UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES ARAGÓN

ESTRATEGIA DE MICROSOFT PARA EL DESARROLLO DE APLICACIONES DE TRES CAPAS UTILIZANDO VISUAL BASIC 6.0

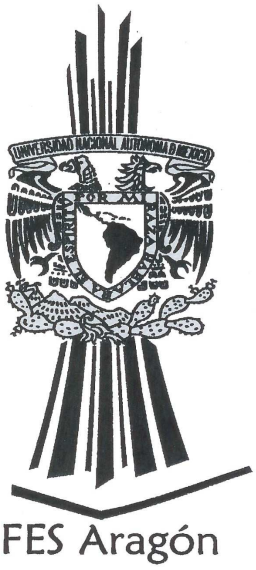
T E S I S

QUE PARA OBTENER EL TÍTULO DE:
INGENIERO EN COMPUTACIÓN
P R E S E N T A :

HANOI HERNÁNDEZ VILLASEÑOR

ASESOR:

ING. SILVIA VEGA MUYTOY



SAN JUÁN DE ARAGÓN, ESTADO DE MÉXICO, 2009

Agradezco Sinceramente

A mi madre Celia:

Por enseñarme lo que significa la palabra Luchar.

A mi padre René Sergio:

Quien no dejó que muriera el niño dentro de mí.

A mis hermanos:

Por que juntos hemos tenido una vida maravillosa y envidiable.

A mi esposa Ana:

Por permitirme seguir viviendo un sueño.

A la Ing. Silvia Vega:

Por su paciencia y dedicación para con un servidor.

A la Universidad Nacional Autónoma de México:

Por confiar en mí para seguir poniendo en alto el nombre de esta gloriosa institución.

INDICE

INTRODUCCIÓN

Con el avance tecnológico en el medio computacional, la necesidad que tienen los usuarios de manipular grandes volúmenes de información con aplicaciones confiables y escalables en tiempos más reducidos han puesto en duda la confiabilidad y el rendimiento de los Sistemas Informáticos Cliente-Servidor Tradicionales de Arquitectura de Dos Capas, dando como consecuencia un incremento alarmante en el Costo Total de Propietario (*TCO*).

En la actualidad, Microsoft a propuesto una nueva Arquitectura de n Capas basada en Servicios, estos son: Servicios de Usuario, Servicios de Negocios y Servicios de Datos. Donde los Servicios de Usuario es aquella funcionalidad que está en las computadoras del Cliente y los Servicios de Negocios y de Datos se encuentran en un Servidor respectivamente.

Con esta nueva estrategia, Microsoft asegura un servicio rentable para la empresa, un desarrollo y mantenimiento fácil para los programadores ya que cada capa es independiente y la confiabilidad y rapidez para los clientes.

Este trabajo se enfoca a los servicios de Negocios y Datos, y se encuentra dividido en tres partes:

En la primer parte se comparan los Sistemas Informáticos de Arquitectura de Dos Capas contra los Sistemas Informáticos de Arquitectura de n Capas.

En la segunda parte se explica el concepto de Arquitectura de n Capas así como los servicios que comprende y una explicación de las herramientas con las que se pueden crear cada uno de estos servicios.

Para finalizar, en la tercer parte se presentan algunas herramientas adicionales con las cuales se puede incrementar la confiabilidad, el rendimiento y los alcances del sistema.

A lo largo de este trabajo se podrán encontrar algunos ejemplos de codificación en Visual Basic Versión 6.0 para la creación de los servicios, así como también los pasos necesarios para implementar seguridad, manipular transacciones y administrar la herramienta de Microsoft Transaction Server (*MTS*).

1. ESTRATEGIA DE MICROSOFT PARA EL DESARROLLO DE APLICACIONES DE TRES CAPAS CLIENTE/SERVIDOR

Antes de empezar a crear aplicaciones de tres capas, es necesario hacer memoria de las distintas configuraciones por las que han pasado los sistemas que cubren los requerimientos de un cliente.

Después de recordar estas arquitecturas podremos empezar a estudiar las aplicaciones de tres capas y podremos comprobar sus bondades contra de las arquitecturas anteriores.

1.1. EVOLUCIÓN DE LAS APLICACIONES

Al hablar de una aplicación, se da ha entender un sistema que tiene interacción con el usuario, en la cual él proporciona datos y recibe información.

A este tipo de aplicaciones se les puede llamar Cliente/Servidor, en la cual el cliente es el usuario y la aplicación es el servidor.

Antiguamente estas aplicaciones daban servicio en una sola máquina, con esto su distribución era sencilla y su mantenimiento también. Sin embargo si existían diferentes usuarios manejando la misma aplicación el unir la información almacenada implicaba costos en tiempo, además de posibles errores al realizar esta tarea (duplicidad o pérdida de información). Al llegar las redes computacionales el Cliente/Servidor, evolucionó convirtiéndose en una aplicación distribuida de dos capas, donde diferentes clientes (usuarios) podían manejar la misma aplicación para manipular la misma información almacenada en una sola máquina reduciendo costos en tiempo y posibles errores, ya que la información es centralizada.

Definiendo Servicios en un Cliente/Servidor.

- *Servicios de Usuarios*. Unidad lógica de la aplicación que se encarga de presentar las pantallas a través de las cuales interactúa el cliente.
- *Servicios de Negocio*. Unidad lógica de la aplicación que controla las reglas del negocio, validaciones de información y la integridad transaccional de las operaciones que realiza el usuario.
- *Servicios de Datos*. Se encarga de solicitar y manipular la información que se encuentra en la base de datos.

Actualmente, algunos sistemas Cliente/Servidor, separan la capa de usuario de la capa de negocios y datos, dando como resultado una arquitectura de dos capas, otros separan los tres servicios, teniendo como base una arquitectura de tres capas. Cabe aclarar que éstas no corresponden necesariamente a diferentes locaciones físicas en la red; es decir, las tres pueden existir en sólo dos máquinas o en más.

La arquitectura de tres capas a tenido tanto éxito que hoy en día las grandes empresas han adoptado esta tecnología para desarrollar aplicaciones que den servicio a sus empleados o incluso para dar servicio a sus clientes a través de Internet.

1.1.1. APLICACIONES DE DOS CAPAS

Estas aplicaciones agrupan los tres servicios en diferentes esquemas:

Esquema 1: En la máquina del cliente se agrupan los tres servicios, dejando en el servidor sólo la base de datos que almacena la información.

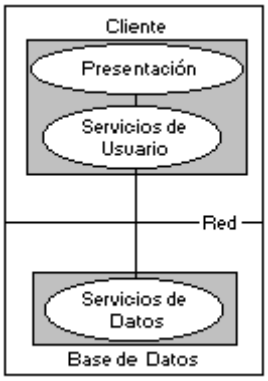


Fig. 1.1

Ventajas:

- Datos compartidos.
- Información consistente y centralizada.
- Reducción de mantenimiento y duplicación.
- Seguridad.

Desventajas:

- Aplicaciones monolíticas (todo está integrado y es imposible su integración con otras aplicaciones).
- Difíciles de escalar (tiene un número limitado de conexiones).
- Difícil mantenimiento (cualquier cambio en la aplicación, implica la redistribución a los clientes, involucrando perdida de tiempo y aumento en costos).

- Baja confidencialidad (todas las reglas del negocio se encuentran del lado de los clientes).

Esquema 2: Los servicios de usuario se encuentran en el cliente, y los servicios de negocios y datos se encuentran en el servidor como procedimientos almacenados (*Stored Procedures*) en la base de datos.

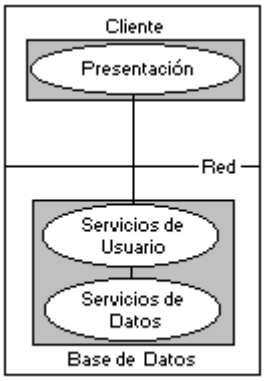


Fig. 1.2

Beneficios al utilizar *Stored Procedures*:

- El proceso ocurre en el servidor.
- Mejora la seguridad.
- Permite reutilización de los Stored Procedures para otras aplicaciones.
- Mejora tiempo de respuesta.

Limitaciones:

- El lenguaje SQL tiene grandes deficiencias en comparación con otros lenguajes como Visual C++, Visual Basic, Visual FoxPro, etc.
- Si el Stored Procedures es muy largo, las conexiones a la base de datos serán atadas durante mucho tiempo.

- Los Stored Procedures están atados al lenguaje de la base de datos donde fueron desarrollados, por lo cual si deseamos migrar la base de datos éstos tendrán que ser modificados.

Cuando desarrollamos aplicaciones bajo esta arquitectura se vuelven independientes, por ejemplo:

Tomemos el caso de un banco.

Inicialmente se tuvieron que desarrollar aplicaciones en x lenguaje para manipular la información (datos particulares, transacciones monetarias, etc.).

Al crearse los cajeros automáticos se necesitó de una aplicación diferente para manejar transacciones monetarias (servicios que ya se manejaban en aplicaciones anteriormente).

Lo que se trata de hacer notar es que para diferentes clientes que operen los mismos servicios es necesaria una nueva aplicación Cliente/Servidor para administrar la información global.

1.1.2. APLICACIONES DE TRES CAPAS

También llamadas aplicaciones distribuidas o aplicaciones de n capas, son aquellas que separan los servicios de datos, de negocios y de usuarios, volviéndolos independientes conceptualmente, pero dependientes lógicamente.

Los servicios de usuario, se encuentran en la máquina del cliente, los servicios de negocios y de datos se encuentran encapsulados como objetos

bajo la tecnología Component Object Model (*COM*) dentro de uno o más servidores y la base de datos puede estar en estos servidores o en uno propio.

Beneficios:

Al estar aislados los servicios, el desarrollo de la aplicación se vuelve más fácil y rápido.

El mantenimiento de la aplicación es más sencillo, ya que sí necesitamos modificar alguna regla del negocio, algún acceso a datos o alguna pantalla nos dirigimos al servicio necesario.

Sí los servicios de datos o de negocio son modificados no tenemos que hacer una redistribución a los clientes, ya que estos servicios se encuentran en el servidor(es) y los clientes no sufrirán cambios a menos que sea necesario.

Sí estamos consientes de cómo se integran los diferentes servicios, y cómo se encuentran aislados, un cambio en cualquiera de ellos no debe afectar a los otros.

Los servicios de negocios y los de datos, se encuentran encapsulados en diferentes objetos *COM*, por lo tanto podemos hacer uso de la reutilización de estos objetos para poder hacer uso de ellos desde otros clientes. Actualmente, existen diferentes lenguajes de programación que soportan objetos *COM*.

Los diferentes servicios pueden ser desarrollados en distintos lenguajes e integrados posteriormente para otorgar la misma funcionalidad.

Mayor seguridad en la aplicación al proteger las capas de negocio y datos usando la herramienta Microsoft Transaction Server.

Escalabilidad de la aplicación. Entiéndase por escalabilidad, que si la aplicación fue diseñada para cien clientes dando una respuesta adecuada al integrar un mayor número de usuarios (2000 ó más) el tiempo de respuesta no se verá disminuido.

Simplifica el acceso a datos en diferentes bases de datos.

Todos estos beneficios nos llevan a una reducción significativa en el Costo Total de Propiedad (*TCO Total Cost Ownership*). Costo en desarrollo, distribución, mantenimiento, y evolución de la aplicación Cliente/Servidor.

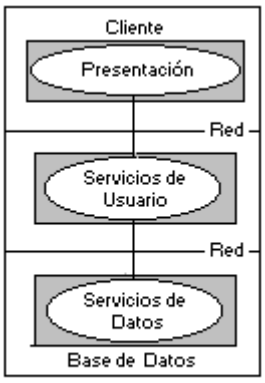


Fig. 1.3

¿Cómo funciona una aplicación de Tres Capas?

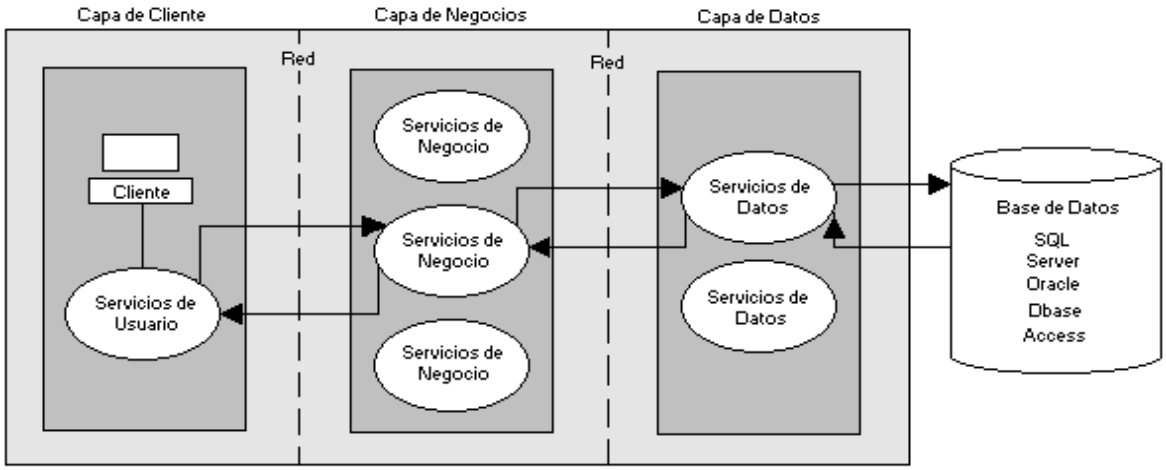


Fig. 1.4

Como lo podemos observar en la figura 1.4 el cliente pide cierta información, la capa de usuario manda llamar a la capa de negocios en donde se encuentran los servicios de negocio, éstos se encargan de validar la petición solicitada por el cliente, si ésta es incorrecta se generará un error que se enviará al servicio de usuario para presentarlo al cliente; si la petición es correcta se solicita a la capa de datos el servicio de datos adecuado para que éste a su vez obtenga la información de la base de datos, esta información viajará de la base de datos al servicio de datos, del servicio de datos al de negocios y éste la entregará al servicio de usuario el cual se encargará de presentar la información de forma adecuada al cliente.

Estrategia de Microsoft de una aplicación distribuida

Microsoft nos propone esta nueva arquitectura para poder reutilizar los servicios de negocios y datos por diferentes clientes.

Como se muestra en la figura 1.5 los clientes creados en diferentes lenguajes pueden utilizar los mismos objetos *COM* obteniendo la misma

funcionalidad e información encontrándose en una o diferentes bases de datos.

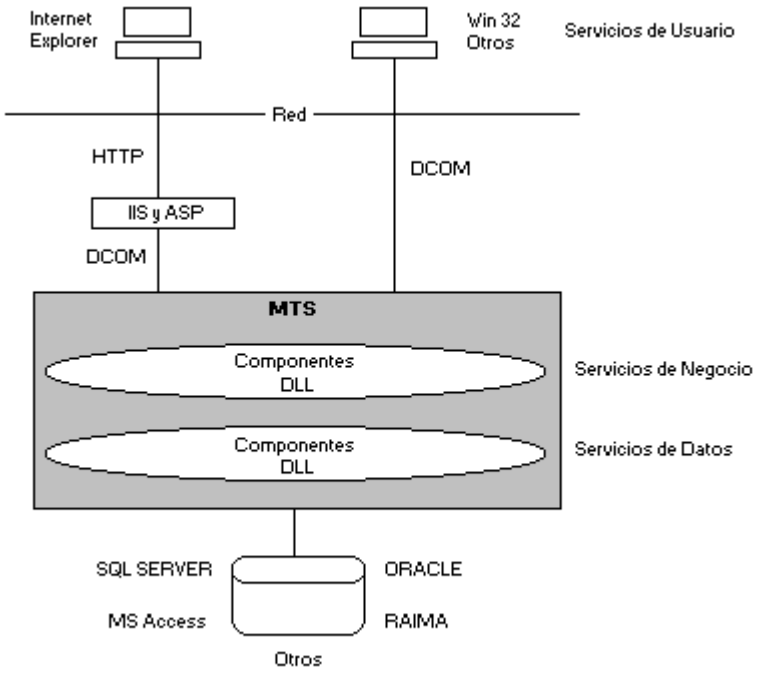


Fig. 1.5

Retomando el ejemplo del banco, sí se hubieran desarrollado las aplicaciones bajo la arquitectura de tres capas, los servicios de datos y de negocios podrían ser utilizados por clientes en red, cajeros automáticos e incluso en *INTERNET*, es decir la única que tendría que crearse para las diferentes tecnologías es la capa de usuario.

1.2. HERRAMIENTAS DE DESARROLLO

Actualmente existen diferentes herramientas para la creación de las diferentes capas que constituyen una aplicación, es fundamental mencionar que las herramientas deben soportar tecnología *COM*, a continuación se mencionan aquellas que son recomendadas por Microsoft.

Microsoft Visual Basic: Esta herramienta se cataloga como Rapid Application Development (*RAD*), es decir, ofrece un desarrollo rápido en la Interfaz de Usuario. Permite conectividad a Bases de Datos y en el podemos crear tecnología Active X (*COM DLL, COM EXE, COM OCX*).

Microsoft Visual C++: Nos ofrece un mayor performance en ejecución de código ya que con él tenemos un control absoluto de los procesos que corren dentro de la máquina. Podemos crear Interfaz de Usuario, Tecnología ActiveX (COM *DLL, COM EXE, COM OCX*), conectividad a Bases de Datos, además de un desarrollo Orientado a Objetos. Esta herramienta no se considera *RAD*.

Microsoft Visual InterDev: Tambien considerada Herramienta *RAD*, es utilizada para crear páginas *WEB*, dentro de esta herramienta podemos utilizar Tecnología *HTML*, *DHTML* e incluso *ASP*.

Estas herramientas se pueden encontrar por separado o en un Set llamado Microsoft Visual Studio.

Pero, cuáles son los fundamentos para utilizar estas aplicaciones?, veamos:

Los Servicios de Usuario los podemos crear en Microsoft Visual Basic, Microsoft Visual C++ o en Microsoft Visual InterDev.

Los Servicios de Negocios y Datos los podemos crear en Microsoft Visual Basic o en Microsoft Visual C++.

La herramienta elegida depende de muchos factores, por ejemplo: la experiencia del desarrollador en el lenguaje, la rapidez con que se necesite la aplicación, si se va a desarrollar para *WEB* o no, etc. Estas preguntas

son básicas en la primer fase de desarrollo de una aplicación ya que si un desarrollador tiene experiencia en lenguaje C, será más facil y rápido crear los servicios en Microsoft Visual C++ que en Microsoft Visual Basic o viceversa.

Como cualquiera de estas herramientas soportan tecnología *COM* podemos crear los Servicos de Datos y Negocios en Microsoft Visual Basic o en Microsoft Visual C++, estos servicios estarán encapsulados en componentes *COM DLL's*. Los Servicios de Usuario pueden ser creados en las diferentes herramientas mencionadas y pueden mandar a llamar a los componentes *COM DLL's* cuando sea necesario.

Imaginemos que tenemos un grupo de desarrolladores entre los cuales existen algunas personas que conocen Microsoft Visual Basic, otras conocen Microsoft Visual C++ y otras conocen Microsoft Visual InterDev y necesitamos una aplicación que permita a los Clientes de la empresa utilizar sus servicios a través de Internet y los datos de estos Clientes serán administrados por personal de la empresa en una aplicación que corra sobre la red.

La aplicación Web puede ser creada en Microsoft Visual InterDev utilizando la tecnología *ASP*, y la aplicación para los administradores de datos puede ser creada en Microsoft Visual Basic. La funcionalidad de las aplicaciones estará a cargo de los componentes *COM DLL's* creados en Microsoft Visual C++. Los Servicios de Usuario creados en Microsoft Visual InterDev y en Microsoft Visual Basic pueden compartir los componentes para obtener una funcionalidad en común sin necesidad de crear un componente diferente por cada cliente.

Ahora que ya conocemos la arquitectura de 3 capas, veamos como crear los componentes encargados de la lógica del negocio.

2. CREACIÓN DE COMPONENTES COM DLL CON VISUAL BASIC 6.0

De acuerdo a la arquitectura de una aplicación de tres capas, debemos crear toda la lógica funcional de los servicios de negocio dentro de componentes *DLL* (*Dynamic Link Library*).

En estos servicios se integran reglas de mercado y requerimientos del negocio y serán los encargados de la parte medular de la aplicación.

En éste capítulo veremos como implementar la lógica y como crear los componentes *DLL*.

2.1. IMPLEMENTANDO SERVICIOS DE NEGOCIO

Los Servicios de Negocio en una aplicación son la unidad lógica que controla las reglas del negocio, validaciones de información y la integridad transaccional de las operaciones que realiza el usuario. Se encargan de transformar los datos en información útil para la aplicación.

Estos servicios deben implementarse en la Capa de negocio de la aplicación, y estarán dentro de componentes *COM DLL´s* creados en cualquier lenguaje de programación que soporte esta tecnología.

2.2. COMPONENT OBJECT MODEL (COM)

La evolución de la arquitectura Cliente/Servidor, los continuos avances en las aplicaciones para Internet y el rápido crecimiento en hardware y

software han puesto a los desarrolladores un nuevo reto para vencer problemas tales como:

- El tiempo de desarrollo de una aplicación compleja.
- El costo de mantenimiento de estas aplicaciones.
- El costo en la distribución de la aplicación a los diferentes clientes, pudiendo ser una distribución local, nacional o internacional.
- El control de versiones en una aplicación.
- La imposibilidad de compartir código entre diferentes aplicaciones.

Para la solución de estos problemas Microsoft ofrece la tecnología *COM*, la cual se basa en software empacado que puede ser utilizado por más de una aplicación.

COM es un modelo de programación basado en objetos que ofrecen propiedades, métodos y eventos para su integración con los diferentes clientes que manipulen este software. Este modelo dicta la forma de crear y manipular los objetos bajo las responsabilidades de *COM*, pero no indica como se implementan esas responsabilidades, es decir, al mencionar la palabra CASA se puede imaginar una estructura donde puede habitar una familia pero no se sabe sabe cuantos cuartos, cuantos baños, etc. tiene esa casa. La manera de utilizar este modelo depende de la funcionalidad que requiera el desarrollador.

Las especificaciones de *COM* son:

- Como es instanciado un objeto.
- Como puede un cliente tomar la funcionalidad del objeto.

- El objeto debe destruirse por si mismo cuando no sea utilizado por un periodo de tiempo largo.

Ventajas:

- *Compatibilidad Binaria*. Los componentes *COM* pueden ser creados en diferentes lenguajes de programación al igual que los clientes que manipulan estos componentes, por lo tanto, un cliente creado en Visual C++ o una macro creada en Microsoft Office pueden utilizar un componente creado en Visual C++ o Visual Basic.
- *Interoperabilidad entre plataformas*. Un componente *COM* corriendo sobre windows 9? puede comunicarse con un componente de tecnología Common Object Request Broker Architecture (*CORBA*) corriendo sobre UNIX.
- *Reutilización de código*. Los componentes contienen grupos de métodos llamados interfaces. Estas interfaces pueden ser utilizadas por diferentes clientes evitando la duplicidad de código.
- *Transparencia en llamados*. Los clientes pueden levantar instancias de componentes que se encuentren en la misma máquina o en otra diferente. La forma de levantar la instancia de un componente remoto es la misma que se utiliza para levantar la instancia de un componente alojado en la misma estación de trabajo.
- *Control de Versiones*. Un cliente tiene un código fijo a través del cual toma la funcionalidad de un componente, cuando una nueva versión del componente es instalada, el cliente seguirá utilizando la misma funcionalidad que tomaba anteriormente sin verse afectado por los nuevos cambios. Éste deberá ser modificado y recompilado si desea tomar la nueva funcionalidad del componente. Ésta

ventaja la ofrecen los componentes gracias a sus interfaces. Una interfaz es fija, no puede ser modificada. Si se necesita nueva funcionalidad en el componente se debe agregar otra interfaz con la funcionalidad para el cliente, con esto se evita que los clientes de versiones anteriores sufran daños por los cambios a los componentes.

COM ofrece tres tipos de Componentes: *DLL, OCX y EXE*.

Dynamic Link Library (*DLL*). Corren de forma in-process, es decir, se ejecutan en el mismo espacio de memoria que la aplicación cliente dando como resultado una comunicación más rápida entre *DLL* y cliente. Por cada cliente que utiliza este componente se levanta una instancia de la *DLL*. El riesgo de utilizar este tipo de *COM* es que si el componente tiene errores fatales en run-time el cliente también tendrá conflictos causando que la aplicación termine.

Object Component Extension (*OCX*). Tienen el mismo comportamiento que los *DLL* sólo que contienen interfaz gráfica. Ejemplo: un control Grid que presenta datos.

Execute (*EXE*). Se ejecutan de manera out-of-process, es decir, tienen un espacio de memoria independiente al cliente. Una sola instancia de estos componentes le dan servicio a los diferentes clientes que lo requieran sin embargo la comunicación entre el componente y el cliente es más lenta. La ventaja que presentan con respecto a los *DLL* es que si llegan a tener errores fatales no se ve afectado el cliente.

Los servicios de negocio y datos de las aplicaciones de tres capas se encapsulan en componentes *COM*. Para que estos componentes puedan

darle funcionalidad a los clientes necesitan estar registrados bajo Microsoft Transaction Server (*MTS*).

Es requerimiento del *MTS* que los componentes *COM* sean *DLL*.

2.3. DESARROLLO DE UN COMPONENTE COM DLL

Para crear un COM DLL en Visual Basic 6.0, se debe elegir el tipo adecuado de proyecto *ActiveX DLL*.

Bajo el menú *Archivo* seleccione *Nuevo Proyecto*, de los diferentes proyectos que se presentan se debe elegir *ActiveX DLL*.

Los módulos principales de este tipo de proyecto, son las clases.

Un módulo de clase no contiene interfaz de usuario, sólo tiene código ejecutable. Los módulos de clase son plantillas para definir propiedades, métodos y eventos de un objeto. Un *ActiveX DLL* puede contener más de una clase para poder formar un modelo de objetos.

Cuando se hace la instancia de una clase en tiempo de ejecución se genera el objeto que dará servicio al cliente.

Cuando se selecciona un nuevo proyecto *ActiveX DLL* en Visual Basic, éste ya contiene un módulo de clase, si se desea agregar nuevos módulos de clase seleccione *Agregar Módulo Clase* en el menú *Proyecto*.

Propiedades de un proyecto ActiveX DLL

Cuando se desarrolla una *DLL* en Visual Basic, es necesario asignar ciertas propiedades básicas al proyecto.

Seleccionando el comando *Nombre_Proyecto Propiedades* del menú *Proyecto*. Bajo la pestaña *General* de la ventana *Propiedades del Proyecto,* se pueden asignar estas propiedades.

- *Tipo de Proyecto (Project Type)*. Permite seleccionar el tipo de proyecto con el cual queremos trabajar. Visual Basic selecciona automáticamente esta opción cuando elegimos el tipo de proyecto en la ventana *Nuevo Proyecto*.

- *Objeto Inicial (Startup Object).* Permite seleccionar el objeto con el cual se va iniciar la aplicación. Para la mayoria de las *DLL's* esta opción debe estar asignada a *Ninguno.* Si se necesita correr algún código de inicialización al cargarse la *DLL* es necesario seleccionar *Sub Main*.

- *Nombre del Proyecto (Project Name)*. Permite asignar el nombre al proyecto. Esta es la primer parte de la identificación de la clase que queremos utilizar. Por ejemplo, si el proyecto se llama Clientes y la clase tiene el nombre de Datos_Basicos, cuando utilicemos esta clase la tenemos que identificar como Clientes.Datos_Basicos.

- *Descripción del Proyecto (Project Description)*. Es utilizado para asignar una descripción del propósito del proyecto, esta descripción se presentará en la ventana *Referencias* cuando necesitemos asignar referencias de *DLL's* al proyecto cliente.

- *Actualización de Controles ActiveX (Upgrade ActiveX Controls)*. Asegura que cualquier control *ActiveX* utilizado por un proyecto será el más actual en caso de haber instalado diferentes versiones del mismo control en la estación de trabajo.

- *Ejecución desatendida (Unattended Execution)*. Habilita al componente a correr sin necesidad de interactuar con el usuario. Se puede utilizar en caso que necesitemos enviar mensajes al *Log de Eventos (Event Log*) de Windows NT sin necesidad de presentar al usuario la ventana habitual de la función *MsgBox*.

- *Modelo de Threads (Threading Model)*. Para elegir el comportamiento del componente *DLL* en el modelo de threads (*Single Threaded* o *Apartment Threaded*). Si el componente se desarrolla para una aplicación tres capas debe elegir *Apartment Threaded* debido a que cada cliente será atendido por la misma instancia del componente pero en localidades de memoria distintas.

Propiedades de una Clase

Para determinar como se crean las clases al dar servicio a un cliente se deben asignar ciertas propiedades básicas a cada una de las clases que se encuentran dentro de un componente *DLL*.

Para asignar las propiedades a una clase utilizamos la ventana de Propiedades una vez seleccionada la clase en la ventana de Proyectos.

- *Nombre (Name)*. Nombre con el cual se identificará la clase para su manipulación.

- *Instanciando (Instancing)*. Define la manera en que los clientes levantarán las instancias de las clases. Esta propiedad tiene cuatro opciones si el proyecto es tipo *DLL*:

- *Privado (Private)*. Las clases de la *DLL* sólo podrán ser instanciadas por módulos dentro de la misma *DLL*.
- *No Creable Públicamente (PublicNotCreatable)*. Los clientes de esta clase sólo podrán utilizarla si el componente crea primero al objeto. Esta opción es utilizada si se necesita crear un modelo de objetos dependientes.
- *Uso Múltiple (MultiUse)*. Permite que cualquier cliente genere instancias de la clase. Una instancia de la clase puede dar atención a diferentes clientes a la vez.
- *Uso Múltiple Global (GlobalMultiUse)*. Es parecida a la anterior, sólo que los métodos y propiedades se vuelven globales. Permite al cliente utilizar los métodos y propiedades de la clase sin crear explícitamente al objeto que los contiene.

Las clases utilizadas en aplicaciones tres capas deberán tener asigada la opción *MultiUse* a la propiedad *Instancing.*

Eventos de las Clases

Las clases contienen dos eventos definidos; *Initialize* y *Terminate*.

El evento *Initialize,* ocurre cuando se hace una instancia de la clase, y puede ser utilizado para inicializar variables vitales en el código.

El evento *Terminate*, ocurre cuando un objeto se destruye (La destrucción de un objeto sucede cuando esta fuera de su alcance o cuando el cliente que lo mando llamar sale del alcance del objeto o cuando el cliente explícitamente lo destruye asignándole *Nothing),* y es utilizado para salvar información o ejecutar sentencias de destrucción de variables.

Creación de Métodos en Clases

En Visual Basic se pueden agregar procedimientos tipo *Sub* ó *Function*. La diferencia que existe entre estos procedimientos es que los tipo *Function* regresan valores por si solos y los tipo *Sub* carecen de esta funcionalidad.

Cuando se agregan este tipo de procedimientos en una clase se les llama métodos. Para que los métodos puedan ser vistos por la aplicación cliente, es necesario declararlos de tipo *Public.*

Ejemplo: El siguiente método es utilizado para validar el identificador y el nombre del cliente que será insertado en la base de datos.

```
Public Function Insertar_Cliente (ByVal lngIdentificador as Long, ByVal strNombre as
String) as Long

    Código del método

End Function
```

Dentro de una clase pueden ser agregados propiedades y eventos; sin embargo, si el componente va a dar funcionalidad en una aplicación 3 capas, no se recomienda agregar los mismos, ya que éstos causarían menor rendimiento en la aplicación.

Disparando Errores al Cliente

Cuando desarrollamos componentes *DLL's*, es importante agregarles tantas trampas de error como sean necesarias, ya que si un error en tiempo de ejecución es disparado y la DLL no lo atrapa, ésta se destruirá terminando la comunicación entre ella y el cliente.

Todos los errores atrapados en los componentes, deben ser solucionados, algunos de estos errores necesitan la intervención del cliente para llegar a su solución y otros no.

Un componente *DLL* al validar información puede encontrar que estos datos son erróneos y debe dar aviso al cliente, la manera más sencilla de hacerlo es disparando errores personalizados del componente.

Utilizando el método *Raise* del objeto *Err* en Visual Basic, podemos disparar errores en tiempo de ejecución, para que sean atrapados por los clientes.

La sintaxis del método *Raise* es:

Err.Raise (*Número, Fuente, Descripción, Archivo de ayuda, Contexto de la ayuda*)

donde:

- *Número*, es el número de error que se va a disparar. Si el error es personalizado el número debe estar entre el 513 y 65,535, y se le suma la constante *vbObjectError*, con el fin de que los números de error sean únicos en la estación de trabajo donde estamos trabajando.

- *Fuente*, indica el origen del error. Nos sirve para proporcionarle al cliente información acerca del método donde fue disparado el error.
- *Descripción*, muestra la causa del error.
- *Archivo de ayuda*, ruta y nombre del archivo en donde se puede encontrar información para solucionar el error.
- *Contexto de la ayuda*, número que indica la posición dentro del archivo de ayuda donde se encuentra la información del error.

La *Fuente, Descripción, Archivo de ayuda y Contexto de la ayuda* son argumentos opcionales.

El siguiente ejemplo muestra como disparar un error en tiempo de ejecución en un método:

```
Public Function Insertar_Cliente (ByVal lngIdentificador as Long, ByVal strNombre as String) as Long
        Dim lngNumErr as Long
        On Error Goto Errores

        'Código del método
        Exit Function
Errores:
        'Si el error es personalizado
        lngNumErr = 520 + vbObjectError
        Err.Raise lngNumErr, "Insertar_Cliente", "Los datos proporcionados no son válidos"
        'Si existe un error en tiempo de ejecución
        lngNumErr = Err.Number
        Err.Raise lngNumErr, "Insertar_Cliente", Err.Description

End Function
```

Probando un proyecto ActiveX DLL

En Visual Basic ofrece la posibilidad de generar grupos de proyectos para poder realizar las pruebas necesarias antes de compilarlos.

Para probar un proyecto ActiveX DLL debemos:

1. Abrir el proyecto que se va a probar.
2. En el menú *Archivo*, seleccione el comando *Agregar Proyecto* y después selccione *Standard EXE*.
3. En la ventana de *Proyectos* seleccione el *Standard EXE*, de un click derecho y seleccione *Proyecto Inicial*.
4. En el proyecto *Standard EXE*, agregue la referencia hacia el proyecto *ActiveX DLL* dando un click en el comando *Referencias* del menú *Proyecto* y seleccionando el proyecto *ActiveX DLL*.
5. En el proyecto *Standard EXE* agregue los objetos necesarios para la prueba (controles, formas, etc).
6. En los eventos de los objetos agregue el código para hacer la instancia y llamar los métodos de la clase que desea probar.

   En el siguiente ejemplo contamos con un proyecto *ActiveX DLL* llamado Clientes, el proyecto contiene una clase llamada Datos_Basicos y la clase contiene un método llamado Insertar_Cliente.

   ```
   Dim objCliente as Clientes.Datos_Básicos
   Set objCliente = New Clientes.Datos_Basicos
   objCliente.Insertar_Cliente(15, “Hanoi Hernández”)
   ```

7. Corra la aplicación.

Compilación de un proyecto ActiveX DLL

Al terminar de desarrollar un proyecto ActiveX DLL, debemos compilarlo para generar el archivo DLL que será utilizado por los clientes.

Esta tarea se logra eligiendo el comando *Make* del menú *File* de Visual Basic.

Compatibilidad de Versiones

Si se va a compilar una nueva versión de una *DLL* ya existente, se debe determinar que tipo de compatibilidad mantendrá con el cliente que fue compilado utilizando la versión anterior de la *DLL*.

Cuando se compila una *DLL*, se agrega un identificador único (*GUID Globally Unique Identifier*) a cada una de las clases (*CLSID*) que la componen, al *Type Library* (*LIBID*) y también se agrega un *GUID* a cada una de las interfaces (*IID*) de las clases de la *DLL*. Las aplicaciones que utilizan esta *DLL* manejan los identificadores únicos para crear y manipular los objetos. Si los identificadores únicos de la *DLL* llegan a cambiar, los clientes de versiones anteriores tendrán problemas si no son actualizados.

La compatibilidad entre versiones es muy importante para aplicaciones Cliente/Servidor de arquitectura tres capas.

Para asignar la compatibilidad entre versiones en Visual Basic se debe:

1. Seleccionar el comando *Propiedades del Proyecto* del menú *Proyecto*.
2. Seleccionar la pestaña *Componente*.
3. Elegir la opción adecuada en la sección *Compatibilidad entre Versiones (Version Compatibility)*.

Existen tres opciones en la sección *Compatibilidad entre Versiones (Version Compatibility)*:

- *No Compatibilidad (No Compatibility)*. Cada vez que el proyecto sea compilado se generarán nuevos *CLSID's*, *LIBID's* y *IID's* causando que no exista compatibilidad alguna entre las diferentes

versiones de la *DLL* y provocando errores en los clientes no actualizados que manipulen la nueva versión de la DLL.

- *Compatibilidad de proyecto (Project Compatibility)*. Es utilizado sólo para pruebas de proyecto. Genera nuevos *CLSID's* y *IID's*, pero mantiene el mismo *LIBID*. Esta opción tampoco mantiene compatibilidad con versiones anteriores de la *DLL*. Esta opción esta seleccionada por default para el proyecto.
- *Compatibilidad Binaria (Binary Compatibility)*. Mantiene los mismos *CLSID's*, *LIBID's* y *IID's* guardando la compatibilidad entre versiones del componente *DLL*. Si el cliente no está actualizado con la nueva versión del componente, podrá seguir utilizando la funcionalidad anterior sin ningún problema, si el cliente necesita utilizar la nueva funcionalidad de la *DLL* deberá ser modificado y recompilado utilizando la nueva versión del componente. Para que esta compatibilidad tenga éxito, no se debe borrar ningún método o clase, tampoco se pueden modificar nombres de métodos, clases o tipos de datos de parámetros.

Cómo Utilizar Componentes desde un Cliente

El cliente de una *DLL* puede ser una aplicación *EXE*, otra *DLL*, o incluso un componente *OCX*.

Para que un cliente pueda utilizar los servicios de una *DLL* es necesario levantar la instancia de la clase que se va a utilizar, para ello se debe:

1. Abrir el proyecto del cliente.
2. Seleccionar el comando *Referencias* del menú *Proyecto*.
3. Seleccionar la *DLL* deseada en la ventana *Referencias* y pulsar *Aceptar*.
4. Agregue los objetos necesarios al cliente (controles, formas, etc).

5. En los eventos de los objetos agregue el código para hacer la instancia y llamar los métodos de la clase que desea probar.

En Visual Basic existen dos formas de levantar instancias de una clase, utilizando el operador *New* o la función *CreateObject*. La diferencia entre estos operadores es la velocidad con que logran levantar la instancia. El operador *New* ofrece una velocidad mayor al levantar la instancia debido a que no utiliza el COM standard.

El siguiente ejemplo genera una instancia de la clase Datos_Básicos utilizando el operador New:

```
Dim objCliente as Clientes.Datos_Básicos
Set objCliente = New Clientes.Datos_Básicos          'Levantando la instancia
objCliente.Insertar_Cliente(15, "Hanoi Hernández")
```

Utilizando la función *CreateObject*:

```
Dim objCliente as Clientes.Datos_Básicos
Set objCliente = CreateObject ("Clientes.Datos_Básicos")    'Levantando la instancia
objCliente.Insertar_Cliente(15, "Hanoi Hernández")
```

Las *DLL* creadas para aplicaciones de tres capas serán agregadas al *MTS* para dar los servicios a los clientes, los clientes de estas *DLL's* deberán levantar las instancias de sus clases utilizando la función *CreateObject.*

2.4. INTERFACES EN COMPONENTES

Una de las ventajas que ofrece *COM* son las Interfaces.

Una Interfaz se define como un grupo de funciones que ofrece funcionalidad a los clientes del componente.

Las interfaces tienen cuatro características básicas, estas son:

- Una interfaz es una clase abstracta. Una interfaz no puede ser instanciada por un cliente, debido a que una clase abstracta solamente contiene definiciones de métodos. Para que estos métodos ofrezcan funcionalidad la clase abstracta debe ser implementada en otras clases.
- Una interfaz no es un objeto. Ya que un objeto es la instancia de una clase, las interfaces no pueden ser objetos ya que no pueden ser instanciadas directamente desde los clientes, solamente son implementadas dentro de clases del mismo componente.
- Una interfaz tiene un identificador único (*IID*). Para evitar errores en los clientes que utilicen las interfaces del componente, cada interfaz tendrá su *IID* independiente, con el objeto de evitar errores al mandar a llamar dos interfaces diferentes con el mismo nombre en diferentes *DLL's*.
- Las interfaces no cambian. Para evitar errores de compatibilidad entre versiones de componentes las interfaces no pueden ser modificadas de versión a versión, de tal forma que si se necesita agregar nuevos métodos o nueva funcionalidad a métodos creados anteriormente tendremos que definirlos en una nueva interfaz.

En el siguiente ejemplo se muestra el concepto de interfaz:

Un objeto videocasetera tiene una interfaz básica que contienen las funciones Play, Pause, Stop, Rev, Fwd, y Videocassette. Si se le agregan más funciones tendrán que ser hechas en una nueva interfaz, para que el objeto pueda continuar llamándose videocasetera en un nuevo modelo (versión). Si la interfaz básica es cambiada y en vez de aceptar videocassette acepta DVD, el objeto videocasetera se convertirá en un

nuevo objeto llamado DVD y perderá compatibilidad con versiones anteriores.

Una interfaz debe ser tomada como un contrato entre el cliente y el componente, donde todo lo que se tiene en una versión debe conservarse sin modificaciones para la siguiente versión, lo nuevo debe agregarse en nuevas interfaces. Así el cliente de versiones anteriores no tendrá ningún problema al tomar la misma funcionalidad del componente, aunque éste, sea una nueva versión.

Al encapsular la funcionalidad en objetos accesados a través de interfaces, *COM* ofrece componentes abiertos y expandibles.

Creación de Interfaces

Para crear una interfaz se necesita una clase abstracta. Las clases abstractas son módulos de clase que no contienen funcionalidad, solamente contienen la definición de los métodos que serán implementados en las clases del componente.

Ejemplo:

A continuación se define una interfaz llamada IDatosGenerales con dos métodos llamados DatosPersonales y DatosLaborales:

1. En un proyecto *ActiveX DLL*, agregue un módulo de clase.
2. En la propiedad *Name* de la clase asigne IDatosGenerales.
3. En la propiedad *Instancing* seleccione la opción *PublicNotCreatable*, con el fin de evitar que los clientes puedan levantar instancias de esta clase.

4. En la ventana de código, defina los métodos DatosPersonales y DatosLaborales, como se muestra a continuación:

```
Public Function DatosPersonales (ByVal Identificador as Long) as ADODB.Recordset
End Function

Public Function DatosLaborales (ByVal Identificador as Long) as ADODB.Recordset
End Function
```

Los métodos DatosPersonales y DatosLaborales regresan un *Recordset* al cliente con la información personal y laboral del cliente solicitado.

Implementación de la Interfaz

Una vez creada la clase abstracta, se implementa dentro de las clases que le darán la funcionalidad a los métodos definidos. Para ello se utiliza la sentencia *Implements* en la ventana de código de la clase donde se utilizará la intefaz.

Ejemplo:

En una clase llamada Clientes se implementa la interfaz IDatosGenerales.

1. Dentro de la sección *Declarations* de la ventana de código de la clase Clientes definamos:

```
Implements IDatosGenerales
```

   Una vez implementada la interfaz, dentro del combo *Object* de la ventana de Código se crea el objeto *IDatosGenerales.* Al seleccionar este objeto, en el combo Procedure aparecen los dos métodos definidos en la clase abstracta.

2. Seleccione cada uno de los métodos de la interfaz y agregue el código funcional.

```
Public Function IDatosGenerales_DatosPersonales (ByVal Identificador as Long) as ADODB.Recordset
        'Implementar
        'código
        'funcional
End Function

Public Function IDatosGenerales_DatosLaborales (ByVal Identificador as Long) as ADODB.Recordset
        'Implementar
        'código
        'funcional
End Function
```

Uso de Intefaces desde un Cliente

Una vez creada la DLL y sus interfaces, se utilizan en los clientes.

El siguiente código muestra la manera en que el cliente puede llamar la interfaz deseada para utilizar sus métodos:

```
Dim recPersonal as ADODB.Recordset
Dim recLaboral as ADODB.Recordset
Dim objClientes as NombreProyecto.Clientes                    'Clase
Dim objDatosGenerales as NombreProyecto.IDatosGenerales       'Interfaz

Set objClientes = CreateObject("NombreProyecto.Clientes")     'Crea instancia de la clase
Set objDatosGenerales = objClientes                           'Llama a la Interfaz

Set recPersonal = New ADODB.Recordset
Set recLaboral = New ADODB.Recordset

'Llamando métodos
recPersonal = objDatosGenerales.DatosPersonales (100)
recLaboral = objDatosGenerales.DatosLaborales (100)
```

Ventajas de las Interfaces

Una interfaz puede implementarse en una o varias clases con funcionalidad diferente para cada uno de sus métodos. Esto es llamado *Polimorfismo* y se define como la llamada al mismo método esperando resultados diferentes.

En el ejemplo anterior se mostró como implementar la interfaz IDatosGenerales en la clase Clientes para obtener los datos personales y laborales de un cliente. Bien, si se implementa la misma interfaz en otra clase de la *DLL* llamada Proveedores y se agrega a sus métodos código de consulta a las tablas de proveedores de la base de datos, se obtiene información diferente llamando a los mismos métodos.

Ejemplo:

```
Dim recCliePersonal as ADODB.Recordset
Dim recClieLaboral as ADODB.Recordset
Dim recProvPersonal as ADODB.Recordset
Dim recProvLaboral as ADODB.Recordset

Dim objClientes as NombreProyecto.Clientes                    'Clase Clientes
Dim objProveedores as NombreProyecto.Proveedores              'Clase
Proveedores

Dim objDatosGenerales as NombreProyecto.IDatosGenerales       'Interfaz

Set objClientes = CreateObject("NombreProyecto.Clientes")     'Crea instancia
de la clase
Set objProveedores = CreateObject("NombreProyecto.Proveedores") 'Crea instancia
de la clase

Set objDatosGenerales = objClientes                           'Llama a la
Interfaz

Set recCliePersonal = New ADODB.Recordset
Set recClieLaboral = New ADODB.Recordset
Set recProvPersonal = New ADODB.Recordset
Set recProvLaboral = New ADODB.Recordset

'Llamando métodos
recCliePersonal = objDatosGenerales.DatosPersonales (100)
recClieLaboral = objDatosGenerales.DatosLaborales (100)
```

```
Set objDatosGenerales = objProveedores                    'Llama a la Interfaz

Set recPersonal = New ADODB.Recordset
Set recLaboral = New ADODB.Recordset

'Llamando métodos
recProvPersonal = objDatosGenerales.DatosPersonales (20)
recProvLaboral = objDatosGenerales.DatosLaborales (20)
```

Otra ventaja que ofrecen, es que una clase puede implementar varias interfaces. Sólo debemos utilizar la sentencia *Implements NombreInterfaz* cada vez que se desee implementar una nueva.

```
Implements IDatosGenerales
Implements Interfaz2
Implements Interfaz3
```

Esto permite que exista encapsulamiento de código y compatibilidad entre versiones.

2.5. REGISTRO DE UN COMPONENTE COM DLL

Antes de que un cliente pueda utilizar un *COM DLL* se debe registrar en el sistema Windows de la estación de trabajo.

A continuación se muestran las formas de registrar una *DLL*:

- Con un programa de Instalación.
- Compilando el proyecto *ActiveX DLL* en Visual Basic se registra la *DLL* automáticamente en la estación de trabajo.
- Con el comando *Regsvr32.exe*.

```
Regsvr32.exe "C:\NombreDLL.dll"
```

- Cuando se agrega una *DLL* al *MTS*, automáticamente se registra en el servidor donde está corriendo el *MTS*.

Si se necesita desregistrar una *DLL* del sistema de la estación de trabajo se debe correr el comando *Regsvr32.exe /u*.

```
Regsvr32.exe /u "C:\NombreDLL.dll"
```

De acuerdo a este capítulo, la implementación de todas las reglas del negocio debe hacerse bajo los componentes *DLL*. Por si solos, estos componentes pueden dar servicio a una o más aplicaciones, sin embargo, debe recordarse que las aplicaciones de tres capas pueden dar servicio a muchos clientes al mismo tiempo. Para que estos componentes soporten automáticamente la concurrencia de usuarios debemos utilizar un software llamado *MTS*.

En el siguiente capítulo se presenta el *MTS*.

3. MICROSOFT TRANSACTION SERVER (MTS)

Las aplicaciones de tres capas deben soportar la concurrencia de clientes, debido a que deben dar servicio a muchos de ellos al mismo tiempo. Generalmente, estas aplicaciones también deben manejar seguridad y transacciones entre otras cosas.

Existe un software de Microsoft llamado MTS, él se encarga de proporcionar a estas aplicaciones los requerimientos necesarios sin necesidad de escribir código alguno en la aplicación.

3.1. ¿QUÉ ES MTS?

Las aplicaciones Cliente/Servidor están compuestas por una serie de servicios para el manejo de múltiples usuarios al mismo tiempo. Estos servicios deben manejar recursos como conexiones a la Base de Datos, transacciones y seguridad, entre otros.

El MTS es un software basado en un sistema de procesamiento de transacciones que puede crear, distribuir y administrar los componentes de un sistema Cliente/Servidor. Al basarse en un sistema de transacciones se asegura que la manipulación de la información de una Base de Datos se llevará a cabo satisfactoriamente, es decir, o se realiza completa una acción o se cancela la acción. Con ello se puede certificar que la información en la Base de Datos será consistente. Las transacciones en el *MTS* son administradas por *el Microsoft Distrubuted Transaction Coordinator (DTC)*.

Las aplicaciones de 3 capas deben ser capaces de soportar llamadas de diferentes clientes al mismo tiempo. El *MTS* maneja el *Modelo de*

Concurrencia, el cual está basado en Actividades. Una actividad es un plan de ejecución que ocurre desde que un cliente crea un componente *MTS* hasta que lo destruye. Durante la actividad, el cliente puede hacer todas las llamadas que desea al componente teniendo una respuesta inmediata. Ya que el *MTS* administra componentes Apartment Threaded, el modelo asegura la creación de una sola instancia del componente *MTS* para la atención de todos los clientes, pero a cada cliente se le reservará un thread diferente para su atención.

Es necesario manejar una buena *Escalabilidad* (entiéndase por escalabilidad, el crecimiento de clientes en una aplicación sin tener un gran impacto en el performance de la misma) en una aplicación Cliente/Servidor. Para ello se deben manejar recursos compartidos como conexiones en red, conexiones a Base de Datos, memoria y espacio en disco y utilizarlos sólo cuando sea necesario. El *MTS* puede administrar estos recursos compartidos utilizando el *Just-in-Time Activation* y el *Connection Pooling*.

- *Just-in-Time Activation*. Cuando un cliente manda a llamar el componente *MTS*, éste es creado en el servidor permaneciendo inactivo hasta que él mismo u otro cliente mande llamar un método de este componente, en ese momento el componente es activado y permite que el llamado al método proceda. Cuando la ejecución de ese método termina, el objeto se desactiva y espera otro llamado a algún método para activarse. El objeto será destruido por el *MTS* o por el cliente siempre y cuando no se encuentre activo atendiendo a otro cliente.
- *Cola de Conexiones (Connection Pooling)*. Es un espacio en memoria del servidor que almacena las conexiones a la Base de Datos que no se están utilizando, así, cuando un cliente desea conectarse a la base de datos, el *connection pooling* puede asignarle rápidamente una conexión válida para ese cliente. Si

ninguna conexión cumple con las propiedades del cliente se creará una nueva. El *connection pooling* destruye las conexiones no utilizadas en cierto tiempo.

La seguridad juega un papel muy importante en las aplicaciones Cliente/Servidor, el *MTS* también maneja y administra la seguridad. Ésta debe ser a través de Roles y puede ser manejada en forma declarativa o programática. En el capítulo siete se verá como implementar la seguridad.

El *MTS* puede administrar servicios que no son propiamente de él, sin embargo, al ser parte de las herramientas de Windows NT Server, puede manejarlos automáticamente sin tener problemas de comunicación. Estos servicios se dividen en:

- *Resources Managers.* Son servicios que permiten almacenar información si ocurre algún crash en el sistema. Estos servicios aseguran la consistencia de la información. Algunos ejemplos son SQL Server 6.5 y Oracle 7.3.3.
- *Resources Dispensers*. Son aquellos servicios que manejan información útil para la aplicación pero si ocurre algún crash no mantienen los datos. Por ejemplo ODBC resource dispenser (*Connection Pooling*).

3.2. ARQUITECTURA DEL MTS

El *MTS* se compone de Paquetes y Componentes.

Los Paquetes son grupos de componentes que ejecutan tareas relacionadas.

Los Componentes son las DLL's que representan las capas de Negocios y Datos en un sistema Cliente/Servidor de 3 Capas.

Para administrar los componentes y paquetes necesitamos utilizar el *MTS* Explorer.

MTS Explorer

Corre bajo la Consola de Administración (Microsoft Management Console MMC) y es una herramienta que sirve para la creación y administración de Componentes y Paquetes. Dentro de él se pueden asignar propiedades a los componentes y paquetes, distribuir paquetes, monitorear transacciones e instalar paquetes y darles mantenimiento.

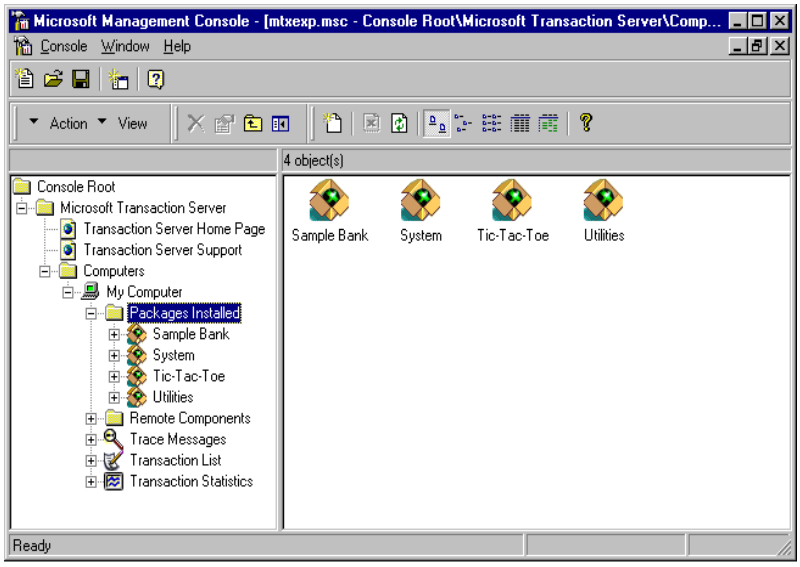


Fig. 3.1

Como explorador, permite administrar sus objetos en diferentes vistas que modifican la información en pantalla de acuerdo al análisis que deseamos. Las vistas permitidas dependen del objeto seleccionado.

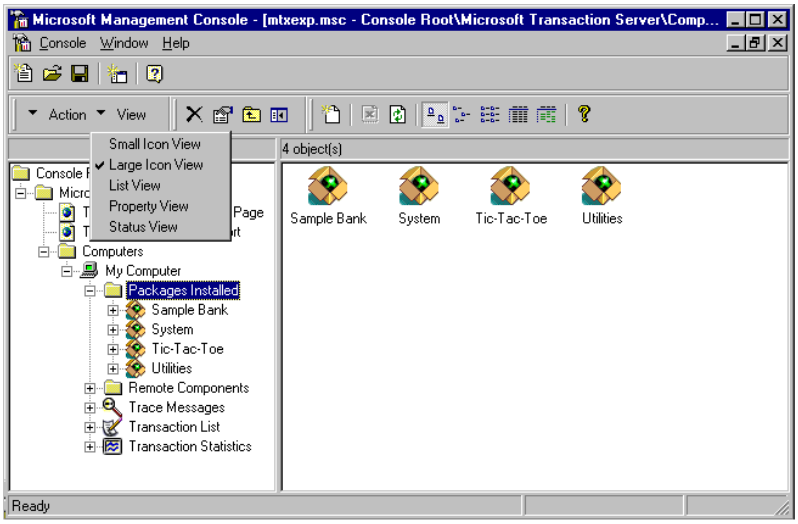


Fig. 3.2

Cada objeto dentro del explorador tiene propiedades, algunas son compartidas y otras son propias del objeto. Si se desean observar las propiedades de estos objetos, se puede seleccionar el objeto y la vista Propiedad (*Property*) del menú Vista (*View*). A continuación se presenta una tabla donde se muestran las propiedades desplegadas para cada uno de los objetos.

Objeto	**Propiedades**
Computers	Name, Timeout
Packages Installed	Name, Security, Authentication, Shutdown, Run Always, Account, Package ID, Activation
Components	Prog ID, Transaction, DLL, CLSID, Threading, Security
Roles	Name, Role ID
Interfaces	Name, Interface ID, Proxy DLL, TypeLib File
Methods	Name
Role Membership	Name, Role ID

Creación de Paquetes

Un Paquete es un grupo de componentes DLL's con tareas relacionadas. Representa a un proceso que administra y da seguridad a los componentes DLL's de una aplicación de 3 capas.

Cada paquete esta aislado con el motivo de no afectar a otros si algún paquete llega a sufrir errores en tiempo de ejecución.

Para crear un paquete de debe:

1. Seleccionar la carpeta u objeto Paquetes Instalados (*Package Installed*).
2. Dar un click en el comando Nuevo (*New*) del menú Acción (*Action*), y después en el comando Paquete (*Package*). O bien, dar un click derecho sobre la carpeta Paquetes Instalados (*Package Installed*) y seleccionar el comando Nuevo (*New*), después dar un click en el comando Paquete (*Package*).
3. En el Asistente del Paquete, dar un click sobre Crear un Paquete Vacío (*Create an Empty Package*).
4. Asignar el nombre del paquete y pulsar Siguiente (*Next*).
5. En la caja de dialogo Asignar la Identidad del Paquete (*Set Package Identity*), seleccionar la opción Cuenta (*Account*) adecuada y pulsar Terminar (*Finish*). En el capítulo 7 se verán a detalle estas opciones.

Propiedades de los Paquetes

Las propiedades de los paquetes permiten definir como se levantará la instancia del paquete, activar o desactivar la seguridad en el paquete, el tiempo máximo de ejecución, entre otras.

Para asignar las propiedades se selecciona el paquete y con botón derecho se da un click en el comando Propiedades (*Properties*). Se presentará el siguiente cuadro de dialogo:

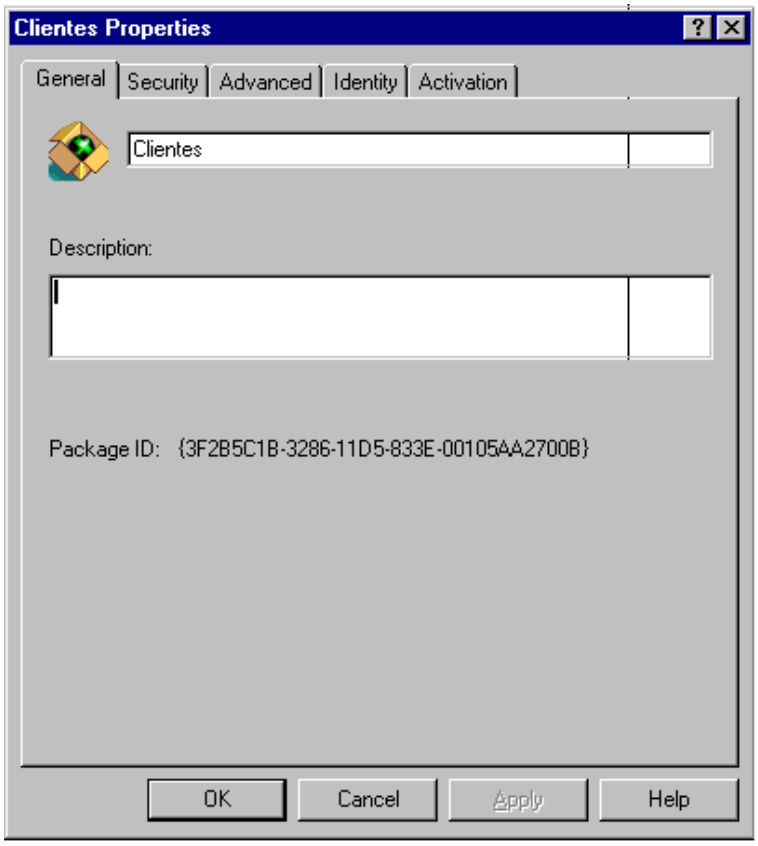


Fig. 3.3

Bajo la pestaña *General*, se puede modificar el nombre del paquete y asignarle una descripción. Además presenta el ID del Paquete (*Package ID*).

El tabulador Seguridad (*Security*), permite habilitar la seguridad del paquete y el nivel deseado.

En la opción Avanzado (*Advanced*), se determina el tiempo máximo que un proceso del servidor asociado al paquete puede estar activo. Puede ser un periodo de tiempo o un tiempo indefinido. También permite decidir si se asegura este cuadro de dialogo de tal forma que no permita cambios posteriores en las propiedades.

La más importante en cuanto a conexión a Base de Datos es la opción Identidad (*Identity*), ya que es donde a través de la propiedad Cuenta (*Account*), se asigna la cuenta que estará accesando a la Base de Datos.

- *Usuario Interactivo (Interactive User)*. Levanta la conexión a la Base de Datos con la cuenta del cliente que realizó la llamada.
- *Este Usuario (This User)*. Asigna una cuenta en específico que utilizará la conexión a la Base de Datos para todos los clientes del paquete.

En la pestaña Activación (Activation) se decide el lugar en donde se levantará la instancia del paquete.

- *Paquete Librería (Library Package). El paquete será creado en el proceso del cliente que lo llamó, lo cual beneficia en velocidad de ejecución, sin embargo no ofrece seguridad y no permite el aislamiento de los paquetes.*
- *Paquete Servidor (Server Package). Los paquetes serán creados en el servidor para tomar todos los beneficios que ofrece el MTS (recomendado).*

Creación de Componentes MTS

Un componente MTS es un componente DLL agregado a un paquete MTS.

Una vez creado el paquete, se pueden agregar los componentes DLL's que serán agrupados dentro de ese paquete. Un componente sólo se podrá ser agregado a un paquete en un servidor.

La forma de agregar un componente DLL a un paquete es la siguiente:

1. Dar doble click en el paquete que se desea.
2. Seleccionar con botón derecho la carpeta Componentes (*Components*), dar un click en el comando Nuevo (*New*) y después en el comando Componente (*Component*).
3. Del asistente, seleccionar Instalar Nuevos Componentes (*Install New Component(s)*).
4. De la caja de dialogo Seleccionar los Archivos a Instalar (*Select Files to Install*), seleccionar Agregar Archivos (*Add Files*).
5. Seleccionar el(los) componente(s) adecuado(s) y dar un click en Abrir (*Open*).
6. Dar un click en Terminar (*Finish*).

Propiedades de los Componentes

Los componentes MTS también tienen una serie de propiedades que dictan su comportamiento en el ambiente de la aplicación.

Estas propiedades se pueden asignar dando un click derecho sobre el componente MTS y selecciando el comando Propiedades (*Properties*) (figura 3.4).

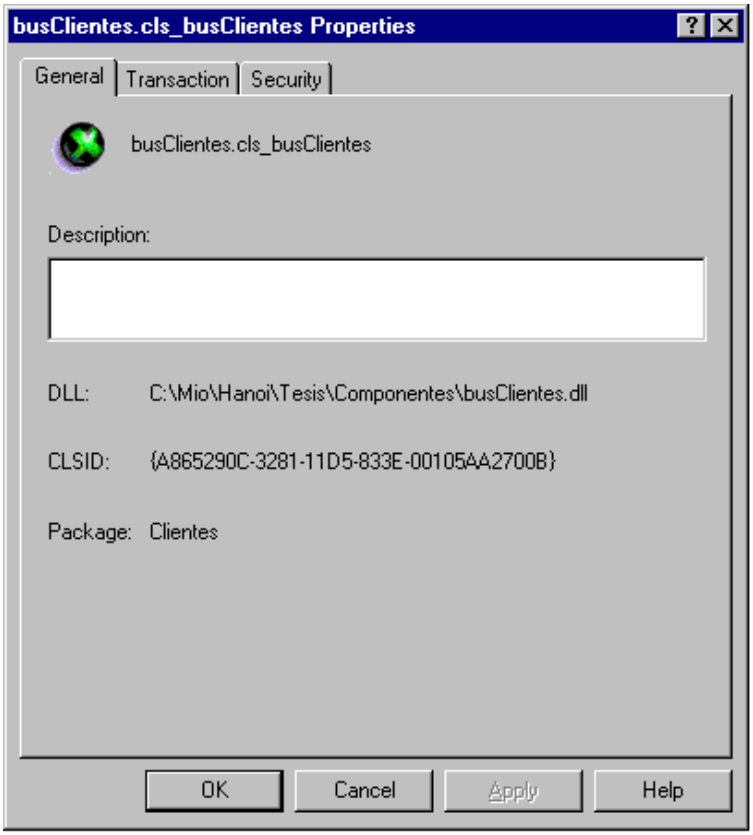


Fig. 3.4

La opción *General*, permite consultar datos propios del componente y asignar una descripción.

La opción Transacción (*Transaction*), define el tipo de comportamiento transaccional que tendrá el componente MTS al correr bajo el proceso.

Por último, en Seguridad (*Security*), se habilita o deshabilita la seguridad a nivel de componente.

Hasta este momento, se ha mostrado como crear un componente y como darlo de alta dentro del *MTS*, en este punto, el componente ya puede empezar a utilizar automáticamente algunos recursos proporcionados por el

MTS, sin embargo si se desea tomar al 100% la funcionalidad del software y controlar esos recursos entonces se debe agregar código a los componentes para que puedan integrarse completamente al *MTS*.

4. CREACIÓN DE COMPONENTES COM DLL PARA EL MTS UTILIZANDO MICROSOFT VISUAL BASIC 6.0

Al momento de agregar un componente DLL en el *MTS*, el componente toma automáticamente los servicios que el *MTS* ofrece, sin embargo, algunos de estos servicios necesitan programación adicional dentro del componente para integrarlo completamente con los recursos.

En este capítulo se verá como integrar las transacciones, como manejar el estado de los objetos, como atrapar los errores en los componentes y como configurar a los clientes para que utilicen estos componentes.

4.1. TRANSACCIONES

Una transacción, es un grupo de cambios que se le hacen a los datos con una función implícita, o se realizan todos o no se realiza ninguno.

Una transacción tiene las siguientes propiedades:

- *Atomicidad*. Todos las funciones de los objetos agrupados bajo una transacción son una unidad, si una no se cumple, ninguna se cumplirá.
- *Consistencia*. Asegura la consistencia en la información de la Base de Datos.
- *Aislamiento*. Cada transacción es independiente, si existe más de una transacción ejecutándose y alguna de ellas falla, las otras no se verán afectadas.

- *Durabilidad*. Gracias a los Log Transaccionales (SQL Server 6.5 por ejemplo), si el sistema llega a fallar, las modificaciones a los datos pueden ser recuperadas al ser restablecido el sistema.

En una aplicación 3 capas, la mayoría de funciones que interactúan con la Base de Datos deben ser controladas por transacciones. El *MTS* tiene la capacidad de tratar grupos de funciones de diferentes componentes MTS en una misma transacción, con la meta de llevar a un fin satisfactorio el intercambio de información con la Base de Datos evitando inconsistencia de datos.

Por ejemplo, un cliente necesita facturar productos. El cliente deberá llamar una función que agregue los datos de la nueva factura a la Base de Datos y además, implícitamente, se debe llamar a una función que elimine del inventario los productos que están saliendo. Estas dos funciones deberán estar agrupadas bajo una misma transacción, con el objetivo de asegurarnos que los procesos se llevarán a cabo totalmente, si alguno de los dos falla, ninguno se verá reflejado en la Base de Datos.

En el *MTS*, las transacciones son administradas por el servicio Coordinador de Transacciones Distribuidas (*DTC Distributed Transaction Coordinator*). Este servicio se encarga de asociar las transacciones que se ejecutan en uno o más de un servidor.

Una transacción en el *MTS* comienza cuando un cliente manda a llamar un componente MTS. El componente MTS creado se considera el Objeto Principal, ya que fue el primero en crearse, si el Objeto Principal manda a

llamar otros componentes y los asocia, estos objetos estarán trabajando de acuerdo al Objeto Principal.

Una transacción MTS llegará a su fin cuando:

- el tiempo de ejecución de la transacción termine (el tiempo de ejecución de las transacciones puede ser modificado en la propiedad Timeout de la Transacción (*Transaction Timeout*) de la pestaña Opciones (*Options*), en las propiedades del objeto Mi Computadora (*MyComputer*) en el explorador de el *MTS*).
- el Objeto Principal llama al método *SetComplete* o *SetAbort* del objeto *ObjectContext* (más adelante se dará una explicación de estos métodos).
- el cliente destruya al Objeto Principal.

4.2. USANDO LOS SERVICIOS DE TRANSACCIONES DE MTS

Cuando un objeto es creado por un cliente para resolver alguna petición, se dice que ese objeto está trabajando bajo su *Contexto*. El *contexto* de los objetos es muy importante dentro de una aplicación tres capas, ya que el es el que se encarga de manejar las transacciones y avisarle al *DTC* si una transacción termino satisfactoriamente o debe ser cancelada.

El *MTS* permite agrupar los contextos de dos o más componentes MTS para trabajarlos bajo una misma transacción, así, si algún método llega a

fallar, todos los métodos de todos los componentes MTS que esten asociados bajo el mismo contexto y transacción se cancelarán.

Por ejemplo, si se desea agregar a la Base de Datos un nuevo Cliente, entonces se debe mandar a llamar al Componente de Negocios que se encargará de validar la información del nuevo cliente, si la información es válida, ese componente debe mandar a llamar al Componente de Datos, el cual insertará la información a la Base de Datos. Como esta operación debe ser manejada en una transacción, los contextos de los dos componentes deben asociarse para trabajar como uno sólo.

Al agregar un Componente DLL al *MTS*, podemos asignarle la forma en que se debe comportar con sus transacciones. Para ello, se debe accesar la opción Transacción (*Transaction*) de las Propiedades del Componente MTS (figura 4.1).

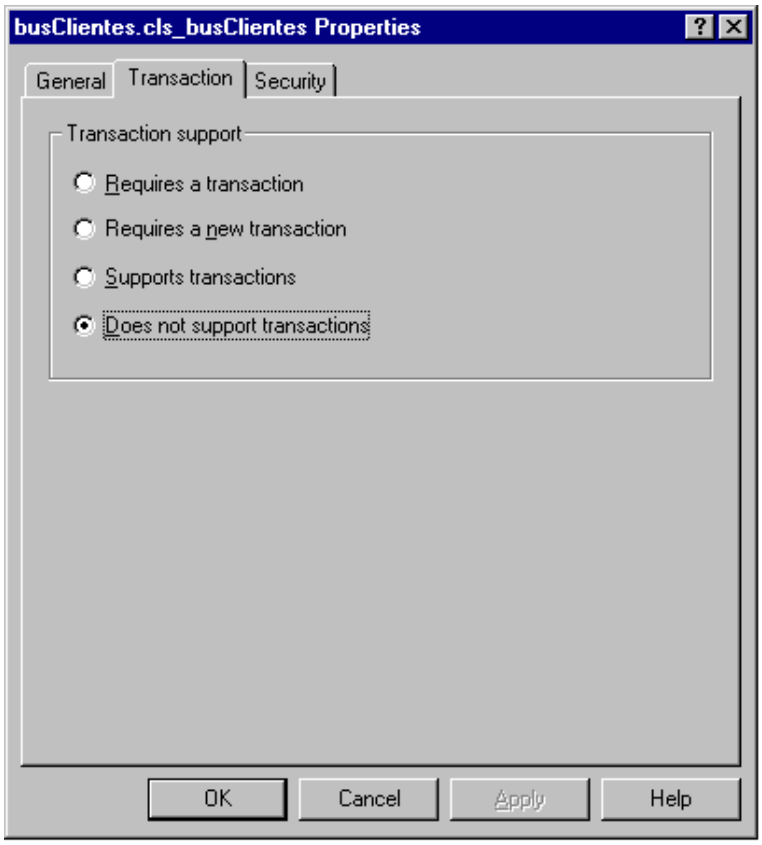


Fig. 4.1

Las opciones son:

- *Requiere una Transacción (Requires a transaction)*. El objeto debe trabajar bajo una transacción. Si el llamado se realizó desde un objeto que tiene transacción, entonces se asocian, si no, el objeto trabajará bajo una nueva transacción.
- *Requiere una Nueva Transacción (Requires a new transaction)*. El objeto estará en una nueva transacción, no importa si fue llamado desde otro objeto que este trabajando bajo una transacción.
- *Soporta Transacciones (Supports transactions)*. Si la llamada la realizó un objeto que tiene transacción, entonces se asocian, si no, el objeto será creado sin transacciones.

- *No Soporta Transacciones (Does not support transactions)*. El objeto trabajará sin transacciones.

Es importante determinar la forma en que trabajaran los componenetes bajo transacciones antes de crearlos, pues de eso depende que la información en la Base de Datos se mantenga sin errores.

Desarrollo de Componentes MTS

Para que un componente DLL tome todas las ventajas que ofrece el *MTS*, no basta con volverlo un componente MTS, se debe diseñar y programar para correr bajo el *MTS*.

A continuación se muestra una metodología sencilla para la creación de los componentes DLL:

1. Definir si el objeto trabajará bajo transacciones.
2. Hacer referencia a la librería de Microsoft Transaction Server para poder utilizar sus objetos.
3. Obtener la referencia hacia el *Contexto* del objeto.
4. Si el componente va a trabajar bajo transacción entonces se debe llamar a SetComplete (si tuvo éxito el método, la transacción se lleva a cabo) o a SetAbort (si el método falló, la transacción se cancela).
5. Agregar código para manejar el estado de los componentes.

Manejo de los objetos de MTS

Para poder crear componentes DLL's que correran bajo el MTS es necesario hacer referencia a la librería *mtxas.dll (Microsoft Transaction Server Type Library*) en el proyecto de Visual Basic.

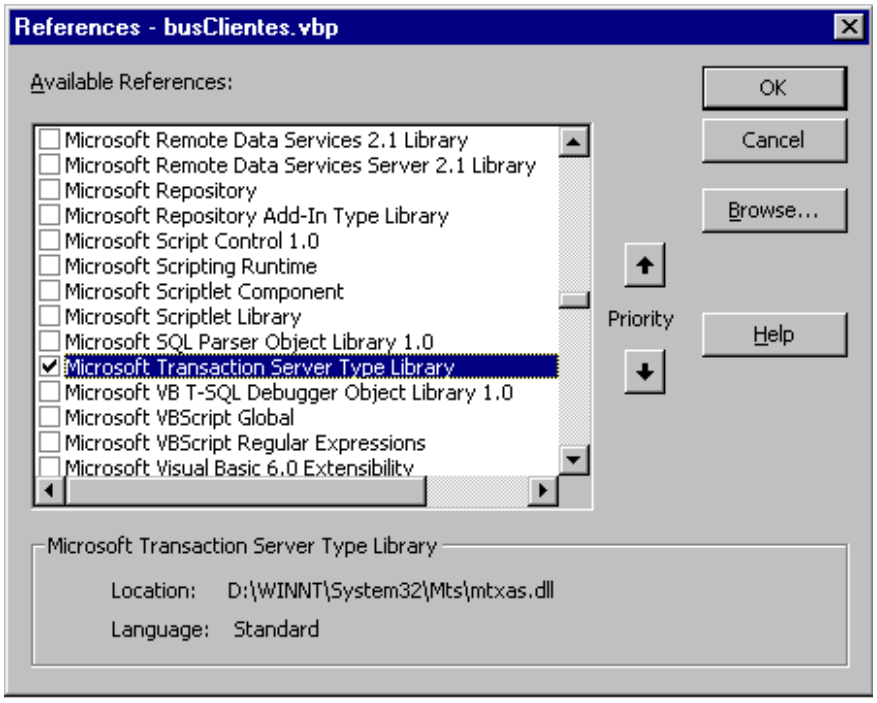


Fig. 4.2

Una vez hecha la referencia, se puede iniciar la construcción del componente utilizando los objetos de *MTS*.

Contexto del Objeto

El *Contexto* del objeto permite entre otros:

- Controlar al 100 % las transacciones.
- Conocer si el componente a sido creado bajo un proceso con seguridad o no.

- Saber cual es el cliente que mando a llamar al componente u objeto.
- Manejar seguridad programática.

Para poder manejar el *Contexto* del objeto se debe llamar a la función *GetObjectContext()*, que puede ser con:

```
Dim objctx as ObjectContext
Set objctx = GetObjectContext()
```

o también con:

```
GetObjectContext.Método
```

Ya sea con una variable apuntando al *Contexto* o con la función *GetObjectContext()*, se pueden manipular los métodos del *Contexto* del objeto.

Llamando Objetos desde un Objeto

En las aplicaciones de 3 capas, generalmente existen objetos que mandan llamar a otros y necesitan que sus contextos se asocien para participar en la misma transacción. Para lograr esto, el objeto 1 deberá mandar a llamar al objeto 2 con el método *CreateInstance* del objeto *ObjectContext* (recordemos que la propiedad Transacción (*Transaction*) del componente MTS debe estar asignada a Requiere una Transacción o Soporta Transacciones).

Los objetos también pueden ser creados con la función *CreateObject()*, pero a diferencia de *CreateInstance()*, esta función no tiene la capacidad de asociar contextos.

En el siguiente ejemplo, se muestra como el objeto de Negocio cls_*busClientes* en el método *Insertar_Cliente* manda a llamar al método Insertar_Clientes del objeto *cls_datClientes* (al ser funciones con tareas relacionadas sus contextos deben asociarse).

```
Public Function Insertar_Cliente(ByVal lngIdentificador As Long, ByVal strNombre
As String) As Long
   Dim datClientes As datClientes.cls_datClientes
   Dim lngError As Long
   On Error GoTo Errores
   'Validar Datos de Entrada

   'Llamar al componente y al método
   'que tiene el código para introducir
   'los datos a la base de datos.
   Set datClientes = GetObjectContext.CreateInstance("datClientes.cls_datClientes")
   Call datClientes.Insertar_Cliente(lngIdentificador, strNombre)

   Exit Function
Errores:
   lngError = 513 + Err.Number
   Err.Raise lngError + vbObjectError, "cls_busClientes.Insertar_Cliente"
End Function
```

Métodos SetComplete y SetAbort

Cuando uno o varios Componentes MTS estan corriendo bajo una transacción, se debe informar al *Contexto* si el(los) método(s) tuvieron éxito o no. El se encargará de ejecutar la transacción o de cancelarla.

Para informar del éxito o fallo de los métodos, es necesario mandar a llamar las funciones *SetComplete* y *SetAbort* del objeto *ObjectContext*.

- *SetComplete*. Informa que la ejecución del método no ha tenido errores y que la transacción puede llevarse a cabo.

- *SetAbort*. Permite comunicarle al *Contexto* que el método tuvo errores y que debe ser cancelada la transacción.

Ejemplo:

```
Public Function Insertar_Cliente(ByVal lngIdentificador As Long, ByVal strNombre As
String) As Long
   Dim datClientes As datClientes.cls_datClientes
   Dim lngError As Long
   On Error GoTo Errores
   'Validar Datos de Entrada

   'Llamar al componente y al método
   'que tiene el código para introducir
   'los datos a la base de datos.
   Set datClientes = GetObjectContext.CreateInstance("datClientes.cls_datClientes")
   Call datClientes.Insertar_Cliente(lngIdentificador, strNombre)

   GetObjectContext.SetComplete

   Exit Function
Errores:

   GetObjectContext.SetAbort

   lngError = 513 + Err.Number
   Err.Raise lngError + vbObjectError, "cls_busClientes.Insertar_Cliente"
End Function
```

Es recomendable que todos y cada uno de los métodos de los componentes MTS que participen o no participen en transacciones contengan los métodos SetComplete y SetAbort, con la finalidad de liberar los recursos del servidor.

4.3. MANEJANDO EL ESTADO DE LOS OBJETOS

El Estado de un Objeto se puede definir como la capacidad que tiene el objeto para almacenar la información que utilizan sus métodos.

Los componentes MTS pueden ser manejados como:

StateFull. Cuando un objeto es creado, el cliente puede almacenar dentro del objeto información a un nivel público del componente, la información

podrá ser utilizada por uno o varios métodos del componente hasta que el objeto sea destruido.

StateLess. Cada método de un componente deberá tener argumentos, a través de los cuales se pasará la información que ocupe ese método. Ningún método podrá utilizar información que no provenga de sus argumentos. La información sólo estará almacenada durante la ejecución del método, aún cuando el objeto no sea destruido.

El manejo del estado de los objetos es muy importante en una aplicación 3 capas, ya que influye de manera directa sobre la Escalabilidad de la aplicación.

Un objeto *StateFul* consumirá más recursos de memoria del servidor al dar servicio a diferentes clientes que un objeto *StateLess*.

En terminos generales, se recomienda que los componentes MTS sean *StateLess*.

Cuando un objeto es llamado por un cliente, el objeto es creado pero éste no se activa hasta que el cliente llame a un método. El objeto se desactivará cuando se manda a llamar al método *SetComplete* o *SetAbort*. Este proceso lo lleva a cabo el *Just-in-Time Activation* (*JIT*).

El *JIT* ayuda a administrar los recursos de memoria del servidor, debido a que el estado de los objetos es almacenado solamente durante el tiempo en que esten activos.

Si se desea agregar código de inicialización y término en las clases del componente MTS para que éste se ejecute al activarse o desactivarse un objeto, lo podemos hacer en los métodos *Activate* y *Deactivate* de la

interfaz *IObjectControl* que proporciona la librería de *MTS*. Estos métodos suplen a los eventos *Initialize* y *Terminate* de las clases.

Shared Property Manager (SPM)

A veces es necesario almacenar el estado de los objetos para compartirlo entre diferentes clientes durante la ejecución del objeto.

Por ejemplo:

> Si hay un componente MTS que se encarga de dar de alta nuevas facturas, el dato más importante para una nueva factura es el folio. Generalmente este identificador es asignado automáticamente por la aplicación.
>
> Debido a lo anterior, debe existir un método que vaya a la base de datos a consultar el último número de folio, esta información será recuperada por el componente para poder calcular el próximo número de folio.
>
> Si el componente sigue activo y necesita consultar el nuevo folio y el Estado del Objeto no fue almacenado, se necesita volver a realizar el proceso anterior. Ahora, si esta acción la realiza un componente que le esta dando servicio a un número indefinido de clientes al mismo tiempo, la aplicación reducirá su desempeño, impactando directamente en la escalabilidad del sistema.

El *SPM es* un modelo de objetos jerárquico que permite almacenar información en forma de propiedades y compartirla entre los métodos de los objetos en un mismo proceso dando servicio a diferentes clientes.

Cuando un objeto MTS se levanta, éste corre bajo un paquete de *MTS*, el *SPM* será creado a nivel paquete para permitir que los diferentes clientes que utilicen el objeto MTS puedan hacer uso de las propiedades del *SPM*.

Los objetos que componen al *SPM* son:

- *SharedPropertyGroupManager*. Permite la creación y uso de grupos de propiedades compartidas.
- *SharedPropertyGroup*. Utilizado para crear y manejar propiedades en los grupos existentes.
- *SharedProperty*. Empleado para asignar y recuperar valores en las propiedades compartidas.

SharedPropertyGroupManager, contiene:

- *CreatePropertyGroup.* Método que permite la creación de un Grupo compartido, si ese Grupo ya existe, devuelve la referencia hacia él.

  Sintaxis:

  CreatePrepertyGroup(Nombre, Bloqueo, Duración, Booleano)

  donde:

  - *Nombre*, es el nombre del nuevo Grupo.
  - *Bloqueo*, da seguridad al Grupo a través de las constantes *LockSetGet* (bloqueo a nivel propiedad) y *LockMethod* (bloqueo a nivel grupo). Evita que dos usuarios diferentes asignen valores al mismo tiempo.

- *Duración*, asigna el tiempo de vida del Grupo. Las constantes utilizadas son *Standard* (el Grupo se destruye automáticamente cuando no existe ninguna referencia hacia él) y *Process*(el Grupo tendrá vida hasta que el proceso donde se haya creado termine).
- *Booleano*, indica si un Grupo existe recuperando los valores *True* o *False*. Si el Grupo ya existe hace referencia hacia él, si no, crea el Grupo.

➢ *Grupo*. Propiedad que hace referencia hacia el Grupo especificado.

Sintaxis:

Group(Nombre)

donde:

- *Nombre*, es el nombre del Grupo deseado.

SharedPropertyGroup, contiene las siguientes propiedades y métodos:

➢ *CreateProperty.* Utilizado para crear una propiedad con identificador *String*, si la propiedad ya existe, devuelve la referencia hacia ella.

Sintaxis:

CreateProperty(Identificador String, Booleano)

donde:

- *Identificador String*, es el nombre de la nueva Propiedad.
- *Booleano*, indica si una Propiedad ya existe a través de *True* o *False*.

- *CreatePropertyByPosition*. Igual al anterior, sólo que el identificador será un tipo *Long*.

Sintaxis:

CreatePropertyByPosition(Identificador Long, Booleano)

donde:

- *Identificador Long*, es el índice de la nueva Propiedad.
- *Booleano*, indica si una Propiedad ya existe a través de *True* o *False*.

- *Property*. Hace referencia a una propiedad existente con identificador *String*.

Sintaxis:

Property(Identificador String)

donde:

- *Identificador String*, es el nombre de la nueva Propiedad.

- *PropertyByPosition.* Hace referencia a una propiedad existente con identificador *Long*.

  Sintaxis:

  PropertyByPosition(Identificador Long)

  donde:

  - *Identificador Long*, es el índice de la nueva *Propiedad*.

SharedProperty, contiene únicamente la propiedad *Value*. Esta propiedad es de tipo *Variant*.

Si se desea utilizar este modelo de objetos, se debe hacer referencia a la librería *mtxspm.dll (Shared Property Type Library)*, en el proyecto de Visual Basic.

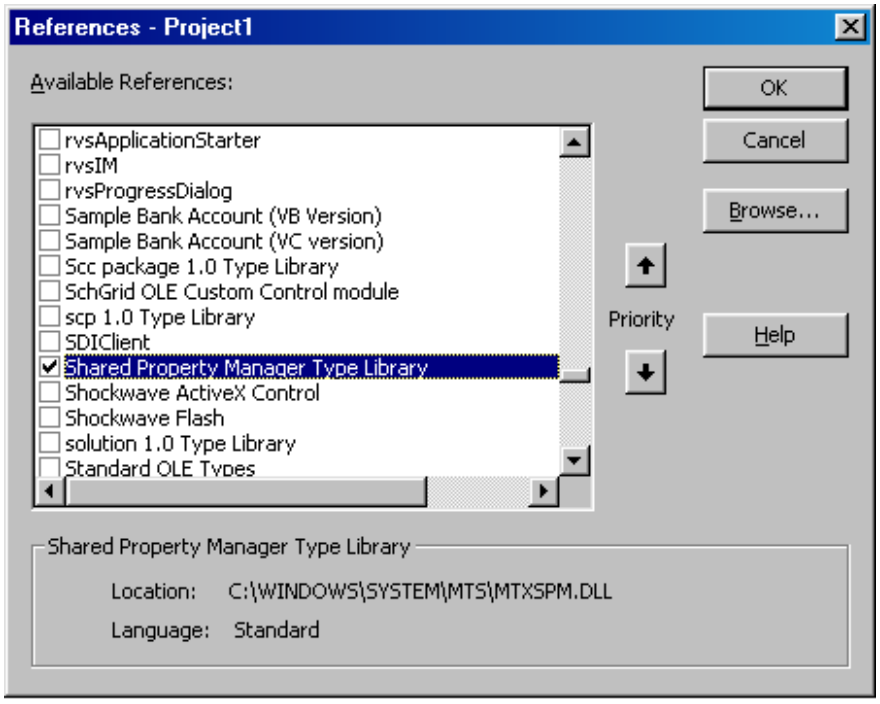


Fig. 4.3

4.4. MANEJANDO ERRORES EN COMPONENTES MTS

El buen manejo de errores en tiempo de ejecución es vital en cualquier aplicación, ya que evita acciones inesperadas que puedan corromper la información.

Los componentes MTS, también deben tener un robusto manejador de errores.

Como cualquier proyecto en Visual Basic, para atrapar errores en tiempo de ejecución se usa la sentencia *On Error GoTo*.

Ya que un componente MTS puede estar corriendo bajo transacciones, al atrapar un error, si es necesario, se debe cancelar la transacción y esto lo hace el método *SetAbort* del objeto *ObjectContext*.

Una vez atrapado el error y cancelado la transacción, se informa al cliente de lo ocurrido, para ello se utiliza el método *Raise* del objeto *Err*.

Ejemplo:

```
Public Function Insertar_Cliente(ByVal lngIdentificador As Long, ByVal strNombre
As String) As Long
   Dim datClientes As datClientes.cls_datClientes
   Dim lngError As Long

  ‘Trampa de Errores
   On Error GoTo Errores

   'Validar Datos de Entrada

   'Llamar al componente y al método
   'que tiene el código para introducir
   'los datos a la base de datos.
   Set datClientes = GetObjectContext.CreateInstance("datClientes.cls_datClientes")
   Call datClientes.Insertar_Cliente(lngIdentificador, strNombre)

   GetObjectContext.SetComplete

   Exit Function
```

```
Errores:

    'Cancela la transacción
    GetObjectContext.SetAbort

    lngError = 513 + Err.Number

    'Dispara el Error
    Err.Raise lngError + vbObjectError, "cls_busClientes.Insertar_Cliente"

End Function
```

Es recomendable que estas trampas de error existan en todos y cada uno de los métodos de un componente MTS. Así, si varios objetos están corriendo bajo un mismo contexto y uno de ellos presenta un error en tiempo de ejecución, inmediatamente cancelará la transacción evitando perder tiempo y liberando recursos útiles del servidor.

Existen diferentes herramientas, que permiten localizar e identificar errores que ocurren en la aplicación. Algunas de ellas son:

- *Visor de Eventos de Windows NT (Windows NT Event Log)*. Almacena información acerca de los diferentes errores que ocurren en el sistema, estos errores pueden ser arrojados por los componentes MTS o por cualquier otra aplicación. La información almacenada presenta datos acerca del origen y causa del error.
- *Espía MTS (MTS Spy)*. Utilizada para monitorear objetos MTS, permite conocer el curso de ejecución de tareas específicas, observar eventos de transacciones, métodos, objetos, usuarios, entre otros.
- *Explorador de MTS (MTS Explorer)*. A través de las ventanas Seguimiento de Mensajes (*Trace Messages*), Lista de Transacciones (*Transaction List*) y Estadísticas de Transacciones (*Transaction Statistics*), se puede monitorear el curso de las transacciones activas, saber cuantas transacciones tuvieron éxito o fracaso, etc.

4.5. CONFIGURACIÓN DE CLIENTES

La aplicación del cliente (Servicios de Usuario), estará utilizando los componentes MTS montados en el servidor. Esta aplicación creará los objetos de forma remota de manera local, sin darse cuenta que las DLL's utilizadas están siendo creadas en un servidor externo.

Los equipos donde se instalará la aplicación cliente, deben ser configurados para que la aplicación pueda comunicarse con los componentes MTS sin errores.

La configuración es muy sencilla, a continuación se muestran los pasos a seguir.

1. Exportar Paquetes.
2. Configurar Clientes.
3. Instalar DCOM.

Exportando Paquetes

Una vez creados los paquetes con sus respectivos componentes MTS, se deben exportar para poder configurar las estaciones de trabajo de los clientes.

Para exportar los paquetes debemos:

1. Dar click derecho sobre el paquete deseado y seleccionar el comando Exportar (*Export*).
2. Sobre la caja de diálogo Exportar Paquete (*Export Package*), escribir la Ruta donde será almacenado el paquete y el nombre de éste.

3. Seleccionar Salvar los Identificadores de los Usuarios de Windows NT Asociados con los Roles (*Save Windows NT user ids associates whit roles*), si se quieren almacenar los identificadores de los usuarios mapeados a los roles del paquete.
4. Dar un click en Exportar (*Export*).

Al exportar un paquete, en la ruta asignada se copian los componentes DLL's del paquete y se genera un archivo *.pak*, además de una carpeta llamada Cliente (*Client*).

El archivo *.pak*, contiene la configuración del paquete, es decir, las referencias hacia los componentes MTS del mismo, propiedades asignadas, Roles, etc. Si por un error borráramos un paquete del *MTS*, con este archivo lo podemos recuperar.

La Carpeta Cliente (*Client*), contiene un archivo ejecutable (.*exe*) que configura a los clientes para que utilicen los componentes de ese paquete.

Configurando Clientes

En las estaciones de trabajo donde estará corriendo la aplicación cliente, se debe ejecutar el archivo *.exe* almacenado en la carpeta Cliente (*Client*) creada al exportar el paquete.

Se recomienda que estos archivos se copien en un disco para su distribución a los clientes.

Estos ejecutables serán registrados en las estaciones de trabajo como programas remotos y podrán ser desinstalados desde Agregar o Quitar Programas (*Add/Remove Programs*) de Windows.

Los archivos de configuración de clientes *no deben ser ejecutados* bajo el servidor donde se encuentra el *MTS*, ya que estos modifican el Registro de Windows (*Registry*) del equipo borrando las referencias del paquete MTS que contiene a los componentes MTS. En caso de que se cometa el error, se deben desinstalar los archivos y regenerar el paquete de distribución.

Si la aplicación cliente también se instalará bajo el servidor donde está el *MTS*, no es necesario correr los archivos de configuración, ya que el equipo tiene configurados en su totalidad a los paquetes y componentes MTS de la aplicación.

Instalación del protocolo COM Distribuido (DCOM)

Los clientes de una aplicación 3 capas pueden estar sobre equipos Windows 95, Windows 98, Windows NT y Windows 2000. Para lograr la comunicación de la aplicación con los componentes MTS de forma remota se debe instalar *DCOM*.

Actualmente existen diferentes versiones de *DCOM* para satisfacer a los diferentes clientes.

Cliente	DCOM
Windows 95	DCOM95.exe
Windows 98	DCOM98.exe
Windows NT	integrado con Service Pack 3 o superior. ADO versión 2.0 o superior.
Windows 2000	ADO version 3.6

Al crear componentes ligados programáticamente con *MTS*, se vuelven capaces de trabajar integrados al 100% con ese software.

5. ACCESO A LA BASE DE DATOS DESDE LOS COMPONENTES MTS

Las aplicaciones de tres capas se componen de tres servicios.

- *Usuario*. Presentan la interfaz en las máquinas de los clientes.

- *Negocio*. Reciben la información del cliente, la validan y envían las peticiones a la Capa de Datos.

- *Datos*. Encargados de recibir la información de la Capa de Negocios, ejecutar las órdenes del cliente en forma de operaciones (Select, Insert, Update y Delete) con la Base de Datos y regresar las respuestas a las peticiones del cliente.

Debido a este esquema, cualquier operación realizada con la Base de Datos deberá estar encapsulada en los componentes de la Capa de Datos.

A continuación se presenta la forma de comunicarse con una Base de Datos SQL Server desde los componentes en el *MTS*.

5.1. UNIVERSAL DATA ACCESS (UDA)

En el desarrollo de Aplicaciones de Tres Capas, es necesario contar con una serie de herramientas que permitan acceder a cualquier tipo de datos, no importando se si encuentran en Base de Datos Relacionales o No Relacionales.

La plataforma de desarrollo para el acceso a los datos que ofrece Microsoft es llamada *UDA*, y se basa en componentes COM que interactúan entre si a través de interfaces OLE DB (Proveedor de Datos).

Así pues, cualquier lenguaje de programación que acepte COM podrá trabajar el acceso a los datos utilizando *UDA*.

La arquitectura de esta plataforma se muestra y explica a continuación.

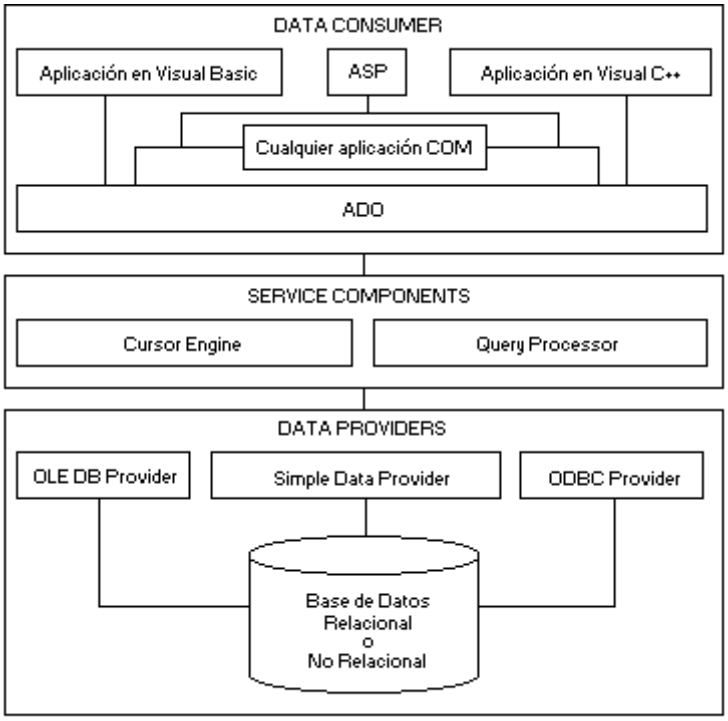


Fig. 5.1

La arquitectura de *UDA* se encuentra dividida en tres partes:

- *Consumidores de Datos (Data Consumers).* Son aquellos componentes que se encargan de manipular los datos para presentarlos en el cliente. Los clientes que pueden hacer uso de este servicio son aquellos que acepten la tecnología COM.

- *Componentes de Servicio (Service Components).* La función de estos componentes es la de producir los datos que el cliente está pidiendo, por ejemplo, la ejecución de una consulta.

- *Proveedores de Datos (Data Providers).* Son componentes que llevan a cabo la comunicación con cualquier tipo de Base de Datos, se podría decir que en la arquitectura *UDA*, representan a las fuentes de datos.

Está plataforma de desarrollo, se implementa bajo el nombre de Componentes Microsoft de Acceso a Datos (*MDAC Microsoft Data Access Components*). El *MDAC* contiene diferentes tecnologías, entre las más importantes se encuentran:

- *Objetos de Datos ActiveX (ActiveX Data Objects ADO).* Es un modelo de objetos que permite el acceso a cualquier tipo de Base de Datos a través del proveedor de *datos* OLE DB.

- *Proveedor de Datos OLE DB.* Es el sucesor de la Conectividad a Base de Datos (*ODBC Open Data Base Connectivity*), y es considerado como una especificación para un acceso estándar a las Bases de Datos. Al ser un acceso natural para las Bases de Datos que así lo permiten, el *OLE DB* mejora el tiempo de respuesta en comparación con el *ODBC*.

- *Conectividad a Base de Datos ODBC Versión 3.5.* Es una interfaz común para el acceso a datos en Bases de Datos basadas en el Lenguaje de Consultas Estructurado (*SQL Structured Query Language*).

- *Servicio de Datos Remotos (RDS Remote Data Service).* Es una tecnología utilizada para trasportar un grupo de registros del servidor al cliente. Cuando el grupo de registros se encuentra en el lado del cliente, es desconectado de la base de datos liberando

recursos en el Servidor y permitiendo que el cliente trabaje libremente con él.

5.2. ACTIVEX DATA OBJECTS (ADO)

Como ya se dijo en el punto 5.1, *ADO* es una tecnología que pertenece a *UDA* y es una interfaz compuesta de un modelo de objetos que sirve para accesar a cualquier tipo de Bases de Datos, sean relacionales o no.

El proveedor de datos utilizado por *ADO* para comunicarse con la Base de Datos es *OLE DB*. Si la Base de Datos ofrece un proveedor *OLE DB*, la comunicación será transparente ya que el lenguaje de los proveedores es nativo. Si no se ofrece el proveedor *OLE DB* por parte de la Base de Datos entonces *ADO* utilizará un proveedor *OLE DB* específico para la comunicación, por ejemplo, el *OLE DB* para *ODBC*.

El modelo de objetos de *ADO* en comparación con otros modelos para el acceso a Base de Datos (*DAO Data Access Objects y RDO Remote Data Objects*) es muy sencillo pues se compone de seis objetos (mostrados en la fig. 5.2), además, los objetos pueden ser manipulados de forma jerárquica o no.

La siguiente figura representa el modelo de objetos de *ADO*.

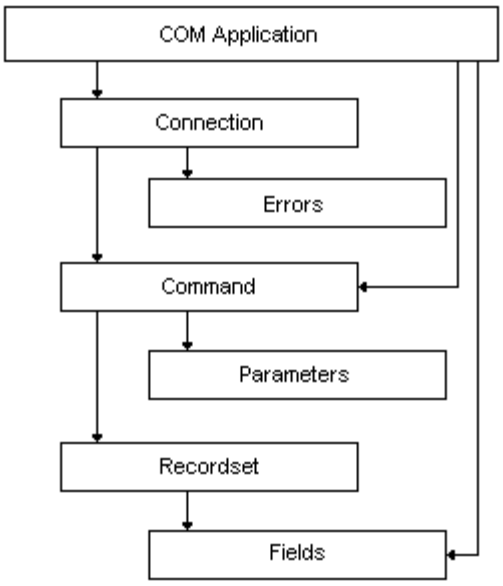


Fig. 5.2

Como se puede observar en la figura 5.2, cualquier aplicación que acepte la tecnología de COM puede utilizar el modelo de objetos de *ADO* para accesar a la información que necesita.

Existen tres objetos principales: Conexión (*Connection*), Comando (*Command*) y Recordset. Estos objetos pueden ser utilizados de forma jerárquica o de manera independiente. Por ejemplo, si se desea obtener un Recordset, se puede realizar una conexión a la base de datos y después generar un comando (instrucción de acción) que utilice esa conexión para recuperarlo, o solamente se podría utilizar el objeto Recordset y en sus propiedades asignar los datos de conexión y selección de información.

Gracias a esta facilidad, se puede utilizar una sola conexión para manipular diferentes comandos y recordsets. Opción que en modelos anteriores no se tenía pues por cada recordset se debía abrir una conexión propia para ese

objeto, es decir, si en la aplicación se utilizaban tres recordsets, cada uno debía tener una conexión propia.

Cada objeto principal tiene a su vez un objeto dependiente:

- Conexión (*Connection*) – Errores (*Errors*)
- Comando (*Command*) – Parametros (*Parameters*)
- Recordset – Campos (*Fields*)

Más adelante se analizarán cada uno de los objetos que componen a *ADO*.

Para poder manipular *ADO* en Visual Basic es necesario hacer referencia a la librería *msado15.dll (Microsoft ActiveX Data Objects 2.6 Library)* en la ventana de Referencias del proyecto de Visual Basic:

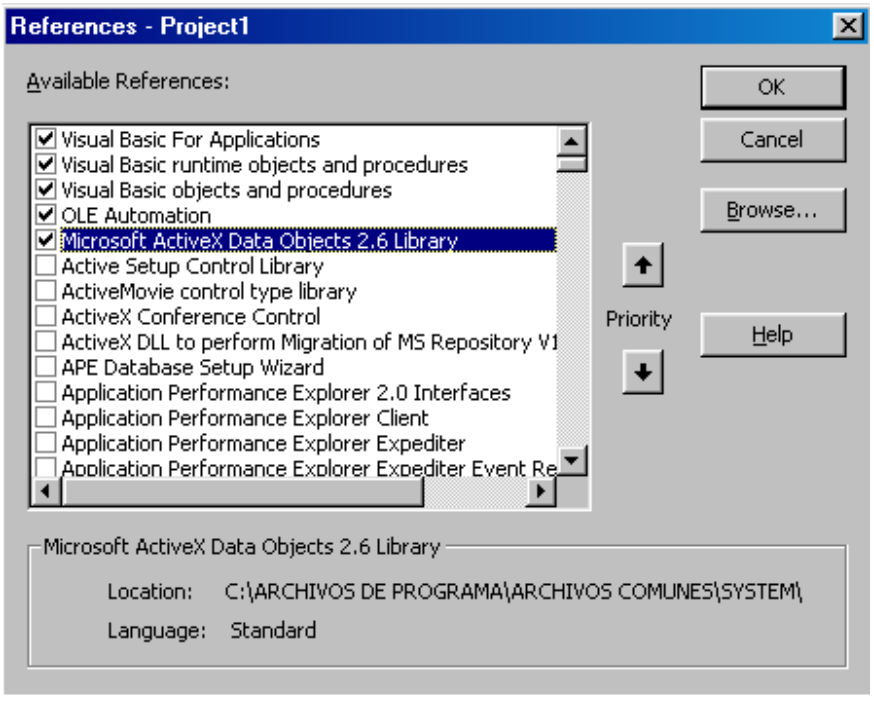


Fig. 5.3

5.2.1. OBJETO CONEXIÓN (CONNECTION)

Permite establecer la comunicación con la base de datos.

La conexión puede ser establecida al iniciar la aplicación para que todos los objetos de *ADO* utilicen la misma. Sin embargo, en las aplicaciones de tres capas, se recomienda abrir la conexión en el momento necesario y una vez realizada la acción que utiliza la conexión, ésta se deberá cerrar.

Los pasos a seguir para abrir una conexión son los siguientes:

1. Declarar una variable de tipo ADODB.Connection (Modelo de Objetos.Clase).
2. Instanciar la variable.
3. Especificar el proveedor de datos *OLE DB* que utilizará la conexión.
4. Dar la información necesaria para la conexión: usuario, password, nombre de la Base de Datos, servidor, etc. (la información varía dependiendo la Base de Datos a la cual se conectará *ADO*).
5. Abrir la conexión.

Abriendo una Conexión

```
'Declarando la Variable
   Dim cnBaseDatos As ADODB.Connection

'Instanciando la Variable
   Set cnBaseDatos = New ADODB.Recordset

   With cnBaseDatos

      'Especificando el Provide para SQL Server
      .Provider = "SQLOLEDB"

      'Otorgando datos de Conexión
```

```
        .ConnectionString = "Data Source = Servidor; Initial Catalogo = Base_Datos;
User Id = sa; Password = sapa"

        'Abriendo la conexion
        .Open

    End With
```

Una vez teniendo una conexión abierta, se puede utilizar para cualquier tipo de acción con la Base de Datos, desde una selección hasta cualquier inserción, actualización o eliminación de información.

Como ya se había mencionado, es recomendable cerrar la conexión una vez realizada la acción requerida para liberar los recursos del servidor. El método utilizado es *Close* del objeto *Connection*.

Cerrando una Conexión

```
'Cerrando la conexión
cnBaseDatos.Close

'Liberando los recursos
Set cnBaseDatos = Nothing
```

El método *Close*, libera los recursos de la Base de Datos y al asignar la variable a *Nothing* se permiten liberar los recursos del servidor donde se manejo la conexión.

Manejo de Errores

Al conectarse con una Base de Datos se pueden recibir errores inesperados en tiempo de ejecución. Es vital para la aplicación atrapar estos errores y los resolverlos inmediatamente antes de que se compliquen y vuelvan inestable al sistema.

Para solucionar los problemas que se presentan cuando se disparan errores en tiempo de ejecución, Visual Basic nos proporciona el objeto Error (*Err*) junto con la trampa de errores En Error (*On Error*). Sin embargo, cuando se tratan de errores en una conexión a Base de Datos a veces el objeto *Err* no nos proporciona la información adecuada o completa para poder solucionar el error de la mejor manera. La mejor forma de manejar estos errores es utilizando el objeto Errores (*Errors*) de *ADO*.

Objeto Errores (Errors)

El objeto *Connection* tiene un objeto asociado llamado *Errors* el cual funciona como una colección de objetos que contienen información detallada acerca del error producido en tiempo de ejecución relacionado con la conexión.

Entendiendo el tema, por cada error generado en la conexión se dispara uno o más objetos *Error* que se guardan dentro de la colección *Errors* conteniendo la información detallada del error atrapado. Para analizar el error, es necesario agregar al procedimiento de Visual Basic una trampa *On Error* y recorrer la colección *Errors* del Objeto *Connection*.

Cada objeto *Error* tiene las siguientes propiedades:

- Descripción (Description). Contiene la descripción del error.
- Número (*Number*). Especifica el número de error.
- Fuente (*Source*). Identifica el sitio u objeto en donde se produjo el error.
- Archivo de Ayuda (*HelpFile*) y Contexto de Ayuda (*HelpContext*). Indica el archivo y tópico de ayuda del error (sí existe).

- Estado SQL (*SQLState*) y Error Nativo (*NativeError*). Proporcionan información acerca del error nativo de la Base de Datos.

Analizando las propiedades de los objetos *Error* se pueden determinar de que tipo es y la manera de resolverlo.

Ejemplo:

```
Dim cnBaseDatos As ADODB.Connection

'Variable Error
Dim DispError As ADODB.Error

'Trampa de Errores
On Error GoTo Errores

Set cnBaseDatos = New ADODB.Recordset

With cnBaseDatos
   .Provider = "SQLOLEDB"
   .ConnectionString = "Data Source = Servidor; Initial Catalog = Base_Datos; User Id = sa;
Password = sapa"
   .Open
End With

'Realizar la acción con la Base de Datos

cnBaseDatos.Close
Set cnBaseDatos = Nothing
Exit Sub

Errores:

   For Each DispError In cnBaseDatos.Errors
      'Presento en el mensaje el Número, la Descripción y la Fuente del Error.
      MsgBox DispError.Number & vbCrLf & DispError.Description & vbCrLf &
DispError.Source
   Next
```

Cada vez que ocurre un error, la colección *Errors* se limpia y guarda los nuevos objetos *Error*.

5.2.2. OBJETO COMANDO (COMMAND)

Para cualquier acción en donde se involucre la manipulación de información de la Base de Datos se utiliza el objeto *Command*.

Este objeto permite:

- Consultar.
- Insertar.
- Actualizar.
- Eliminar.
- Llamar a consultas almacenadas de la Base de Datos.

Un comando debe ser ejecutado dentro de una conexión. *ADO* permite manipular sus objetos de forma jerárquica o independiente, es decir, si se desea ejecutar un comando entonces debe asociarse con una conexión ya creada o generarla implícitamente dentro del comando.

Un objeto conexión puede ser utilizado para cubrir todos los comandos necesarios en un sistema, pero esto no es muy recomendable, ya que entre ejecución de comando y comando puede haber un tiempo tal vez de 5 segundos a 1 hora, lógicamente la conexión se mantendría abierta durante ese tiempo utilizando recursos necesarios para atender otras peticiones de los clientes.

Al utilizar *ADO* en aplicaciones de Tres Capas, es necesario crear una conexión implícita dentro del comando a ejecutar para que ésta se genere y mantenga sólo el tiempo necesario para la ejecución del comando. Gracias a este método de conexión se asegura un aprovechamiento óptimo de los recursos del Servidor.

A continuación se muestra la manera de ejecutar un comando de forma jerárquica y de forma independiente:

Forma Jerárquica

```
'Declarando la variable
 Dim comComando As ADODB.Command

'Instanciando la Variable
Set comComando = New ADODB.Command

With comComando

   'Asignando la conexión abierta
   Set .ActiveConnection = cnBaseDatos

   'Selección del tipo de Comando
   .CommandType = adCmdText

   'Dando la orden a realizar
   .CommandText = "Update Clientes Set NombreCliente = 'Hanoi Hernández
   Villaseñor' Where IdCliente =      55021"

   'Ejecutando el Comando
   .Execute

End With

'Liberando recursos
Set comComando = Nothing
```

Forma Independiente (recomendada)

```
'Declarando la variable
 Dim comComando As ADODB.Command

'Instanciando la Variable
Set comComando = New ADODB.Command

With comComando

   'Asignando la conexión implícita
   .ActiveConnection = "Provider = SQLOLEDB; Data Source = Servidor; Initial Catalog =
   Base_Datos; User Id =   sa; Password =   sapa"

   'Selección del tipo de Comando
   .CommandType = adCmdText

   'Dando la orden a realizar
```

```
    .CommandText = "Update Clientes Set NombreCliente = 'Hanoi Hernández Villaseñor'
    Where IdCliente = 55021"

    'Ejecutando el Comando
    .Execute

End With

'Liberando recursos
Set comComando = Nothing
```

En los ejemplos anteriores se muestra la ejecución de un comando de actualización, de la misma forma se pueden utilizar comandos de inserción o eliminación en lenguaje *SQL*.

Para poder ejecutar una consulta almacenada (*Stored Procedure*) de la Base de Datos se le dice al objeto *Command* el tipo de comando a ejecutar y se da el nombre del Stored Procedure. Ejemplo:

```
'Selección del tipo de Comando
.CommandType = adCmdStoredProc

'Dando el nombre del Stored Procedure
.CommandText = "Actualiza_Precios"
```

Cuando un Stored Procedure no requiere parámetros para su ejecución, el método anterior es suficiente para llamarlo. Sin embargo, cuando se necesitan parámetros de entrada o salida se debe utilizar al objeto Parámetros (*Parameters*) del objeto *Command*.

Objeto Parámetros (Parameters)

Es un objeto colección que contiene de forma jerárquica el objeto *Command* de *ADO*.

Generalmente, los Stores Procedures necesitan parámetros para su ejecución. A través de ellos se pueden pasar valores de entrada que se

utilizan en el procedimiento y el procedimiento puede arrojar datos al sistema a través de parámetros de salida.

Existen tres tipos de parámetros:

- *Entrada*. Usados para pasar datos al procedimiento.
- *Salida*. Recuperan información enviada del procedimiento al sistema.
- *Entrada o Salida*. Pueden enviar o recibir información entre el procedimiento y el sistema.

Por cada parámetro requerido por el Stored Procedure se debe crear y agregar un Objeto Parameter a la colección Parameters.

Ejemplo:

Dentro de la Base de Datos se encuentra el Stored Procedure llamado Actualiza_Cliente, el cual contiene tres parámetros @Contraseña (tipo entrada), @Servicio (tipo entrada) y @Telefono (tipo salida).

El procedimiento actualiza el servicio del cliente siempre y cuando la contraseña del mismo sea válida y regresa el teléfono del cliente para confirmar su nuevo servicio.

```
CREATE PROCEDURE Actualiza_Cliente
        @Contraseña varchar (20),
        @Servicio varchar (10),
        @Telefono int OUTPUT

AS

        DECLARE @varCliente int

        SELECT @varCliente = idCliente FROM Clientes
                WHERE Contraseña = @Contraseña

        IF @varCliente IS NULL
```

```
        Return(0)
ELSE
        Begin
          UPDATE Clientes SET CliServicio = @Servicio
                WHERE Contraseña = @Contraseña
          SELECT @Telefono = CliTelefono FROM Clientes
                WHERE Contraseña = @Contraseña
          Return(1)
          End
```

La forma de llamar el Stored Procedure Actualiza_Cliente desde los componentes de datos es:

```
Dim comComando As ADODB.Command
'Declarando la variable tipo Parámetro
Dim prmStored As ADODB.Parameter

'Instanciando la Variable
Set comComando = New ADODB.Command

With comComando

   'Asignando la conexión implícita
   .ActiveConnection = "Provider = SQLOLEDB; Data Source = Servidor; Initial
   Catalog = Base_Datos; User Id =  sa; Password =  sapa"

   'Selección del tipo de Comando
   .CommandType = adCmdStoredProc

   'Dando el nombre del Stored Procedure
   .CommandText = "Actualiza_Cliente"

   'Creado los parámetros……..
   'Parámetro para recuperar los valores del Return()
   Set prmStored = .CreateParameter("Return", adInteger, adParamReturnValue, , 0)
   'Agregando a la Colección Parameters
   .Parameters.Append prmStored

   'Parámetro de entrada @Contraseña
   Set prmStored = .CreateParameter("@Contraseña", adVarChar, adParamInput, 20,
"hanoi_123hv_321")
   .Parameters.Append prmStored

   'Parámetro de entrada @Servicio
   Set prmStored = .CreateParameter("@Servicio", adVarChar, adParamInput, 10,
"TRONCAL")
   .Parameters.Append prmStored

   'Parámetro de salida @Telefono
```

```
    Set prmStored = .CreateParameter("@Telefono", adInteger, adParamOutput, , 0)
    .Parameters.Append prmStored

    'Ejecutando el Comando
    .Execute

End With
```

Una vez ejecutado el comando, se pueden leer los valores de los parámetros de salida de la siguiente forma:

```
Dim intTelefono As Integer

If comComando("Return") = 0 Then
    'No existe esa contraseña
    'Mandar mensaje de error
Else
    intTelefono = comComando("@Telefono")
    'Mandar el teléfono al cliente para confirmar el
    'nuevo servicio
End If
```

5.2.3. OBJETO GRUPO DE REGISTOS (RECORDSET)

Si se desea navegar a través de la información de la Base de Datos se utiliza al objeto *Recordset*. Este objeto permite el movimiento entre registros y la modificación de datos.

Al abrirse un *Recordset*, los datos que lo componen se cargan en memoria y permiten a los clientes la manipulación de los mismos. Cualquier modificación de datos hecha en el *Recordset* será almacenada inmediatamente en la Base de Datos.

ADO permite abrir un *Recordset* de diferentes formas, sin embargo, en este proyecto de investigación se presenta la forma de manipular *Recordsets* en aplicaciones de tres capas.

Tipos de Recordsets

Existen diferentes tipos de *Recordsets*, cada uno tiene características especiales para el tipo de aplicación que se desarrolle. Es necesario conocer todos los tipos para poder seleccionar el indicado cuando se construye una aplicación de tres capas, ya que si se utiliza un *Recordset* incorrecto se podrían consumir recursos vitales para la aplicación y perdería el rendimiento causando atraso en las transacciones de clientes y pérdidas monetarias.

Los tipos son:

- *Sólo Adelante (Forward Only)*. Permite recuperar registros, navegarlos y actualizarlos, sin embargo, sólo permite el movimiento en él hacia delante, de tal forma que únicamente se recorre una vez. Los cambios hechos por otros usuarios no son visibles.
- *Estático (Static)*. Hace una copia de los registros en la base de datos permitiendo navegar entre ellos cuantas veces sea necesario. No permite modificaciones a los datos. Los cambios hechos por otros usuarios no son visibles. Son de sólo lectura.
- *Grupo Llave (Keyset)*. Regresa las claves de los registros que componen al *Recordset*, a través de ella, los valores del registro actual son recuperados desde la Base de Datos permitiendo ver las actualizaciones y eliminaciones (las inserciones no son visibles pues las claves de estas no existen dentro del *Recordset*) de registros de otros usuarios. Permite inserción, actualización y eliminación de registros al igual que la navegación entre ellos. No se recomienda para aplicaciones de tres capas.
- *Dinámico (Dynamic)*. Permite inserción, eliminación, actualización de registros. Es el más funcional pero el que consume más

memoria y recursos de red, debido a que en cada movimiento sobre él, la sentencia Select se vuelve a ejecutar para permitir ver las inserciones, actualizaciones y eliminaciones de registros de otros usuarios. No se recomienda para aplicaciones de tres capas.

En las Aplicaciones de Tres Capas, es necesario utilizar *Recordsets* de lectura, es decir, el tipo *Static* o *Fordward Only* (con bloqueo de escritura). Cualquier *Recordset* que permita modificaciones no debe ser utilizado, ya que consume demasiada memoria en el servidor, recursos de red y además es muy posible que existan conflictos cuando se modifiquen los datos, ya que un mismo registro podría ser modificado al mismo tiempo por dos o más usuarios. Recordemos que con el objeto *Command* se pueden insertar, actualizar o borrar registros de la Base de Datos.

Para seleccionar el tipo de *Recordset* se utiliza la propiedad Tipo de Cursor (*CursorType*) asignado los valores *adOpenFordwardOnly*, *adOpenStatic*, *adOpenKeyset* o *adOpenDynamic*.

Donde Construir los Recordsets?

Cuando se utilizan un objeto *Recordset*, es recomendable mandar los datos a las estaciones de los clientes, para evitar una carga masiva de datos y acciones en el servidor.

Ventajas:

- Evitar cargar datos en memoria del servidor (no se consumen recursos innecesarios).
- Liberar recursos de red (ya que los datos existen en el cliente).

- Agilizar la aplicación del cliente ya que los datos son consultados localmente.

Si no se especifica el lugar donde debe ser creado un *Recordset*, éste se abrirá en el servidor.

Para determinar el lugar donde se abrirá el *Recordset*, se utiliza la propiedad Localización del Cursor (*CursorLocation*), los valores que acepta son *adUseClient* (en el cliente) o *adUseServer* (en el servidor).

Todos los *Recordsets* creados en el cliente serán de sólo lectura, si se desea un *Recordset* modificable en el cliente entonces debe ser desconectado de la Base de Datos. Más adelante se analizará el caso.

Bloqueos en el Recordset

Debido a que existen *Recodsets* que permiten modificaciones de datos, es necesario utilizar bloqueos para que no existan conflictos con otros usuarios. Por ejemplo:

Dos usuarios tienen el mismo *Recordset* creado en memoria y están utilizando el mismo registro. Este registro presenta los datos de un cliente bancario.

El Usuario 1 se encuentra modificando el campo teléfono a 55545352 y el Usuario 2 también está modificando el campo teléfono 56575859 al mismo tiempo. ¿Cuál teléfono deberá guardarse en el campo?

El ejemplo puede ser un poco absurdo, pero errores de este estilo ocurren frecuentemente.

El uso de bloqueos permite disparar errores para informar al usuario que hizo la modificación final que los datos están siendo actualizados por otro usuario o que han sido actualizados por otro usuario. De esta forma se brinda la oportunidad al usuario de decidir si actualiza nuevamente o no.

Es necesario recordar que no es recomendable crear *Recordsets* modificables en el servidor cuando se desarrollan aplicaciones de tres capas y que todos los *Recordsets* creados en los clientes son de sólo lectura.

Los tipos de Bloqueo son:

- *Sólo Lectura (Read Only)*. Sólo lectura, los datos no pueden ser modificados.
- *Pesimista (Pessimistic)*. El bloqueo se presenta desde que se modifican los datos, es decir, antes del método Actualizar (*Update*).
- *Optimista (Optimistic)*. El bloqueo se presenta después del método Actualizar (*Update*).
- *Optimista por Lotes (Batch Optimistic)*. Es utilizado en *Recordsets* desconectados.

Se debe tener mucho cuidado al utilizar bloqueos en *Recordsets*, ya que el tamaño de los bloqueos los decide el manejador de Base de Datos al que se está accesando.

Por ejemplo:

Access 97 y SQL Server 6.5 no permiten bloqueos a nivel registro, el bloqueo permitido es por página de memoria que es de 2k. Es decir, si un usuario modifica un registro se bloqueará la página de memoria

donde se encuentre este registro impidiendo así la modificación de cualquier otro registro que se encuentre en esta página.

Access 2000 permite el bloqueo a nivel Página y Registro.

SQL Server 7.0 y 2000 permiten bloqueos a nivel Página, Registro o Tabla.

Creación de un Recordset

Para crear un *Recordset* se necesita:

1. Declarar una variable de tipo ADODB.Recordset (Modelo de Objetos.Clase).
2. Instanciar la variable.
3. Especificar las propiedades.
4. Dar la sentencia de selección.
5. Asignar la Conexión
6. Abrir el *Recordset* utilizando el método Abrir (*Open*).

Ejemplo:

```
'Declarando la variable
 Dim recRecordset As ADODB.Recordset
 Dim strSelect as String

'Instanciando la Variable
Set recRecordset = New ADODB.Recordset

With recRecordset

   'Tipo de Recordset
   .CursorType = adOpenStatic

   'Tipo de Bloqueo
   .LockType = adLockReadOnly
```

```
    'Sitio del Recordset
    .CursorLocation = adUseClient

    'Creando la sentencia Select
     strSelect = "Select * From Clientes Where Estado = 'Aguascalientes'"

    'Asignando la conexión implícita
    .ActiveConnection = "Provider = SQLOLEDB; Data Source = Servidor; Initial Catalog =
    Base_Datos; User Id =   sa; Password =   sapa"

    'Abriendo el Recordset
    .Open strSelect
End With
```

Cerrando un Recordset

Si se desea cerrar un *Recordset* se utiliza al método Cerrar (*Close*) del objeto.

```
'Cerrando la conexión
recRecordset.Close

'Liberando los recursos
Set recRecordset = Nothing
```

Moviendose en el Recordset

Una vez creado el Recorset, se puede navegar a través de sus registros utilizando los métodos:

- *Mover (Move)*. Se mueve un número específico de registros. (Hacia adelante o atrás)
- *Mover al Primero (MoveFirst)*. Se posiciona en el primer registro.
- *Mover al Anterior (MovePrevious)*. Se mueve un registro atrás.
- *Mover al Siguiente (MoveNext)*. Se mueve un registro hacia adelante.
- *Mover al Último (MoveLast)*. Se posiciona en el último registro.

Recordsets Desconectados

Aunque es recomendable evitar la creación de *Recordsets* que puedan modificar la información de la Base de Datos, pueden existir casos en los que sea necesario hacer uso de estos tipos de *Recordsets*.

ADO permite generar *Recordsets* del lado del cliente y desconectarlos, para permitir que se modifique la información evitando conflictos con otros usuarios de la aplicación y manteniendo óptimos los recursos de red y servidor.

A continuación se presenta un ejemplo de cómo utilizar los *Recordsets Desconectados*.

Diferentes usuarios de la aplicación necesitan ver los datos de los clientes bancarios que están registrados en la Base de Datos para actualizar su información en caso de que ésta sea incorrecta o haya sufrido cambios.

Paso 1.

En la aplicación de usuario debe existir un método encargado de llamar la función de los clientes y recuperar el *Recordset* para publicarlo.

```
Private Sub Clientes()
   Dim objClase1 As Objeto.Clase1

   'Instanciando la Clase
   Set objClase1 = CreateObject("Objeto.Clase1")

   Set recClientes = New ADODB.Recordset

     'Llamando al método

   Set recClientes = objClase1.Clientes

   'Cualquier código
       Set objClase1 = Nothing
End Sub
```

Paso 2.

Se requiere de una función en la clase de Datos que recupere los clientes y los mande a la estación del usuario.

```
Public Function Clientes() As ADODB.Recordset
   Dim recRecordset As ADODB.Recordset
   Dim strSelect As String

   'Instanciando la Variable
   Set recRecordset = New ADODB.Recordset

   strSelect = "Select * From Clientes"
   With recRecordset

      .CursorType = adOpenStatic
      .LockType = adLockBatchOptimistic
      .CursorLocation = adUseClient
      .ActiveConnection = "Provider = SQLOLEDB; Data Source = Servidor; Initial Catalog
      = Base_Datos; User Id =   sa; Password =   sapa"
      .Open strSelect

      'Desconectando el Recordset
      Set .ActiveConnection = Nothing
   End With

   'Enviando el Recordset al Cliente
   Set Clientes = recRecordset
End Function
```

Paso 3.

Realizar en el la estación del usuario todas las modificaciones necesarias a los datos a través de los métodos Agregar Nuevo (*AddNew*), Actualizar (*Update*) y Eliminar (*Delete*) del objeto *Recordset*.

Paso 4.

Una vez que el usuario esté listo para enviar los datos a la Base, se necesita de un método en la aplicación que envíe el *Recordset* a la capa de Datos para almacenar la información.

```
Private Sub EnviarClientes()
   Dim objClase1 As Clase1

   'Instanciando la Clase
   Set objClase1 = CreateObject("Objeto.Clase1")

   'Pasando el Recordset con los cambios
   Call objClase1.AlmacenarClientes(recClientes)

   Set objClase1 = Nothing
End Sub
```

Paso 5.

En la clase de Datos debe existir una función que reciba el *Recordset* y almecene sus cambios en la Base de Datos.

```
Public Function AlmacenarClientes(ByVal recClientes As ADODB.Recordset)

   Dim con As ADODB.Connection
   Dim recRecordset As ADODB.Recordset
   Dim strSelect As String

   'Se abre una conexión
   Set con = New ADODB.Connection
   con.ConnectionString = "Provider = SQLOLEDB; Data Source = Servidor; Initial
   Catalog = Base_Datos; User Id =  sa; Password =  sapa"
   con.Open

   'Se asigna esa conexión al Recordset recibido
   Set recClientes.ActiveConnection = con

   'Se almacenan los cambios
   recClientes.UpdateBatch

   'Codigo de conflictos con registros modificados
   'por otro usuario


   'Cerrando la conexión
   con.Close
   Set con = Nothing

   'Liberando el Recordset
   Set reclientes = Nothing
End Function
```

Paso 6.

Código de Conflictos.

Dentro de la función de actualización debe existir un código que permita recuperar conflictos (si existen) de modificaciones hechas por otros usuarios. Estos conflictos pueden resolverse de una forma transparente al usuario.

```
Public Function AlmacenarClientes(ByVal recClientes As ADODB.Recordset)

    Dim con As ADODB.Connection
    Dim recRecordset As ADODB.Recordset
    Dim strSelect As String

    'Se abre una conexión
    Set con = New ADODB.Connection
    con.ConnectionString = "Provider = SQLOLEDB; Data Source = Servidor; Initial
    Catalog = Base_Datos; User Id =  sa; Password =  sapa"
    con.Open

    'Se asigna esa conexión al Recordset recibido
    Set recClientes.ActiveConnection = con

    'Se almacenan los cambios
    recClientes.UpdateBatch

    'Codigo de conflictos con registros modificados
    'por otro usuario

    'A traves del filtro podemos revisar si existen registros en conflicto
    recClientes.Filter = adFilterConflictingRecords

    'Verificamos si el Recordset filtrado contiene registros en conflicto
    If Not recClientes.BOF And Not recClientes.EOF Then
       'Le pedimos al recordset que recupere los nuevos valores
       recClientes.Resync adAffectGroup, adResyncUnderlyingValues

       'Escribimos los valores de nuestro recordset sobre los existentes
       recClientes.UpdateBatch
    End If

    'Cerramos la Transacción
    GetObjectContext.SetComplete

    'Cerrando la conexión
    con.Close
    Set con = Nothing
```

```
    'Liberando el Recordset
    Set reclientes = Nothing
End Function
```

Otra forma de solucionar estos problemas es filtrar el *Recordset* para saber si existen registros en conflicto y enviárselos al usuario para que tome la decisión de actualizar o no los datos.

Después de haber implementado completamente los servicios de negocios y datos dentro de los componentes DLL, es necesario asignar seguridad para evitar que clientes que no están autorizados realicen tareas que no corresponden a su puesto.

En el siguiente capítulo se muestra como asignar la seguridad en una aplicación de tres capas.

6. SEGURIDAD EN APLICACIONES DE TRES CAPAS

Cualquier tipo de aplicación (de dos o tres capas) que de servicio a diferentes usuarios necesita contar con una buena seguridad para evitar ataques maliciosos y destrucción de datos valiosos de la empresa.

Existen dos servicios que no deben dejar de existir en estas aplicaciones: *Autenticación* y *Autorización*. Estos servicios garantizan que el acceso a los datos lo tendrán únicamente aquellos usuarios dados de alta en el sistema.

- *Autenticación*. Validar a un usuario a nivel credenciales digitales para verificar si el usuario es quien dice ser.

- *Autorización*. Asignar a los distintos usuarios a diferentes niveles de seguridad. Así cada nivel tendrá el acceso a los recursos indicados impidiendo que obtengan datos para propósitos distintos a sus funciones.

En la arquitectura tres capas, las seguridad se lleva a cabo a través del MTS y la Base de Datos.

El MTS se hace cargo de la Autenticación de usuarios y la Autorización a los diferentes recursos del sistema y la Base de Datos sólo autoriza el acceso y manipulación de los diferentes objetos alojados en ella.

6.1. SEGURIDAD EN MTS

La seguridad en el MTS se lleva a cabo a través de *Roles*.

Un *Role*, en un grupo de usuarios (cuentas de usuario en Windows NT) que tiene acceso a un paquete.

Antes de continuar, se debe recordar que en el MTS, la arquitectura de una aplicación se basa en paquetes y componentes.

Cada paquete puede tener uno o más *Roles*, cuando el paquete tiene habilitada la seguridad, sólo los usuarios que se encuentren ligados a estos *Roles*, podrán levantar instancias de los componentes dentro del paquete.

Cada *Role* define a los usuarios que pueden llamar a las interfaces en un componente. Cuando se llaman los métodos entre componentes dentro del mismo paquete no se revisan las cuentas nuevamente, ya que dentro de un paquete, todos los componentes se encuentran corriendo en ambientes de confianza.

Cada componente en el paquete también puede habilitar su seguridad, de tal forma que aún con los *Roles* ligados al paquete, si éstos no se encuentran asignados al componente no se podrán levantar instancias de esa clase.

Gracias a que los paquetes y los componentes tienen su propia seguridad, se puede ofrecer en este tipo de aplicaciones distintos niveles de seguridad para diferentes grupos de usuarios ligados a un paquete.

Por ejemplo:

En una aplicación en donde existen dos grupos de usuarios, capturistas y gerentes, hay un paquete con dos componentes, uno de ellos se encarga de ofrecer los servicios a los capturistas para que envíen la información de la pantalla de captura a la base de datos (Clase1) y el otro se encarga de presentar reportes a los gerentes de la empresa (Clase2).

Bien, se tienen dos niveles de usuarios: gerentes que pueden capturar información y además observar reportes que les permitan conocer los alcances y avances de su empresa en el mercado y capturistas que sólo envían la información a la base de datos.

Es por esto que deben crearse dos Roles (Role1 y Role2) ligados al mismo paquete.

El grupo de Gerentes estará ligado al Role1 y el de Capturistas estará ligado al Role2.

Si se habilita la seguridad en el paquete, cualquier grupo de usuarios puede levantar instancias de ambas clases.

Para evitar este problema, el Role1 lo ligamos a la Clase1 y a la Clase2 y el Role2 sólo lo ligamos a la Clase1; una vez realizado el proceso, se habilita la seguridad para cada componente.

Este procedimiento permite que los capturistas levanten instancias sólo de la Clase1 y los gerentes levanten instancias de la Clase1 y de la Clase2.

Al asignar los usuarios a los *Roles* en el paquete queda cubierta la autenticación, y al ligarlos a cada uno de los componentes se otorgan permisos para así cumplir con la autorización en la seguridad.

Cuentas de Usuarios en Windows NT

Para asignar seguridad a las aplicaciones en tres capas, primero se crean las cuentas en Windows NT de cada uno de los usuarios que tendrán acceso a la aplicación, si la aplicación tiene diferentes niveles de usuarios, es recomendable crear grupos para los distintos niveles y ligar a cada uno de los usuarios a su grupo o grupos correspondientes.

La creación de grupos y usuarios en Windows NT se lleva a cabo a través del Administrador de Dominios de Windows (*Windows Manager for Domains*).

Generalmente, este proceso es realizado por los administradores de la empresa. Ellos se encargarán de dar de alta la red y asignar a cada uno de los usuarios sus cuentas bajo el o los dominios en la red.

Creación de Roles en el MTS

Una vez creado el paquete con sus componentes, es necesario agregar seguridad. Para ello se generan *Roles* en los paquetes.

Hay que seguir estos sencillos pasos para crear un *Role* y asignarlo a un paquete:

1. Dentro del MTS, seleccionar el paquete en el que se desee agregar el *Role*.
2. Expandir el paquete y seleccionar la carpeta *Roles*.

3. Dar un clic con botón derecho y seleccionar Nuevo Role (*New Role*).
4. Proporcionar el nombre del *Role* y dar un click en Aceptar.

Una vez creado el *Role* en el paquete, si se desea agregarlo a un componente se debe de:

1. Expandir el componente en el que se desea agregar el *Role*.
2. Seleccionar la carpeta Socios del Role (*Role Membership*).
3. Dar un click con botón derecho y seleccionar Nuevo Role (*New Role*).
4. Una caja de diálogo aparecerá con todos los *Roles* creados en el paquete, seleccionar de éstos el o los que se deseen y dar un click en Aceptar.

Ligando Usuarios a los Roles

Una vez creados los *Roles* y ligados a los componentes, se asignan los usuarios indicados para otorgarles permisos de creación y manejo de componentes.

La manera de asignar usuarios a un *Role* en el *MTS* es muy fácil, lo único que se debe hacer es:

1. Expandir el *Role* en donde se ligarán a los usuarios.
2. Selecionar la carpeta Usuarios (*Users*).
3. Con botón derecho, dar un click y seleccionar Nuevos Usuarios (*New Users*).
4. En la caja de dialogo Agregar Usuarios y Grupos al Role (*Add Users and Groups to Role*), seleccionar al grupo de usuarios o a los usuarios indicados y pulsar Aceptar.

Una vez realizada esta operación, se debe dar tirar al proceso del paquete en el *MTS* antes de poder ver reflejados los cambios.

Habilitando la Seguridad en los Paquetes y Componentes

Después de haber creado los *Roles* y haber ligado los usuarios a los mismos se debe habilitar la seguridad para la aplicación a nivel de paquete o componente.

Los pasos para dar de alta la seguridad son los siguientes:

1. En el *MTS*, seleccionar el paquete o componente y dar un click con el botón derecho.
2. En el menú, seleccionar Propiedades (*Properties*).
3. Sobre el cuadro de dialogo, dar un click en la pestaña Seguridad (*Security*).
4. Seleccionar Habilitar la Autorización (*Enable authorization checking*).

Existen diferentes combinaciones al habilitar la seguridad en paquetes y componentes, es por eso que se recomienda tomar en cuenta la siguiente tabla antes de asignar la seguridad.

Seguridad en Paquete	Seguridad en Componente	Resultado
Habilitada	Habilitada	Seguridad a nivel paquete y componente
Habilitada	Deshabilitada	Seguridad a nivel paquete
Deshabilitada	Habilitada	No hay seguridad
Deshabilitada	Deshabilitada	No hay seguridad

6.2. SEGURIDAD EN BASE DE DATOS

La seguridad en SQL Server se encuentra basada en cuentas de usuario, las cuales contienen un nombre de usuario y una contraseña. Estas cuentas sirven para validar y para asignar permisos a los diferentes usuarios que manipulan los objetos en la Base de Datos.

Antes de asignar los usuarios a la Base de Datos es necesario conocer que SQL Server tiene tres tipos de seguridad: Estándar, Integrada y Mixta.

Seguridad Estándar

Cada usuario necesita una cuenta en Windows NT para validarse en el dominio y tener acceso a la red, además de esta cuenta, ellos necesitan una cuenta personalizada en SQL Server, de tal forma, que cada vez que vayan a conectarse a la Base de Datos necesitan pasar la información de su cuenta para que los valide SQL Server.

La propiedad Cadena de Conexión (*ConnectionString*) del objeto *Connection* de *ADO* debe tener la información de la cuenta de usuario que desea conectarse. Ejemplo:

```
'Otorgando datos de Conexión
con.ConnectionString = "Provider = SQLOLEDB; Data Source = Servidor; Initial Catalogo
= Base_Datos; “ & _
“User Id = sa; Password = sapa"
```

Seguridad Integrada

En este tipo de seguridad, cada usuario deberá tener una cuenta asignada en Windows NT para poder entrar a la red, esa cuenta deberá estar dada de alta en SQL Server, cuando el usuario necesite una conexión a la Base de Datos ya no es necesario enviar la información de la cuenta en la Base

de Datos. Debido a que sus servicios se encuentran integrados con la plataforma Windows NT sólo basta con acceder al dominio en donde se encuentra la Base de Datos para poder manipular sus objetos y administrar su información.

La propiedad Cadena de Conexión (*ConnectionString*) del objeto *Connection* de *ADO* debe tener la información de conexión bajo ambientes de confianza. Ejemplo:

```
'Otorgando datos de Conexión
con.ConnectionString = "Provider = SQLOLEDB; Data Source = Servidor; Initial Catalogo
= Base_Datos; “ & _
“Integrated Security=SSPI"
```

Este tipo de seguridad brinda un mejor rendimiento a este tipo de aplicaciones debido a que la conexión a la Base de Datos evita validar la información del usuario.

Seguridad Mixta

La seguridad mixta en SQL Server envuelve a la seguridad Estándar e Integrada para trabajar aplicaciones en ambientes de confianza o desconfianza.

Por ejemplo, tal vez se tiene una Base de Datos que le da servicio a una aplicación de escritorio y al mismo tiempo a una aplicación en Internet. Es claro que los usuarios en Internet no pueden trabajar bajo un ambiente de confianza, por lo tanto, las conexiones a la Base de Datos se tendrían bajo una seguridad Estándar, en cambio, los usuarios de la aplicación de escritorio se pueden trabajar con seguridad Integrada por la administración en confianza dentro de la empresa.

Seguridad en SQL Server para Aplicaciones Tres Capas

Como ya se ha dicho, las aplicaciones de tres capas de componen de tres servicios, los de Usuario, Negocios y Datos. Donde, los servicios de usuario estarán implementados como el cascarón de la aplicación del lado de los usuarios, mientras que los del Negocio y Datos estarán dentro del servidor.

Los componentes de Datos son los que realizarán conexiones a la Base de Datos, como éstos se encuentran almacenados del lado del servidor se pueden trabajar bajo ambientes de confianza al crear conexiones con la Base de Datos, gracias a esta configuración se puede obtener mayor rendimiento para la aplicación.

Para poder manejar conexiones a la Base de Datos bajo una seguridad integrada debemos crear las cuentas en Windows NT y además asignar estas cuentas a SQL Server.

La creación de cuentas en SQL Server es relativamente fácil, sólo hay que seguir estos pasos:

1. Abrir el Administrador de SQL Server (*SQL Server Enterprise Manager*).
2. Navegar sobre el árbol de servidor y seleccionar la carpeta Seguridad (*Security*).
3. Seleccionar Cuentas (*Login*) y dar un click derecho.
4. Pulsar Nueva Cuenta (*New Login*).
5. En el cuadro de diálogo, seleccionar la cuenta de Windows NT que desea agregar.
6. Asignar la opción Autenticación (Authentication) a Autenticación Windows y seleccionar el dominio.

7. El la opción Default, seleccione la Base de Datos a la que agregará la cuenta.
8. Pulse Aceptar.

Una vez creadas las cuentas en SQL Server, hay que configurar la herramienta para que acepte la seguridad integrada de la siguiente manera:

1. Seleccionar el Servidor dentro del Administrador de SQL Server (*SQL Server Enterprise Manager*)
2. Dar un click derecho y pulsar Propiedades (*Properties*).
3. Del cuadro de diálogo, escoger la pestaña Seguridad (*Security*).
4. En la opción Seguridad (*Security*), se debe seleccionar Seguridad Integrada (*Integrated Security*) o Seguridad Windows (*Windows Security*).
5. Dar un click en Aceptar.

6.3. CONNECTION POOLING

Ya se ha dicho que las aplicaciones de tres capas pueden soportar miles de usuarios conectados al mismo tiempo. Bien, ahora imaginando que la aplicación tiene en este momento 1000 usuarios conectados al mismo tiempo y cada uno de ellos está accesando a la Base de Datos, en este caso, existen 1000 conexiones establecidas al mismo tiempo. Cuando dichas conexiones sean liberadas, como cada una tiene propiedades de conexión diferentes, no pueden ser reutilizadas y trae como resultado una rendimiento bajo en la aplicación así como la pérdida de escalabilidad.

El recurso Cola de Conexiones (*Connection Pooling*) ayuda a mejorar el rendimiento de la aplicación además de asegurar la escalabilidad de la aplicación en cuanto a conexiones a Base de Datos se refiere.

Este permite reutilizar conexiones a la Base de Datos cuando sea necesario siempre y cuando se encuentren liberadas y las propiedades de conexión sean iguales.

Para hacer posible este caso se deben crear identidades a los paquetes y hacer conexiones bajo seguridad integrada, de tal forma que cada usuario que este tratando de accesar a la Base de Datos a través de un componente en un paquete con identidad tome las propiedades de conexión (Nombre de Usuario y Contraseña) del paquete. En el momento en que un usuario libere la conexión, ésta será almacenada en la Cola de Conexiones (*Connection* Pooling) hasta que otro usuario la reutilice o se cumpla el tiempo de conexión (*TimeOut* que es de 60 seg). De esta forma, los objetos conexión se asignan sin tener que volver a crearse y así se agiliza una operación con la Base de Datos.

La configuración del *TimeOut* del *Connection Pooling* se hace sobre el Registro (*Registry*) de Windows en la siguiente ruta:

> \HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBCINST.INI\SQL SERVER\CPTimeout=60

Este será el tiempo máximo que permanezca una conexión sobre el *Connection Pooling*. Si no es reutilizada, será destruida al término del tiempo asignado.

Identidad de los Paquetes

Los paquetes que alojan a los componentes permiten que se les asigne una cuenta configurada en Windows NT para permitir que todos y cada uno de los usuarios que accesen a una Base de Datos a través de los

componentes en el paquete tomen la identidad asignada y puedan reutilizar la conexión.

Otro beneficio de la *Identidad*, es que si existe una aplicación que le da servicio a 5000 usuarios y ésta se encuentra dividida en 10 paquetes, en la Base de Datos sólo deben existir las cuentas de los paquetes y no las de los usuarios, así se minimizan costos y se facilita la administración de las cuentas.

Para asignar la *Identidad* a un paquete se debe hacer lo siguiente:

1. Crear una cuenta de usuario en Windows NT.
2. Dentro del *MTS*, seleccionar el paquete y dar un click con botón derecho y pulsar Propiedades (*Properties*).
3. Dentro de la pestaña Identidad (*Identity*), seleccionar la opción Este Usuario (*This User*).
4. Proporcionar la cuenta de usuario y contraseña creados para el paquete.
5. Dar un clic en Aceptar.

Una vez realizado este paso para todos los paquetes, se tienen que dar de alta las cuentas de usuario en SQL Server y otorgarles los permisos necesarios.

La seguridad es uno de los factores más importantes en una aplicación, no importa que sea de tres capas o no. Es recomendable que desde el diseño lógico de la misma sea analizada la seguridad y durante la creación de la aplicación deberá implementarse, ya que si ésta es tomada en cuenta al termino del desarrollo de la aplicación su implementación podría causar modificaciones al sistema y costos mayores de mantenimiento.

7. HERRAMIENTAS ADICIONALES

De acuerdo a lo que se ha analizado y estudiado a lo largo de este trabajo ya se esta preparado para comenzar a crear aplicaciones de 3 capas. Sin embargo, existen algunas herramientas adicionales que se podrían integrar a la solución para hacerla más confiable y robusta. Algunas herramientas son: el Servidor Clúster de Microsoft (*Microsoft Cluster Server*), la Cola de Mensajes de Microsoft (*Microsoft Message Queue*) y el Servidor SNA de Windows (*Windows SNA Server 4.0*), entre otras.

7.1. SERVIDOR CLUSTER DE MICROSOFT (MICROSOFT CLUSTER SERVER)

Desde que se crearon las aplicaciones que cubren la producción de empresas se ha buscado la tolerancia a fallos, debido a que un error en el servidor puede llegar a causar pérdidas millonarias.

El resultado generado en la búsqueda de soluciones que permitan que una aplicación o sistema sea confiable (término descrito como el tiempo en levantar una aplicación después de haber ocurrido un error, entre más rápido, mayor confiabilidad) ha sido una herramienta conocida como *Microsoft Cluster Server*.

El software permite que una aplicación siempre esté disponible aunque falle el sistema.

El Clúster trabaja de dos formas diferentes:

1. Existen dos servidores que comparten un arreglo de discos, el servidor principal es el encargado de ofrecer el servicio de la aplicación a los clientes, si éste falla, el servidor secundario entra

en acción impidiendo una error total en la aplicación y manteniéndola disponible a los clientes en un tiempo de recuperación relativamente corto.

2. Un arreglo de discos es compartido por 2 ó más servidores (llamados nodos), el nodo principal se encarga de dar el servicio de la aplicación a los clientes, cuando éste es saturado por las peticiones de los clientes, otro nodo entrará en su ayuda para lograr un balance de carga, si el nuevo nodo es saturado al igual que el principal, otro nodo más se habilitará para permitir el servicio. Conforme se vayan liberando las tareas y los servidores dejen de estar saturados uno a uno se irán deshabilitando hasta que solamente quede el nodo principal.

Gracias a este software podemos ofrecer un respaldo completo a los clientes de la aplicación y a los dueños de las mismas evitando posibles errores que se vean reflejados en sus ganancias.

7.2. COLA DE MENSAJES DE MICROSOFT (MSMQ MICROSOFT MESSAGE QUEUE)

Así como se tiene un correo electrónico y comunicación a través de él con otras personas, las aplicaciones también pueden mandar mensajes a través de ellas para informar de algún suceso, sin necesidad de un software como el Microsoft Exchange (software para el intercambio de mensajes electrónicos).

MSMQ, permite mantener a dos o más aplicaciones comunicadas entre sí a través de mensajes en escenarios en donde no se requiera una respuesta inmediata y donde la red sea insuficiente para la transmisión síncrona de la información.

Tomando en cuenta el siguiente escenario, se mostrará la funcionalidad del MSMQ.

Una agencia de viajes permite hacer reservaciones dos semanas antes para sus vuelos, en ellos ofrece entre otras cosas un menú que los clientes pueden seleccionar para hacer más confortable su viaje. De acuerdo a esto, la agencia de viajes se encargará de reservar el boleto de viaje para el cliente, pero debe enviar la información del menú al proveedor de alimentos. El proveedor de alimentos se encarga de preparar los menús y enviarlos al vuelo correspondiente sin confirmación alguna el mismo día en que los clientes viajan.

En este escenario se pueden realizar 2 aplicaciones, una cubrirá a la agencia de viajes y se encargará de hacer las reservaciones de los vuelos y mandar los mensajes de los menús y la otra será la que tenga el proveedor de alimentos para leer los mensajes y saber que menús debe preparar y cuando los debe entregar.

Como en este escenario, el *MSMQ* puede facilitar la tarea de comunicación con otras aplicaciones aún cuando éstas no estén disponibles, ya que los mensajes son almacenados en una Base de Datos asegurando que éstos no se eliminen hasta que hayan sido entregados satisfactoriamente.

7.3. SERVIDOR SNA DE WINDOWS (MICROSOFT SNA SERVER 4.0)

Aunque el almacenamiento de información actualmente sea en Bases de Datos como SQL Server, Oracle, Informix, Sybase, etc., se debe recordar que no todos los posibles clientes se encuentran listos para migrar a estas tecnologías por varios factores (costos, infraestructura o simplemente porque su almacenamiento de información cubre perfectamente sus necesidades).

Si se desea que haya comunicación con información que se encuentra almacenada en Mainframes, se puede utilizar el *Microsoft SNA Server 4.0*, éste nos brinda comunicación con CICS, IMS, AS/400, VSAM y DB2.

El beneficio que ofrece este software es que no se debe hacer que un cliente cambie por completo su tecnología para implementar soluciones de tres capas en su empresa, así reducimos el Costo Total de Propietario.

El uso de tecnologías como las que aquí se presentan, mejoran el rendimiento, disponibilidad, confiabilidad, escalabilidad y minimizan costos en la creación, mantenimiento y distribución de una aplicación de tres capas.

Es recomendable analizar lógicamente el ambiente e infraestructura así como también la solución antes de la creación e implementación física de la misma debido a que es posible que ciertos clientes necesiten implementar tecnologías adicionales y otros no.

CONCLUSIONES

Se debe entender que actualmente existe una relación muy estrecha entre el mundo del negocio y el mundo de la computación, y esto lo podemos resumir en dos conceptos: reglas del negocio y requerimientos del negocio. Las reglas del negocio son todas las políticas y restricciones a través de las cuales se rige una empresa para llevar a cabo su meta de forma competitiva. Los requerimientos del negocio es todo aquello que necesita la empresa de acuerdo al mercado y estos requerimientos se convierten prácticamente en una guía para el programador.

Al estar relacionados estos conceptos, se pueden resumir seis puntos básicos para lograr la expectativa de la empresa en el mercado, éstos son:

- Obtener la información en cualquier momento.
- Generar mejores decisiones que ayuden a crecer al negocio.
- Tener una respuesta rápida al cambio en el mercado.
- Lograr una mejor comunicación entre trabajadores y clientes.
- Impulsar la innovación entre los empleados.
- Generar una mejor retroalimentación y así estar al tanto de las necesidades del cliente.

De acuerdo a la visión del programador, para generar una aplicación que cubra las necesidades de la empresa se debe conocer el negocio para el cual la aplicación será creada, así como también consolidar la información y manejar prioridades sobre ésta. Después de obtener este conocimiento, se realizarán entrevistas con los diferentes usuarios responsables de cada área involucrada en el proyecto para así levantar la documentación que permitirá cubrir cada una de las necesidades en el alcance del sistema. Una vez cubiertos estos puntos, entonces se comienza la fase de diseño y se

elabora un plan de trabajo en donde se especificarán objetivos a corto y largo plazo de acuerdo a las prioridades establecidas en cada una de las áreas.

Es necesario entender que existen diferentes escenarios de programación, como se muestra en esta tesis, y cada uno de esos escenarios tiene ventajas y desventajas, por lo cual, de acuerdo al necesidades del cliente que deberán evaluarse en la fase de diseño se podrá elegir la arquitectura que será utilizada (de dos o de tres capas).

De acuerdo a lo ya expuesto, la presente tesis no exige modificar la forma de trabajar y desarrollar los diferentes sistemas de computación, simplemente procura invitar a los programadores a conocer una nueva forma de generar aplicaciones que además de cubrir perfectamente los requerimientos de la empresa, permita optimizar el tiempo de desarrollo, facilite el mantenimiento del mismo y reduzca los costos involucrados en la creación, vida y actualización del producto final.

BIBLIOGRAFIA

Libros

Platt David S.; **Understanding COM+**; 1ra Edición; Editorial Microsoft Press.

Soukup Ron, Delaney Kalen; **Inside Microsoft SQL Server 7.0**; 1ra Edición; Editorial Microsoft Press.

Thai Thuan L.; **Learning DCOM**; 1ra Edición; Editorial Osborne.

Maloney Jim; **Distributed COM (Application Development Using Visual Basic 6.0)**; 1ra. Edición; Editorial Prentice Hall.

Lomas Paul; **VB & VBA in a Nutshell**; 1ra. Edición; Editorial O'Reilly.

Manuales de Entrenamiento

Microsoft Corporation; **Mastering Distributed Application Design and Development Using Microsoft Visual Studio 6**; USA; 1998.

Microsoft Corporation; **Mastering Microsoft Visual Basic 6 Development**; USA; 1998.

Microsoft Corporation; **Mastering Enterprise Development Using Microsoft Visual Basic 6**; USA; 1998.

Microsoft Corporation; **Implementing a Database on Microsoft SQL Server 7.0**; USA; 1998.

Revistas

Cabrera Ulises; **Windows DNA**; Windows 2000 Magazine America Latina; Año 3-Vol. 1; Editores Asociados de América, S.A. de C.V.