de conexiones, el cliente deberá tener su propio evento **DataArrival** el cual se dispara cuando la requisición de la aplicación cliente ha sido atendida por el controlador y la respuesta ha sido enviada. Nótese, dentro del evento que, al recibir los datos, se valida si la respuesta que se ha recibido es un mensaje de "ERROR" o se trata de datos correctos, caso en el cual se procede a reconstruir dichos datos mediante el

procedimiento interno **ReconstruyeResp**. También nótese la comparación que se realiza para determinar si el total de los datos enviados por el controlador han sido leídos, dichos datos se van concatenando en strData.

```
Private Sub tcpClient_DataArrival(ByVal bytesTotal As Long)
  Dim strDat As String
  Dim i As Long

  tcpClient.GetData strDat
  strData = strData + uncrypt(strDat)

  If UCase(Mid(strData, 1, 5)) = "ERROR" Then
      bDataReceived = True
      Exit Sub
  End If
  'Se posiciona en el último separador "|"
  For i = Len(strData) To 1 Step -1
      If Mid(strData, i, 1) = "|" Then _
          Exit For
  Next i

  If IsNumeric(Mid(strData, i + 1, Len(strData) - i)) Then
      'Se determina si el total de bytes ha sido transmitido:
      If Len(Mid(strData, 1, i - 1)) = _
          Val(Mid(strData, i + 1, Len(strData) - i)) Then
          strData = Mid(strData, 1, i - 1)
          bDataReceived = True
      End If
  End If
End Sub
```

La estructura de los datos (tabla 6.1) es almacenada en un arreglo dinámico llamado `Table` dentro de la función **ReconstruyeResp**:

```
Private Sub ReconstruyeResp()
  Dim i As Integer, nRows As Integer, _
      nParam As Integer, nLastPosition As Integer
  Dim strTime As String, strCols As String

  'busca si hubo error en la ejecución y lo despliega
  If UCase(Mid(strData, 1, 5)) = "ERROR" Then
      MsgBox strData, vbInformation + vbOKOnly, _
             "Controlador de Conexiones"
      Exit Sub
  End If

  'obtiene el número de registros obtenidos del controlador
  For i = Len(strData) To 1 Step -1
      If Mid(strData, i, 1) = "|" Then _
          Exit For
  Next i

  nRows = Val(Mid(strData, i + 1, Len(strData) - i))

  'última posición de datos
  nLastPosition = i - 1

  'inicio de lectura de datos: tiempo de respuesta y _
   número de columnas

  nParam = 0
  For i = 1 To Len(strData)
```

```
        If Mid(strData, i, 1) = "|" Then
            nParam = nParam + 1
            If nParam = 2 Then _
                Exit For
            GoTo Continue
        End If

        Select Case nParam
            Case 0
                strTime = strTime + Mid(strData, i, 1)
            Case 1
                strCols = strCols + Mid(strData, i, 1)
        End Select
Continue:
    Next i

    'se asigna el tiempo de respuesta
    m_TiempoDeRespuesta = CSng(strTime)
    lblCols = strCols

    'se crea la tabla para lectura de datos + 3 renglones _
     para el encabezado de los datos: Nombre, tipo y longitud

    Dim Col As Integer, Row As Integer
    ReDim Table(nRows + 2, Val(strCols) - 1)

    Col = 0
    Row = 0
    For i = i + 1 To nLastPosition
        If Mid(strData, i, 1) = "|" Then
            Col = Col + 1
            If (Col) Mod Val(strCols) = 0 Then
```

```
                Row = Row + 1
                Col = 0
            End If
            GoTo Continue2
        End If
        Table(Row, Col) = Table(Row, Col) + Mid(strData, i, 1)
Continue2:
  Next i
End Sub
```

A este nivel, es posible leer los datos a partir de dicho arreglo, sin embargo, se recomienda armar un cursor en base a dicho arreglo, esto es opcional, es decir, fuera del modelo, pero brinda flexibilidad para el posterior manejo de los datos recibidos. Véase el método **CreaCursor**, el cual lee los datos desde el arreglo, los inserta en una tabla de Access llamada DataResult, para posteriormente releerlos mediante DAO (otro modelo de objetos provisto en Visual Basic).

```
Private Sub CreaCursor()
  Dim strHeader As String
  Dim i As Integer
  Dim Col As Integer, Row As Integer

  strHeader = "CREATE TABLE DataResult ("
  For i = 0 To Val(lblCols) - 1
      strHeader = strHeader + Table(0, i)
      Select Case Val(Table(1, i))
          Case rdTypeCHAR, rdTypeVARCHAR, rdTypeLONGVARCHAR
              strHeader = strHeader + " TEXT,"
          Case rdTypeNUMERIC, rdTypeDECIMAL, rdTypeFLOAT, _
```

```
                rdTypeREAL, rdTypeDOUBLE
                strHeader = strHeader + " DOUBLE,"
            Case rdTypeINTEGER, rdTypeBIGINT
                strHeader = strHeader + " LONG,"
            Case rdTypeSMALLINT, rdTypeTINYINT
                strHeader = strHeader + " SHORT,"
            Case rdTypeDATE, rdTypeTIME, rdTypeTIMESTAMP
                strHeader = strHeader + " DATETIME,"
            Case rdTypeBINARY, rdTypeVARBINARY, _
                 rdTypeLONGVARBINARY
                strHeader = strHeader + " BINARY,"
            Case rdTypeBIT
                strHeader = strHeader + " BIT,"
        End Select
    Next i
    strHeader = Mid(strHeader, 1, Len(strHeader) - 1) + ")"

    Dim rs As Recordset

    Set db = OpenDatabase(App.Path + "\client.mdb")

    'Se borra la última tabla DataResult para crearla nuevamente
    db.Execute "drop table dataresult", dbFailOnError
    db.Execute strHeader, dbFailOnError
    'se comienza a insertar cada registro:
    For Row = 3 To Val(lblRows) + 2
        strHeader = "INSERT INTO DataResult VALUES("
        For Col = 0 To Val(lblCols) - 1
            Select Case Val(Table(1, Col))
                Case rdTypeCHAR, rdTypeVARCHAR, rdTypeLONGVARCHAR
                    strHeader = strHeader + "'" + _
                                Trim(Table(Row, Col)) + "',"
```

```
            Case rdTypeNUMERIC, rdTypeDECIMAL, rdTypeFLOAT, _
                 rdTypeREAL, rdTypeDOUBLE
                strHeader = strHeader + _
                            Trim(Table(Row, Col)) + ","
            Case rdTypeINTEGER, rdTypeBIGINT
                strHeader = strHeader + _
                            Trim(Table(Row, Col)) + ","
            Case rdTypeSMALLINT, rdTypeTINYINT
                strHeader = strHeader + _
                            Trim(Table(Row, Col)) + ","
            Case rdTypeDATE, rdTypeTIME, rdTypeTIMESTAMP
                strHeader = strHeader + "'" + _
                            Trim(Table(Row, Col)) + "',"
            Case rdTypeBINARY, rdTypeVARBINARY, _
                 rdTypeLONGVARBINARY
                strHeader = strHeader + _
                            Trim(Table(Row, Col)) + ","
            Case rdTypeBIT
                strHeader = strHeader + _
                            Trim(Table(Row, Col)) + ","
        End Select
    Next Col

    strHeader = Mid(strHeader, 1, Len(strHeader) - 1) + ")"
    db.Execute strHeader, dbFailOnError
Next Row

'Ahora se abre DataResult mediante el cursor llamado rs
Set rs = db.OpenRecordset("select * from dataresult")
Set Data1.Recordset = rs
Exit Sub
```

```
raise_err:
  Beep
  MsgBox "Error local leyendo datos: " + Err.Description, _
         vbCritical, "Error"
End Sub
```

A este nivel ya tenemos construidos tanto el servidor como el cliente de datos, ahora veamos los requerimientos y limitaciones del cliente.

### *6.3.2 Requerimientos y limitaciones*

- Los requerimientos para la implantación del modelo propuesto dentro de aplicaciones cliente son menores y mas sencillos, basta con conocer la estructura que el controlador está manejando para garantizar una lectura correcta de los datos en el cliente.

- De la misma forma, el cliente puede ser desarrollado en un ambiente diferente al usado para el desarrollo del controlador, para que las aplicaciones cliente puedan acceder a los datos provistos a través del controlador, basta con que conozcan la información acerca de la dirección IP del controlador y el número de puerto que éste está usando para la atención a las aplicaciones cliente.

- Como se vio en el procedimiento **CreaCursor**, los métodos para acceder a los datos contenidos en el arreglo `Table` pueden ser tan variados como posibilidades de manejo de datos posea el lenguaje o herramienta que se esté usando para desarrollar la aplicación cliente.

- La forma de recibir la respuesta desde el controlador en la aplicación cliente puede ser desarrollada de diferentes formas, en base a las necesidades particulares de cada aplicación. Ante una petición de un cliente que no puede ser atendida, el controlador podría:

  a) Bloquear al cliente hasta que pueda atenderle, esto significa una forma de atención síncrona.

  b) El controlador se apunta la petición y le dice al cliente que le contesta cuando pueda, desbloqueando al cliente (forma de atención asíncrona)

  c) Contestarle que no puede. El cliente se provee de algún método para validar si la conexión fue exitosa o no, y tener algún algoritmo para controlar el número de intentos de conexión con un tiempo de espera, como intervalo entre cada intento.

## 6.4 Liberación y consideraciones finales

*Liberación*

Una vez corriendo el controlador de conexiones, se procede a asignarle el Data Source Name, el usuario y password así como el número de conexiones y el puerto lógico; con estos datos el controlador procede a abrir las *n* conexiones solicitadas por medio del Data Source Name a la fuente de datos. En este momento, el controlador está listo para atender cualquier petición hecha por las aplicaciones cliente.

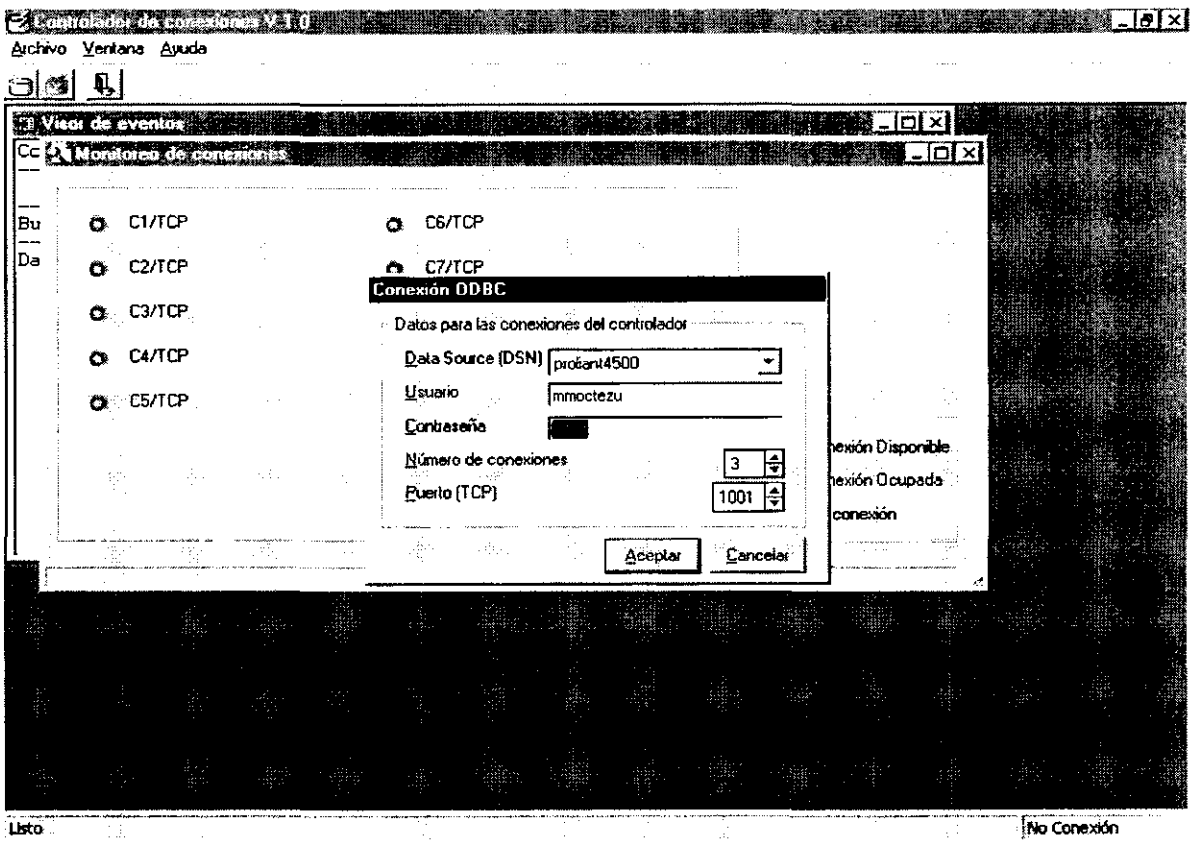



**Figura 6.7 Parámetros iniciales del controlador**

En la figura 6.7 el controlador esta abriendo tres conexiones mediante el Data Source Name "proliant4500" el cual está direccionado a una base de datos SQL Server, el controlador se está conectando a través del usuario "mmoctezu" y usará el puerto 1001.

A continuación, el la figura 6.8 se observa al controlador abriendo las tres conexiones solicitadas así como el respectivo socket por cada conexión.

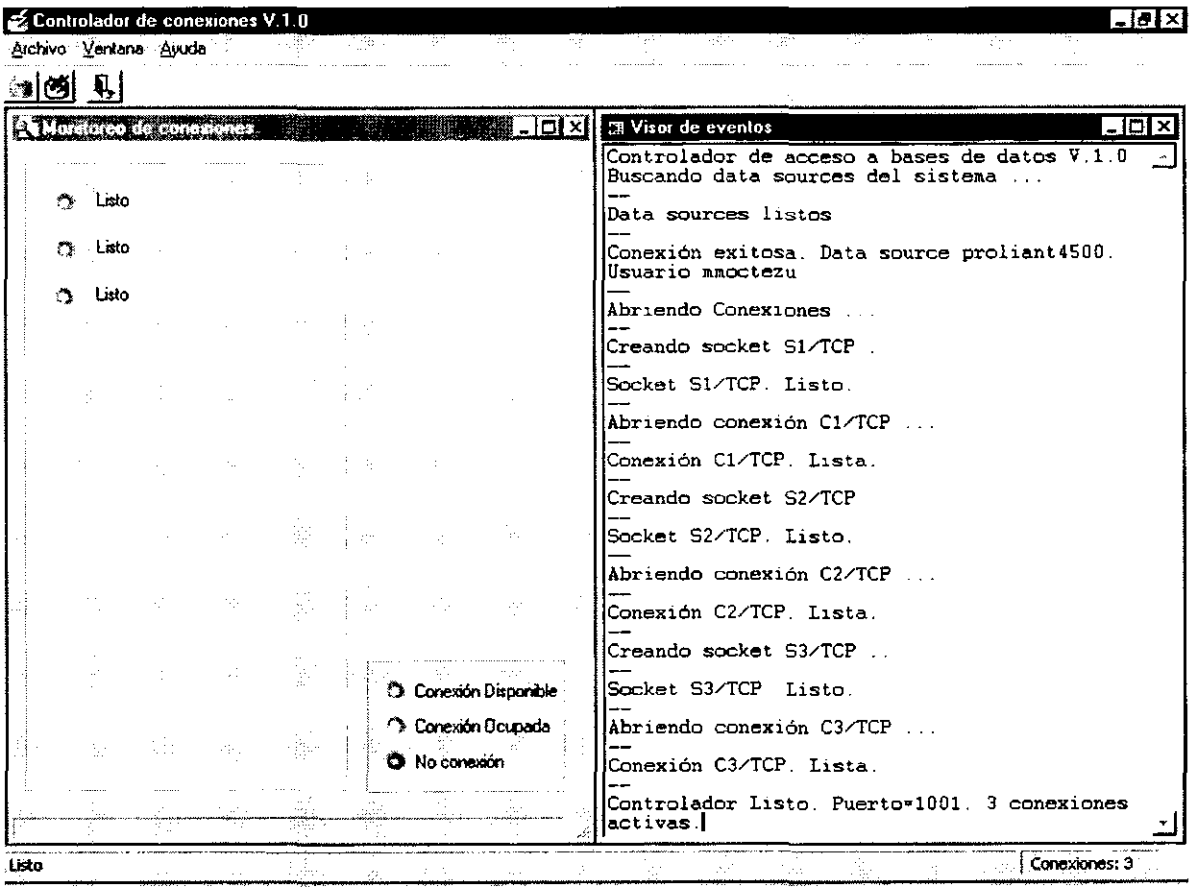



**Figura 6.8 El controlador abriendo conexiones**

Del lado de la aplicación cliente, se procede a asignarle la dirección IP del controlador así como el número de puerto, el usuario y el password –dichos valores deben ser asignados de manera interna, es decir programáticamente, debido a que, para los usuarios finales no es necesario que estos conozcan dichos parámetros-; una vez hecho esto, se hace un llamado a la función `Ejecuta` y la aplicación cliente espera la respuesta del controlador, cuando dicha respuesta le es enviada, nuestra aplicación cliente procede a reconstruir los datos mediante la función `ReconstruyeResp`.

Nuestra aplicación (figura 6.9) es un sencillo programa que solicita cualquier sentencia SQL al controlador, los datos obtenidos los muestra en una tabla que la misma aplicación posee, a partir de la función **ReconstruyeResp**.

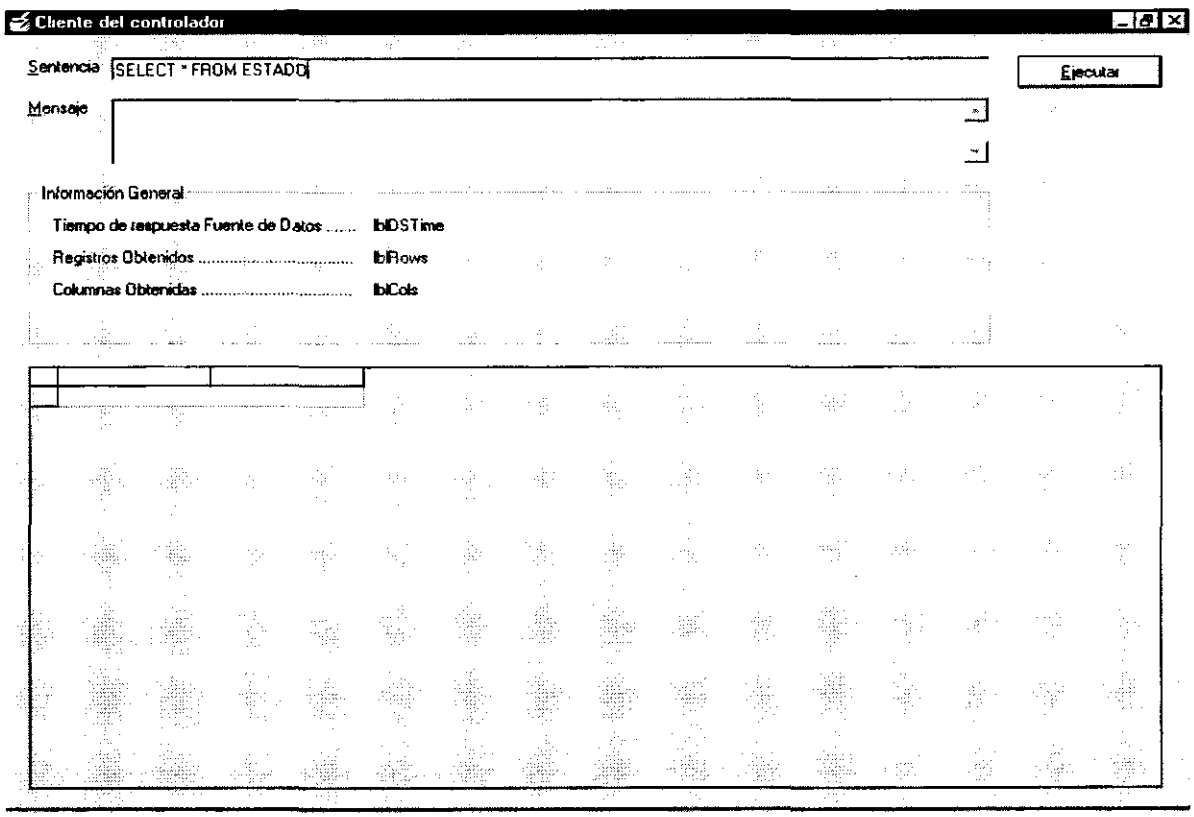



**Figura 6.9 Ejemplo de aplicación cliente.**

Cuando el controlador detecta una llamada procede a registrarla y ejecutarla, una vez realizada la petición envía los datos a la aplicación cliente y procede a su desconexión (figura 6.10).

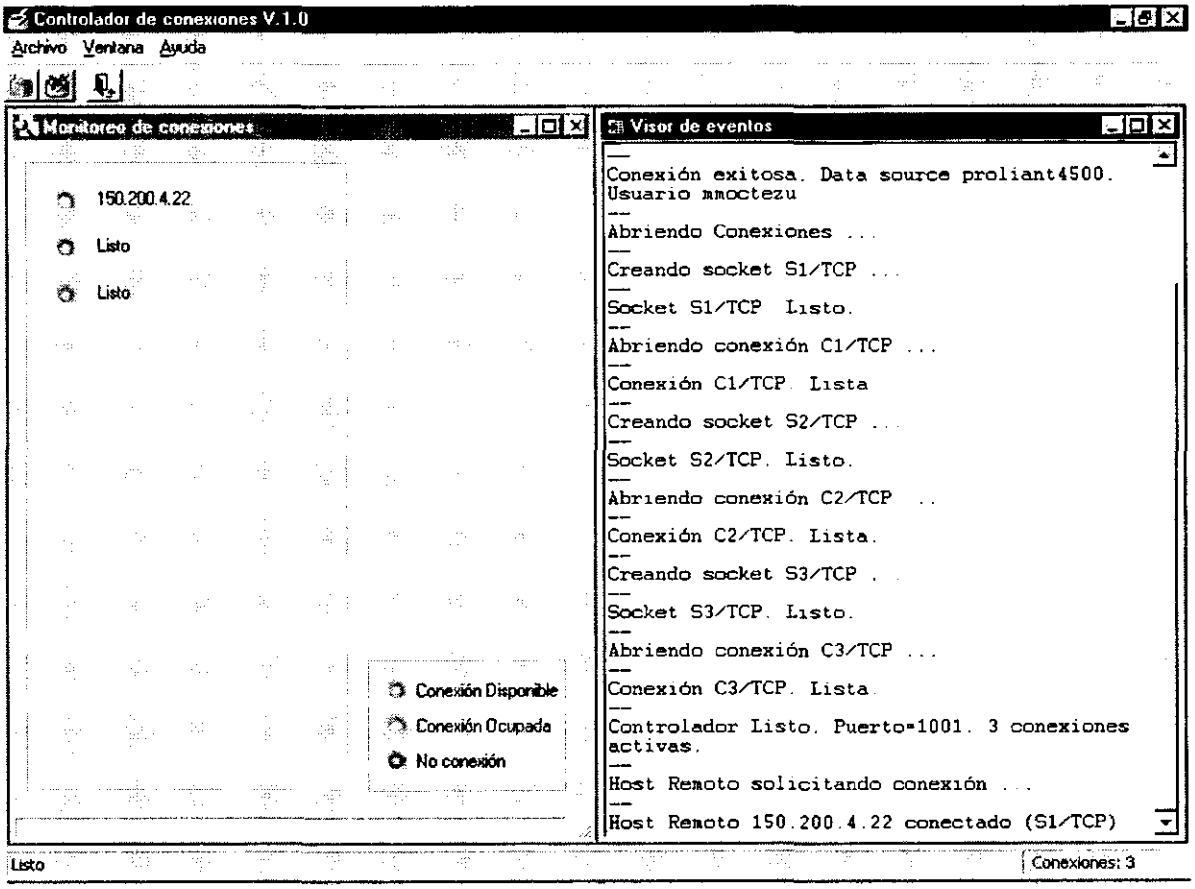



**Figura 6.10 El controlador detectando una petición.**

Inmediatamente, el controlador procede a leer la petición solicitada, ejecutarla y enviar los resultados a la aplicación cliente (figura 6.11).

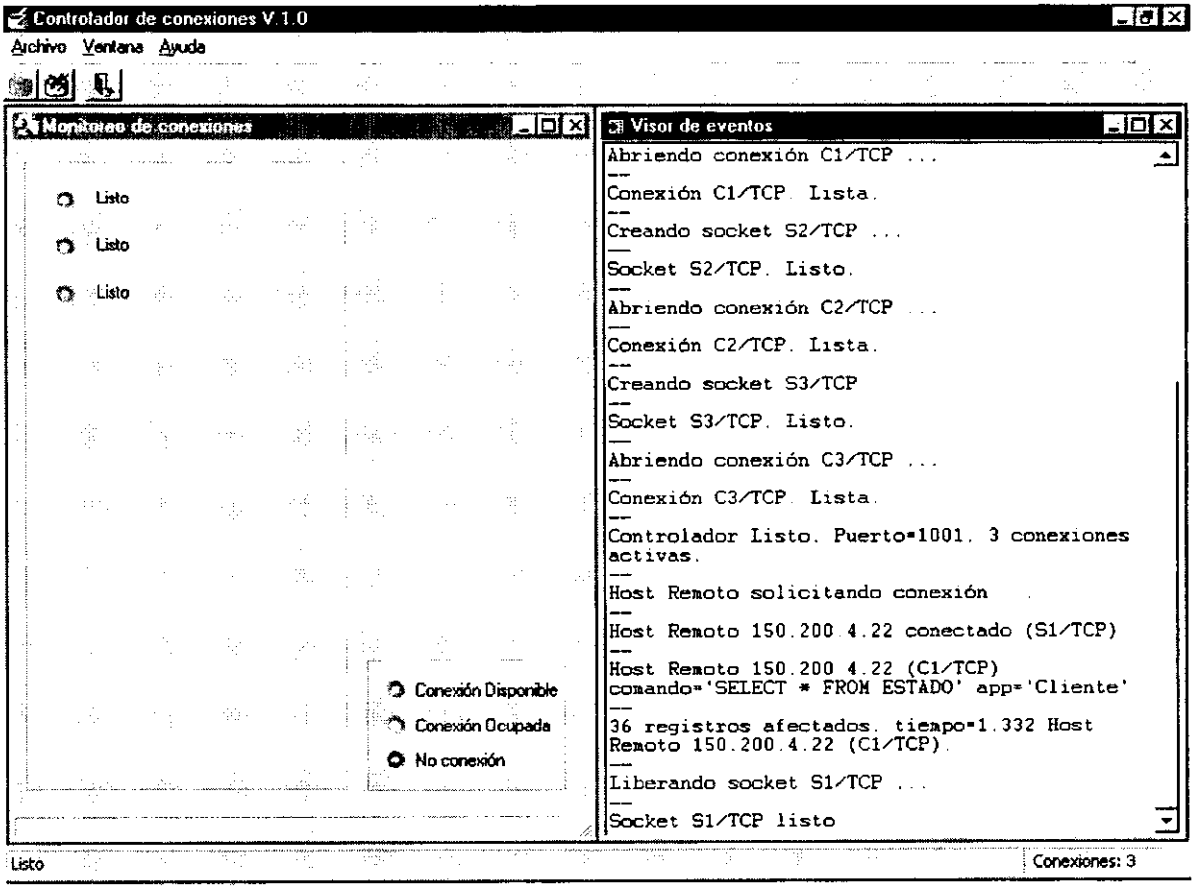
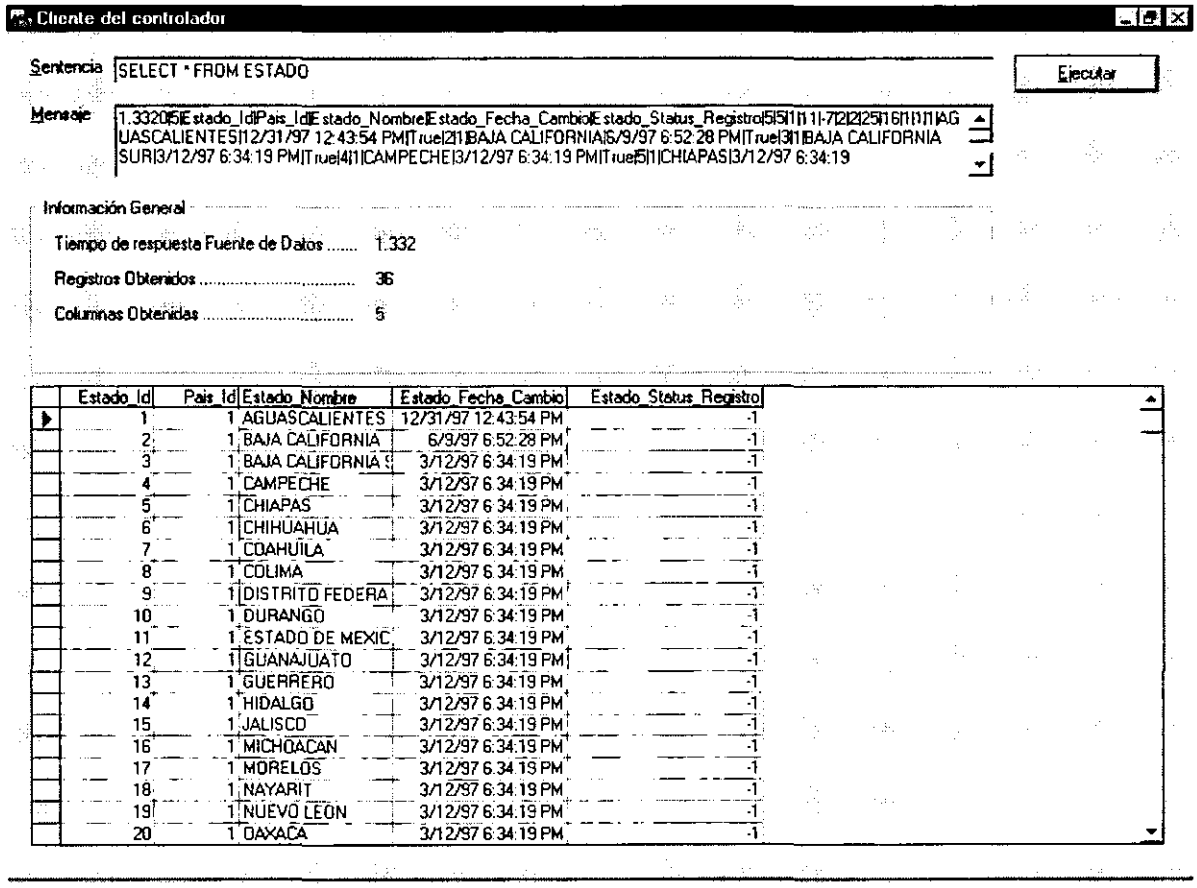



**Figura 6.11 El controlador ejecutando peticiones.**

Como se observa en la figura anterior, una vez procesada la petición de la aplicación cliente la conexión del controlador queda disponible para cualquier otra petición y los datos son enviados a la aplicación cliente (figura 6.12).




| Estado_Id | Pais_Id | Estado_Nombre | Estado_Fecha_Cambio | Estado_Status_Registro |
|---|---|---|---|---|
| 1 | 1 | AGUASCALIENTES | 12/31/97 12:43:54 PM | -1 |
| 2 | 1 | BAJA CALIFORNIA | 6/9/97 6:52:28 PM | -1 |
| 3 | 1 | BAJA CALIFORNIA S | 3/12/97 6:34:19 PM | -1 |
| 4 | 1 | CAMPECHE | 3/12/97 6:34:19 PM | -1 |
| 5 | 1 | CHIAPAS | 3/12/97 6:34:19 PM | -1 |
| 6 | 1 | CHIHUAHUA | 3/12/97 6:34:19 PM | -1 |
| 7 | 1 | COAHUILA | 3/12/97 6:34:19 PM | -1 |
| 8 | 1 | COLIMA | 3/12/97 6:34:19 PM | -1 |
| 9 | 1 | DISTRITO FEDERA | 3/12/97 6:34:19 PM | -1 |
| 10 | 1 | DURANGO | 3/12/97 6:34:19 PM | -1 |
| 11 | 1 | ESTADO DE MEXIC | 3/12/97 6:34:19 PM | -1 |
| 12 | 1 | GUANAJUATO | 3/12/97 6:34:19 PM | -1 |
| 13 | 1 | GUERRERO | 3/12/97 6:34:19 PM | -1 |
| 14 | 1 | HIDALGO | 3/12/97 6:34:19 PM | -1 |
| 15 | 1 | JALISCO | 3/12/97 6:34:19 PM | -1 |
| 16 | 1 | MICHOACAN | 3/12/97 6:34:19 PM | -1 |
| 17 | 1 | MORELOS | 3/12/97 6:34:19 PM | -1 |
| 18 | 1 | NAYARIT | 3/12/97 6:34:19 PM | -1 |
| 19 | 1 | NUEVO LEON | 3/12/97 6:34:19 PM | -1 |
| 20 | 1 | OAXACA | 3/12/97 6:34:19 PM | -1 |

**Figura 6.12 Los resultados de la búsqueda.**

Con la figura anterior termina la muestra del esquema Cliente/Servidor a través del un controlador de conexiones.

*Consideraciones finales.*

- El presente proyecto fue desarrollado en Visual Basic, pero se pueden crear arquitecturas Cliente/Servidor donde el servidor halla sido creado en un lenguaje de programación diferente al cliente, esto permite la flexibilidad en la implantación de modelos basados en controladores de conexiones. Solo es necesario que el cliente "conozca" la dirección IP del controlador que lo atenderá así como el número de puerto para el establecimiento de la conversación.

- Este proyecto no trata de manera exhaustiva la seguridad de la comunicación Cliente/Servidor, por tal motivo el lector, en caso de implantar un modelo similar al presentado, debe diseñar algoritmos realmente eficaces de encriptación de los datos procedentes de la conversación Cliente/Servidor.

- La estructura propuesta en la tabla 6.1 puede ser modificada en base a los requerimientos particulares de cada sistema, siempre y cuando exista concordancia entre los datos que el controlador envía y la forma en que las aplicaciones cliente los interpretan.

- Para el cliente de datos, se recomienda crear una clase o un control con sus respectivos métodos y propiedades con el fin de simplificar el desarrollo de aplicaciones mediante el modelo presente. El desarrollador solo tendrá que instanciar la clase o el control en su proyecto, simplificando así el desarrollo de su aplicación y ahorrando tiempo. Incluso se pueden crear clases y controles como componentes ActiveX lo cual permite la extensibilidad del modelo propuesto a aplicaciones para internet.

La implantación de un modelo similar al aquí propuesto para el acceso a datos, podrá mejorar tiempos de acceso a datos desde un front end, así como usar de una manera optima los recursos disponibles en el servidor que contiene la base de datos.

Este prototipo servirá de base para tener una nueva visión en cuanto al manejo y envío de datos desde un servidor. A partir de aquí se podrían sentar bases para un modelo formal de acceso a datos mediante sockets, abarcando aspectos como la seguridad en el tráfico de datos y control de conexiones concurrentes a una base de datos.

Desde el punto de vista del autor, este proyecto posee efectividad real en cuanto a:

- Optimización de tiempo de acceso a bases de datos, esto originando reducción de costos en cuanto al tiempo horas/hombre, tiempo de procesamiento tanto del cliente como del servidor, tiempo de uso de la red (que en caso de ser satelital el costo en dinero es sumamente elevado), y, en general tiempo de transacción.

- La administración de conexiones, esto es mediante la asignación de un número máximo de conexiones así como su distribución entre los clientes potenciales de datos.

- La generalidad de su uso, mediante la construcción de un modelo que ha sido construido sobre la plataforma ODBC, lo cual garantiza su implantación en diferentes bases de datos tales como Informix, SQL Server, Access, Oracle, SQLBase, Sybase, Paradox; así como manejadores ISAM tales como DBase, Excel, Foxpro e incluso archivos de texto y bibliotecas AS400.

El presente proyecto puede ser el punto de partida para una apertura en el uso de sockets en la implantación de sistemas de acceso a bases de datos.

Este proyecto posee los elementos básicos para lograr un acceso a datos auxiliándose de las bondades de sockets, por tal motivo, se puede mejorar y robustecer casi todas las partes de dicho proyecto, algunas, de las que se pueden enumerar son:

- El uso de ODBC API para hacer conectividad a bases de datos en vez del modelo de objetos RDO.

- La implantación de un control de usuarios que acceden a la base de datos, lo cual permita una validación y autentificación del cliente.

- La encripción de datos en la interface servidor-cliente de datos, como en internet se ha venido probando, a fin de garantizar seguridad al viajar los datos.

Y en general, el presente proyecto puede dejar abiertos aspectos para futuras investigaciones tales como:

- Desarrollo de sistemas Cliente/Servidor para bases de datos.
- Uso de sockets en redes, internet e intranet.
- Administración de servidores de datos y tráfico de transacciones.
- Monitoreo de requisición de transacciones en bases de datos.

La tendencia actual es unificar el acceso a bases de datos mediante el estándar ODBC, por tal motivo las empresas desarrolladoras de manejadores de bases de datos cada vez están mas convencidas de la necesidad de ésta unificación.

Ello ofrece cada vez mas ventajas y facilidades tanto al desarrollador de software como al usuario final y a la empresa: la transparencia entre aplicaciones, bases de datos, e incluso sistemas operativos.

Pensando en esto, el presente proyecto ofrece la posibilidad de tener un servidor de datos "configurable", en cuanto a la base de datos a administrar (la cual puede ser desde un simple DBF hasta una biblioteca AS400), y el número de conexiones máximas concurrentes, mediante el uso de uno de los objetos más robustos diseñados para el acceso a bases de datos: RDO (Remote Data Object).

Incluso es posible desarrollar un cliente de datos orientado a un lenguaje y plataforma particular, el cual siga siendo abastecido por un servidor de datos, ofreciendo adaptibilidad y portabilidad.

Así pues, las cartas están expuestas, el autor considera el presente proyecto, como un proyecto atractivo y abierto. Sus aplicaciones pueden ser muchas y su extensibilidad, viable.

# CONCLUSIONES

Podemos concluir este trabajo, considerando su contenido en primer lugar, y en segundo, comentando las implicaciones que se desprenden del mismo.

El modelo Cliente/Servidor es la solución de tecnología informática que promete satisfacer la necesidad de mejorar la relación costo/beneficio en las organizaciones. En la actualidad se observa una clara tendencia de la industria de la computación hacia este modelo. Muchas empresas están orientando sus pasos hacia el modelo Cliente/Servidor de una u otra forma y, aun cuando las tecnologías relacionadas no estén completamente maduras, el esperar a que esto ocurra para iniciar un cambio tecnológico significaría un retraso a mediano y largo plazo del cual sería difícil y costoso recuperarse.

Así pues, concluimos que la arquitectura Cliente/Servidor está siendo usada cada vez más para la implantación de sistemas de cómputo, dicha arquitectura ofrece un sin número de ventajas en comparación con otros modelos como el modelo multiusuario: se obtiene un mejor aprovechamiento de los recursos, puede operar bajo sistemas abiertos, aumenta la productividad y disminución del costo y tiempos de desarrollo, y sobre todo permite tener una gran flexibilidad de uso y facilidad cuando se requiere realizar migraciones de plataformas de hardware y software.

De la misma manera, se concluye que es necesario que existan estándares para lenguajes, protocolos de comunicación y protocolos de acceso a datos –entre otros-. Dichos estándares facilitan el desarrollo de sistemas y su implantación, así como la interoperabilidad de diferentes plataformas de hardware y software.

En cuanto al acceso a bases de datos tradicional, se puede concluir que generalmente el acceso tiende a ser lento, debido a que se utiliza gran parte del tiempo en procesos de conexión/desconexión, esto no permite un buen aprovechamiento de la potencia de los

equipos servidores de cómputo, ya que con un alta concurrencia de usuarios se genera una degradación en el desempeño de los equipos servidores y, por ende, una degradación total en el modelo de conversación Cliente/Servidor.

De igual forma, el mantener una conexión permanente Cliente/Servidor desde el inicio hasta el fin de una operación (de una aplicación cliente) incrementa el tráfico en la red, debido a que el servidor se ve obligado a mantener "colgada" la conexión del cliente hasta que la misma aplicación cliente solicite su propia desconexión.

La tendencia actual es unificar el acceso a bases de datos, por tal motivo la industria de software está cada vez más convencida de la necesidad de dicha unificación. La generalidad del modelo propuesto mediante ODBC permite su implantación para cada vez más fuentes de datos, a medida que la industria de la computación se va adheriendo a los estándares actuales (SQL, ODBC, etc.).

Se concluye que, mediante la implantación de un controlador como el aquí propuesto se logra una mejora en los tiempos de respuesta a las peticiones cliente, así como una mejor administración y distribución de conexiones a partir de un esquema de conversación Cliente/Servidor basado en petición/respuesta-inmediata en vez del esquema tradicional conexión/aplicación.

# GLOSARIO

**API.** (Aplication Program Interfase). Interfaz de Programa de Aplicación. Conjunto de funciones y procedimientos que están contenidas dentro de librerías (DLLs), estas librerías pueden ser usadas en cualquier aplicación para hacer uso de las funciones y procedimientos contenidas en ellas.

**Asíncrono.** Sin una relación regular al tiempo; inesperado o impredecible respecto a la ejecución de instrucciones de programas.

**Base de Datos.** Es una colección de datos interrelacionados almacenados juntos, con redundancia controlada en función de un esquema para dar servicio a una o más aplicaciones. Es un conjunto de datos almacenados juntos y manipulados por un sistema de administración de bases de datos.

**DDE.** (Dynamic Data Exchange). Intercambio Dinámico de Datos. Protocolo de comunicación entre procesos utilizado por las aplicaciones para definir enlaces dinámicos. La información actualizada en una aplicación se refleja automáticamente en otra aplicación enlazada a la primera a través de DDE.

**Downsizing.** Migración de aplicaciones de negocios hacia plataformas de cómputo mas pequeñas; por ejemplo, migrar desde un sistema principal o "mainframe" a un entorno de sistemas bajo una red LAN, para aprovecharse de ventajas como reducción de costos, flexibilidad, eficiencia y rentabilidad.

**Encapsulación.** En programación orientada a objetos, es la asociación de datos y funciones que tiene el efecto de ocultar al solicitante de una función la forma en que ésta se ha

desarrollado. Por ejemplo, una librería API puede ser usada en aplicaciones finales sin ser necesario el conocer cómo dicha API fue programada.

**Enhebramiento.** Es un ambiente de computadora donde cada proceso tiene un espacio de direcciones y un único hilo (thread) o hebra de control.

**Escalabilidad.** La posibilidad de instalar una estructura de sistemas de información dentro de la configuración de hardware capaz de crecer gradualmente a medida que cambien las necesidades.

**Estación de trabajo.** Es la configuración de equipos de E/S con la que trabaja un operador de computadoras. Terminal o microprocesador, generalmente conectado a un sistema principal o a una red, en el que el usuario puede ejecutar aplicaciones.

**GUI.** (Graphical User Interfase). Interfaz Gráfica de Usuario. Un tipo de interfaz de usuario que sustituye las pantallas basadas en caracteres por pantallas de gráficos de alta resolución, con todos los puntos direccionables, que utiliza ventanas para mostrar simultáneamente múltiples aplicaciones y permite además que el usuario introduzca datos a través del teclado o de un dispositivo apuntador, por ejemplo un mouse, un lápiz óptico o un track ball.

**Handle.** Un Handle es un puntero que lleva la dirección en memoria de otro puntero contenido en una tabla gestionada por el Adminstrador de memoria, que a su vez contiene la dirección en memoria de un dato con un tipo específico, una estructura de datos previamente declarada o bien un bloque de memoria de un tamaño bien definido.

**Host.** En una red de computadoras, es un equipo que proporciona servicios, por ejemplo, de cálculo, acceso a bases de datos y funciones de control de red. Computadora principal o de control en una instalación de múltiples computadoras.

**Ícono.** Una representación pictórica de un elemento seleccionable por el usuario. Los íconos representan cosas -por ejemplo, un documento o un archivo- con las que el usuario puede trabajar o acciones que el usuario puede ejecutar.

**Interoperabilidad.** Es la habilidad de interconectar sistemas de diferentes fabricantes y hacerlos trabajar conjúntamente para satisfacer las necesidades del negocio. Algunos ejemplos o requerimientos son: intercambio de mensajes entre los sistemas, compartimiento de recursos y datos con otras aplicaciones que se ejecuten en diferentes plataformas de hardware y software.

**LAN.** (Local Area Network). Red de Area Local. Red que tiene un dispositivo centralizado llamado servidor, que proporciona servicios a múltiples dispositivos asociados, llamados clientes.

**LAN Manager.** Es la red basada en OS/2 que opera sistemas desarrollados conjuntamente por Microsoft, y 3Com. Varios fabricantes, incluyendo AT&T y NCR tienen versiones del LAN Manager para sus propias plataformas de hardware.

**Mainframe.** Sistema principal o módulo físico en un sistema de múltiples componentes que contiene la Unidad Central de Proceso, y que está conectado a otros componentes del sistema por cables que utilizan conectores estándar. Su uso ha originado el término "Sistemas grandes de cómputo".

**Multitarea.** Extensión del concepto de multiprogramación en el cual el tiempo del procesador es distribuido a través de múltiples aplicaciones dando a cada una acceso al procesador por periodos cortos de tiempo. Una multitarea puede ser cooperativa o primitiva en naturaleza.

**OLE.** (Object Linking and Embbeding). Objeto Ligado e Incrustado. Protocolo de Microsoft para enlazar múltiples formatos de datos, como texto, voz e imagen entre otros a fin de crear un documento compuesto, que puede visualizarse y manipularse como un todo coherente. OLE se centra en los formatos de los documentos, en lugar de hacerlo en la capacidad de las aplicaciones para intercambiar datos, como es el caso de DDE.

**OLTP.** (On-Line Transaction Proccesing). Procesamiento de Transacciones en Linea. Un tipo de proceso que soporta aplicaciones interactivas en las que las peticiones enviadas por los usuarios de terminales se procesan tan pronto como se reciban. Los resultados se devuelven al peticionario en un periodo de tiempo relativamento corto. Un sistema de proceso de transacciones supervisa el proceso de compartir recursos para procesar múltiples transacciones al mismo tiempo, minimiza el tiempo de proceso y duración del bloqueo.

**Orientado a objetos.** Diseño y desarrollo de software basado en los datos que se manipulan, conocidos como objetos; en lugar de basarse en funciones que manipulan datos, según la metodología clásica de desarrollo de software. El diseño y desarrollo orientado a objetos facilita la expansión, reutilización y compatibilidad de software.

**Portabilidad.** Es la posibilidad de mover componentes de software de aplicación de un sistema a otro. La portabilidad perfecta permite hacerlo sin tener que modificar los componentes.

**Proceso cooperativo.** Proceso por el cual una sola aplicación se reparte entre dos o más plataformas de hardware. Muy a menudo se utiliza el término para reflejar una relación muy rígida entre las partes de una aplicación.

**Proceso distribuido.** Procesamiento distribuido es un modelo de aplicación y/o sistema en el cual las funciones y los datos pueden estar distribuidos en múltiples recursos del sistema conectados a una LAN o una WAN.

**Protocolo.** Conjunto formal de convenciones que gobiernan el formato y control de datos. Un conjunto de procedimientos o reglas para establecer y controlar transmisiones desde un dispositivo o proceso fuente a un dispositivo o proceso objeto. Serie de reglas de semántica y sintaxis que determinan la comunicación entre máquinas y el orden en que se envían los datos (bytes o bits).

**Síncrono.** Relativo a dos o más procesos que dependen de que ocurra cierto suceso, por ejemplo, una señal de tiempo común, proceso que ocurre con un intervalo de tiempo predecible o regular.

**Sistemas abiertos.** Conjunto completo y consistente de estándares internacionales de tecnologías de información y funcionales, que especifica interfaces, servicios y formatos de soporte para conseguir la interoperabilidad y portabilidad de aplicaciones y datos.

**SQL.** (Structured Query Language) Lenguaje Estructurado de Búsquedas. SQL comenzó como lenguaje de consulta de IBM para DB2, la base de datos relacional para sistemas IBM. SQL se hizo tan popular con los usuarios y proveedores de IBM que ANSI adoptó una versión de SQL como estándar U.S. en 1986. Un año mas tarde, ISC concedió a SQL el estado formal de estándar internacional.

**TCP.** (Transmision Control Protocol). Protocolo de Control de Transmisiones. Es la capa de TCP/IP que proporciona un flujo fiable y continuo de datos entre procesos situados en nodos distintos de redes de sistemas interconectados. TCP asume que IP (Protocol Internet) es el protocolo subyacente.

**TCP/IP.** (Transmision Control Protocol / Internet Protocol). Es un conjunto de protocolos estándar desarrollados por el Departamento de Defensa de USA para la red ARPANET gubernamental. TCP/IP proporciona comunicaciones host-to-host confiables entre los

nodos de una red. Ha sido aceptado como el protocolo estándar para los sistemas de redes Ethernet de UNIX. TCP/IP es más sencillo que el modelo OSI y contiene cuatro capas:

- Aplicación.
- Protocolo de Control de Transmisión.
- Protocolo Internet.
- Interfaz de red.

**Thread.** Véase Enhebramiento.

**Transacción.** Unidad de proceso (compuesta de uno o más procesos de aplicación) que inicia mediante una única llamada. Una transacción puede requerir la iniciación de una o más tareas para su ejecución.

**Transparencia.** El estado por el cual ciertos elementos del sistema quedan ocultos, de forma que otros componentes del sistema no necesitan depender de la forma específica en que aquellos fueron desarrollados. Por ejemplo, la situación de un servidor es transparente a sus clientes, así, cuando la ubicación cambia, no es necesario cambiar a los clientes.

# BIBLIOGRAFIA

BERSON, Alex. *Client/Server Architecture.* (United States of America, McGraw-Hill, 1992).

BOHNHOFF, Peter, y JANSSEN, Rob. *Building Client/Server Solutions.* (United States of America, International Bussiness Machines, 1994).

BONNER, Patrice. *Network programming with Windows Sockets.* United States of America, Prentice Hall PTR, 1996, 493 pp.

DAY, Michel. *Downsizing to Netware.* (NRP, United States of America, 1992).

ECO, Humberto. *Como se hace una Tesis.* España, Gedisa, 1994, 267 pp.

*Guide to building Client/Server solutions.* (Digital Equipment Corporation, United States of America, 1993).

*Internetworking with TCP/IP on Windows NT 4.* (United States of America, Microsoft Press, 1997).

*Microsoft ODBC 3.0 Software Development Kit and Programmers Reference.* (United States of America, Microsoft Press, 1997).

*Microsoft TCP/IP Training.* (United States of America, Microsoft Press, 1997).

*Microsoft Windows NT Server Guía de Redes.* (United States of America, Microsoft Press, 1996).

RAHMEL Dan, y RAHMEL Ron. *Client/Server applications with Visual Basic 4.* United States of America, Prentice Hall PTR, 1996, 1038 pp.

SMITH, Patrick. *Client/Server Computing.* (United States of America, SAMS, 1992).

VAUGHN, Larry. *Client/Server Design and Implementation.* (United States of America, McGraw-Hill, 1994).

VAUGHN, William R. *Guide to Visual Basic & SQL Server.* Fifth Edition. (United States of America, Microsoft Press, 1997).