UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE CIENCIAS

INTEGRANDO APLICACIONES EN DIFERENTES PLATAFORMAS

T E S I S

QUE PARA OBTENER EL TITULO DE

A C T U A R I O

P R E S E N T A :

JESUS SUAREZ GUERRERO

DIRECTOR. DE TESIS: M. en C. ELISA VISO GUROVICH

MEXICO, D. F. 1999

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

Facultad de Ciencias

INTEGRANDO APLICACIONES EN DIFERENTES PLATAFORMAS

T E S I S

QUE PARA OBTENER EL TITULO DE:

A C T U A R I O

P R E S E N T A:

JESUS SUAREZ GUERRERO

Director de Tesis: M. en C. Elisa Viso Gurovich

MEXICO, D.F. 1999

A mis padres Francisco y Mercedes, que con su ejemplo me dieron las bases para ser libre, y quienes debo lo que soy.

A mi esposa Rosy, compañera y apoyo incondicional en los momentos difíciles de mi vida

A mis hijos Adrian, Ana Karina y Ariel, por quienes me esfuerzo cada día y para quienes deseo que esta sea un ejemplo a superar

A mis hermanos Francisco, Gerardo, Ruben, Raúl, Mario y Héctor, quienes siempre me impulsaron a seguir adelante

A Mario y Elisa Magidin, que gracias a su apoyo hicieron posible la culminación de este trabajo.

VNIVERSIDAD NACIONAL
AVFNOMA DE
MEXICO

MAT. MARGARITA ELVIRA CHÁVEZ CANO
Jefa de la División de Estudios Profesionales de la
Facultad de Ciencias
P r e s e n t e

Comunicamos a usted que hemos revisado el trabajo de Tesis:

Integrando aplicaciones en diferentes plataformas

realizado por Jesús Suárez Guerrero

con número de cuenta 6408342 , pasante de la carrera de Actuario

Dicho trabajo cuenta con nuestro voto aprobatorio.

A t e n t a m e n t e

Director de tesis Propietario		M. en C. Elisa Viso Gurovich
Propietario		Dr. Mario Magidin Matluk
Propietario		M. en C. Javier García García
Suplente		M. en C. Virginia Abrín Batule
Suplente		Act. Mauricio Aguilar González

Consejo Departamental de Matemáticas.

M.en A.P. Ma. del Pilar Alonso Reyes

Indice

Introducción

Este documento está basado en el trabajo desarrollado en el Banco del Atlántico para incorporar aplicaciones con interfase gráfica bajo el esquema cliente-servidor en plataformas abiertas con aplicaciones tradicionales en plataformas cerradas escritas en Cobol, tal como ocurre en una gran cantidad de empresas e instituciones hoy en día.

Podríamos pensar que ya no existen aplicaciones en Cobol, en las escuelas ya no se imparte, se considera obsoleto, fuera de época. Nada más alejado de la realidad, las empresas e instituciones que primero empezaron a automatizarse, lo hicieron en Cobol, y a sus sistemas y aplicaciones originales, se le fueron incorporando a través del tiempo nuevas funciones para apoyar la administración y operación en todas las áreas.

No obstante el gran avance que día a día se presenta en las áreas de sistemas, y que contamos con nuevas y poderosas herramientas, existen muchos sistemas desarrollados en la "antigüedad" y que sin embargo continúan operando ya que el costo y el tiempo requeridos para sustituirlos es demasiado alto.

En general, las nuevas aplicaciones se están desarrollando con herramientas modernas, pero continúa el uso y mantenimiento de los sistemas tradicionales, lo que obliga a incorporar tecnología que permita compartir la información y una migración paulatina a nuevos y modernos esquemas de automatización.

Los usuarios, clientes y empleados, demandan aplicaciones más fáciles de usar como son las construidas con interfases gráficas (GUI Grafical User Interface) éste es el caso de las aplicaciones bajo ambiente Windows.

Sin embargo, muchas empresas no han tocado sus sistemas tradicionales excepto para el mantenimiento rutinario que garantice su funcionamiento, ya que están más ocupados en el desarrollo de nuevos esquemas de automatización para las áreas que están fuera de ellos y que requieren mayor eficiencia y productividad. ¡ Pero se acerca el año 2000 ! Los sistemas tradicionales en su gran mayoría, por no decir todos, no consideran el año de 4 dígitos o los algoritmos que permitan operar con dos dígitos los últimos años del siglo XX y los primeros del XXI, ni en los programas, ni en los datos, y se tiene la urgente necesidad de ajustarlos para que puedan seguir operando a partir del próximo siglo.

Lo anterior, además de ser una necesidad impostergable, representa una oportunidad para crear la infraestructura necesaria que permita adoptar con facilidad las herramientas que la tecnología hoy en día nos ofrece y lo que es más importante, permitir la convivencia armónica de aplicaciones y plataformas actuales y su migración futura a esquemas nuevos, sin repercusiones indeseables.

Es aquí, precisamente, donde entra nuestro trabajo. El Banco del Atlántico tiene sus sistemas principales en Cobol, en archivos tradicionales y en equipo central (Main Frame) cerrado y se pretende desarrollar nuevas aplicaciones en ambiente gráfico, en otras plataformas, con base de datos relacional y que permitan accesar a los sistemas en operación independientemente de donde se encuentren (lugar y plataforma) y en que estén desarrollados. Esto, indudablemente no es una tarea fácil, pero afortunadamente ya existe en el mercado la tecnología que lo permite, el monitor de transacciones.

Existen varios tipos de monitores de transacciones, como Encino y Tuxedo entre otros. El Banco optó por Tuxedo, por lo que será al que nos refiramos en lo sucesivo.

Análisis del problema

Ante la competencia cada vez más fuerte y agresiva del mercado bancario en nuestro país, es necesario ofrecer productos y servicios financieros que demandan una estructura tecnológica que de manera ágil pueda proporcionar a nuestros clientes las nuevas alternativas y mantener al Banco en un nivel sí no de vanguardia, sí de competitividad. Este es el caso de la Banca Electrónica a través de banco por teléfono, banco en su casa, banco en su empresa, cajeros automáticos, tarjetas de débito e Internet principalmente.

Los sistemas de captación y en particular el sistema de cheques, eje de todos los servicios que el Banco proporciona, esta desarrollado en plataforma Unisys, en cobol, con archivos planos. El reto, y por consiguiente, el problema a resolver, es integrarlo a una infraestructura que permita lo siguiente:

- Interactuar con los sistemas del Banco que así lo requieran
- Permitir incorporar al sistema de cheques nuevas funciones y productos, y ponerlas a disposición de otros sistemas
- Permitir la migración de cheques a otra plataforma de manera paulatina sin interrumpir los servicios prestados
- Permitir el acceso a cheques a través de interfases gráficas en lugar de la actual en modo carácter
- Permitir la sustitución del sistema de automatización de sucursales que es la principal fuente proporcionadora de servicios a la clientela

Requerimientos hacia cheques

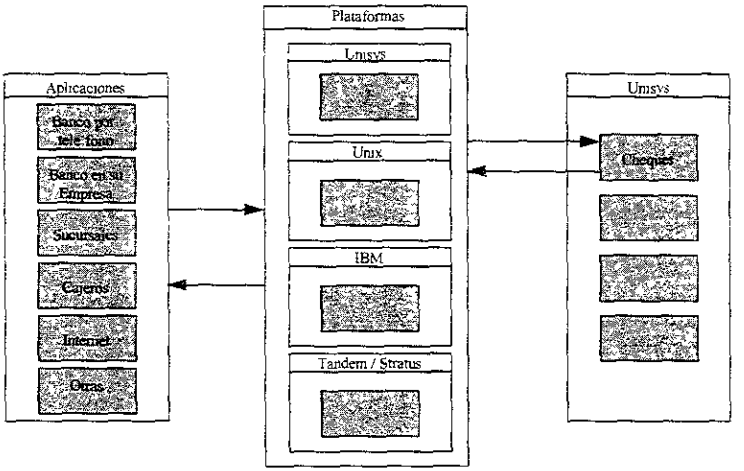

Fig 1

El Banco ante la problemática expuesta, tomó las siguientes estrategias básicas:

- Migrar los sistemas de captación a Altamira en plataforma IBM, con base de datos DB2. Altamira es un sistema bancario que permite desarrollar todas las operaciones de captación y colocación de manera estandarizada.
- Sustituir el actual sistema de automatización de sucursales desarrollado internamente, por Migesa que representa el sistema de automatización de sucursales externo que el Banco decidió adoptar como su estrategia estándar para sucursales.
- Sustituir el equipo Stratus utilizado para el control de cajeros por equipo Tandem.
- Continuar con el equipo Periphonics [1] como entrada - salida del servicio de Banca por Teléfono.
- Implementar los servicios vía Internet a través de un Web en equipo Sun

[1] Equipo que permite convertir voz a datos entre otras

Dado que en ningún momento puede suspenderse el servicio y que los sistemas en operación deben continuar sufriendo ajustes en los servicios y productos proporcionados, y que no es posible sustituir a un tiempo todo el sistema, se consideró conveniente adoptar una estrategia que permita desarrollar de manera simultánea la Banca Electrónica apoyada en los sistemas actualmente en operación y los nuevos sistemas de captación, ésta es, a través del monitor de transacciones Tuxedo, producto con el que se venían haciendo pruebas desde hace más de un año.

El esquema siguiente resume de manera gráfica la estructura propuesta para el caso de VTA (Vía Telefónica Atlántico), la cual será extendida de la misma manera para las otras aplicaciones ya mencionadas y algunas más.

A manera de ilustrar con detalle la solución propuesta, se irá presentando el caso específico de la consulta de saldo en cuenta de cheques.
Existe una aplicación particular para VTA, la cual corre en equipo Periphonics, ésta deberá hacer sus requerimientos para cheques a un Servidor de VTA (Dominio de VTA), dónde se han definido los servicios proporcionados por VTA como es la consulta de saldo en cuenta de cheques, que identificaremos como VtaChqConSdoTodo, y en ellos, los llamados a las aplicaciones responsables de cada producto o servicio. De aquí, se hace un llamado a los servicios de cheques, en nuestro caso, el servicio específico para consulta de saldo de cuenta de cheques identificado como ChqConSdo. En este punto, y dado que el sistema de cheques no se encuentra en una plataforma abierta, fué necesario desarrollar una interfase de comunicaciones que simula un llamado vía "Telnet"[2] para cheques y es aquí donde se establece la comunicación en su última parte. Una vez que llega a cheques la solicitud, éste debe reconocer el requerimiento, en este caso la consulta de saldo, y devolver la respuesta satisfactoria o no a la misma interfase.
En la figura 2 se muestra el esquema general de la solución a implementar, que ilustra el caso específico de VTA (Vía Telefónica Atlántico).

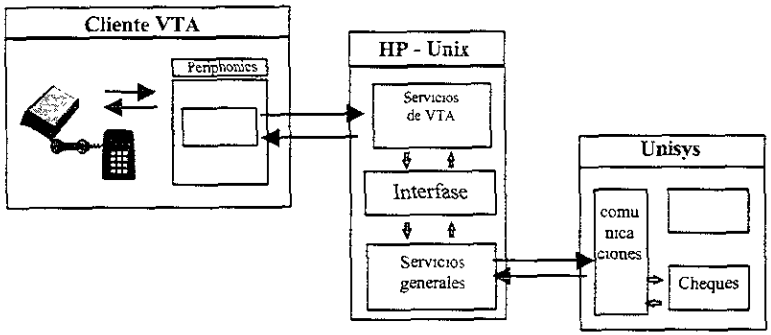

Fig. 2

[2] Emulador de terminales

Conceptos básicos.

Cliente - Servidor

Las empresas e instituciones que iniciaron su desarrollo informático en tempranas fechas, han construido aplicaciones en diferentes lenguajes y plataformas, originalmente como proceso centralizado y más tarde como procesos departamentales en micro o mini computadoras, lo que hace necesario para evitar duplicidad de información y para compartir ésta en primera instancia al concepto de proceso distribuido como solución viable especialmente en redes de computadoras donde el número de máquinas interconectadas es grande y las características de cada una diferentes, pero con tareas específicas.

Algunos ejemplos de servicios distribuidos son los servidores de impresión, de comunicaciones, de bases de datos, de administración de red, fax, correo electrónico, etc., aunque algunos pueden estar en una misma computadora de la red. Una red que integra muchos equipos, justifica dedicar algunos a tareas específicas cuando la velocidad de respuesta es uno de los elementos importantes. El tipo de procesamiento que la tecnología ha desarrollado fuertemente es el proceso cooperativo, también llamado modelo cliente-servidor.

Con el concepto cliente-servidor, mezclamos el proceso local y remoto en una aplicación para obtener lo mejor de ambas partes.

La arquitectura y el concepto de capas.

La arquitectura de las aplicaciones es una de las decisiones más importantes de un proyecto. A principios de los 80's se introdujo el concepto de tres capas para describir las particiones físicas de las aplicaciones a través de terminales (capa 1), minicomputadoras (capa 2) y equipos centrales o mainframes (capa 3), lo que inicia el manejo de computadoras de mediano rango como clientes de equipos centrales. Hoy se usan las capas para describir las particiones lógicas de aplicaciones a través de clientes y servidores.

Dos capas.

Parten el proceso en dos fases, la mayoría de las aplicaciones corren en el cliente con envíos de requerimientos al servidor (equipo central) donde reside normalmente la Base de Datos. Como una gran parte de las aplicaciones corren en el cliente, se llama arquitectura robusta en el cliente (fat client). Para solicitar los datos, el cliente debe conocer como están organizados y almacenados los datos del lado del servidor. en ocasiones, se puede ayudar de procedimientos almacenados (stored procedures) del lado del servidor, y en lugar de enviar requerimientos de datos, invoca una función que corre dentro de la Base de Datos. Algunos piensan que esta es una arquitectura de 2.5 capas.

La arquitectura de dos capas es usada ampliamente en aplicaciones departamentales. Sin embargo, si se usan en gran escala o en aplicaciones de misión crítica, empiezan a demorar la respuesta, lo que nos hace movernos de la arquitectura tradicional de dos capas cliente-servidor a un mundo más complejo donde las aplicaciones son partidas y distribuidas a través de múltiples procesadores a aplicaciones de 3 a n capas.

Tres capas.

Parten el proceso lógico entre 1) clientes corriendo **interfaces gráficas de usuario** (GUI) , 2) aplicaciones corriendo procesos lógicos del negocio en el servidor y 3) aplicaciones de Base de Datos. Esta arquitectura que mueve las aplicaciones lógicas del negocio al servidor se identifica también como arquitectura robusta en el servidor o delgada en el cliente.

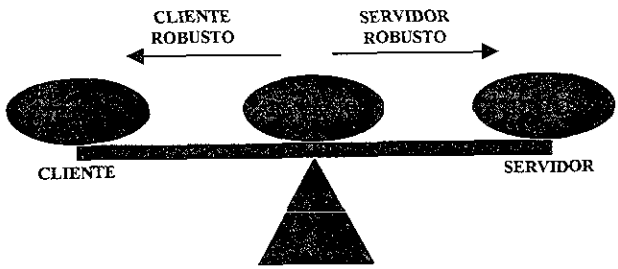

Fig. 3 **Cliente Robusto vs Servidor Robusto**

Por definición, todas las aplicaciones cliente –servidor deben tener al menos dos capas, la interface con el usuario reside en el cliente y los datos en el servidor. Hoy, las aplicaciones están creciendo de su viejo entorno departamental a un mundo con infraestructura distribuida e interconectada. La arquitectura de tres capas fue diseñada para este nuevo reto.

Arquitectura de Aplicaciones Cliente-Servidor a dos Capas

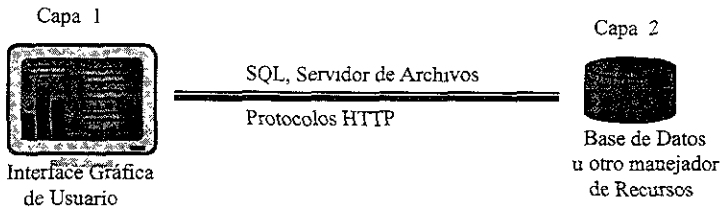

Fig. 4

Arquitectura de Aplicaciones Cliente–Servidor a tres Capas

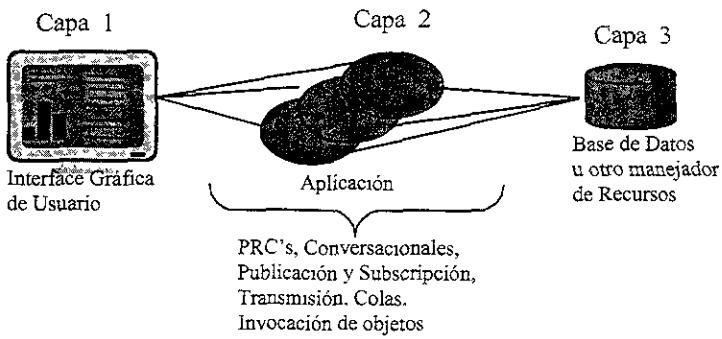

Fig. 5

La arquitectura de tres capas cubre los requerimientos de gran escala de aplicaciones cliente–servidor, de fácil manejo y donde la mayoría de código corre en los servidores, permitiendo crear niveles abstractos de servicio y en lugar de interactuar directamente con la Base de Datos, el cliente llama a procesos lógicos del negocio en el servidor. La arquitectura de tres capas sustituye unos pocos llamados al servidor por muchos requerimientos y actualización de datos, y tiene mucho mejor desempeño que dos capas. También proporciona mayor seguridad ya que no expone la Base de Datos al cliente y da al servidor un mejor manejo de las autorizaciones de acceso. El cliente invoca servicios o métodos.

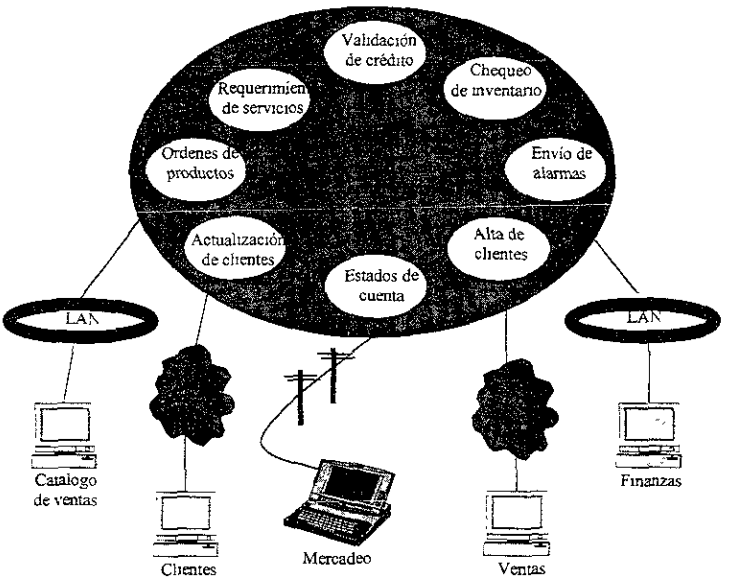

Fig. 6 **Arquitectura de n capas**

Cuando la arquitectura de tres capas se convierte a arquitectura de n capas, nace el concepto de componentes. La mayoría de las aplicaciones de tres capas no son implementadas como un programa monolítico, sino como una solución de componentes que representan pequeñas funciones del negocio, los clientes combinan varios componentes de capas intermedias dentro de una simple transacción. Un componente puede llamar a otro componente para ayudarle a generar

una respuesta a un requerimiento, algunos componentes pueden actuar como puertas de acceso a componentes o comunicaciones (gateways).

Beneficios de arquitecturas basadas en componentes sobre aplicaciones monolíticas.

Las arquitecturas basadas en componentes, permiten:

- Desarrollo de grandes aplicaciones en pequeños pasos como pequeños proyectos, se liberan versiones tempranas a producción reduciendo riesgos y tiempo.
- Componentes reusables como pequeñas cajas negras por diferentes aplicaciones.
- Los clientes pueden accesar datos y funciones con facilidad y seguridad. Los clientes no necesitan saber que Base de Datos esta siendo accesada para satisfacer un requerimiento.
- Se pueden incorporar componentes ya existentes en el mercado, desarrollados por terceros.
- Se pueden ensamblar aplicaciones rápidamente para construir nuevos clientes, adicionando capas intermedias de componentes.
- Se pueden actualizar componentes sin cambiar clientes.
- Se pueden adicionar nuevas capacidades a los procesos del negocio sin repercusiones a otras aplicaciones.

Tipos de componentes en servidores.

De servicios.

Implementan funciones del negocio.

Ejemplo: Actualización de cuentas de cheques.

De objetos

Contiene procedimientos relacionados o métodos.

Ejemplo: Métodos para manejo de cuentas de banco tales como: actualización, bajas, etc.

Hoy, los Agentes de Requerimientos a Objetos u ORB's (Object Request Brokers) proporcionan una infraestructura distribuida de objetos, proporcionan objetos para comunicaciones entre diferentes lenguajes, sistemas operativos y redes, dependiendo de su implementación.

¿ Cómo llamar a los componentes ?

Los clientes envían requerimientos a componentes usando nombres lógicos en lugar de direcciones físicas, la arquitectura de capas intermedias mapea los nombres lógicos a ubicaciones físicas y asegura la entrega del mensaje al componente. Esto además permite el manejo de réplicas en los servidores que manejan los componentes, dando una seguridad en caso de fallas y el balanceo de cargas cuando la demanda del llamado a componentes crece, ya que la infraestructura de capas intermedias rutea el requerimiento a la ubicación física donde el componente se encuentra corriendo en ese momento.

Naturalmente, el modelo de programación implementado en la infraestructura de capas intermedias determina si se llaman a objetos o servicios.

Hoy en día, en la arquitectura de capas intermedias **el Monitor de Proceso de Transacciones** o ***TP Monitor*** esta orientado a ***servicios*** y la ***ORB*** esta orientada a ***objetos***, aunque esta última ha iniciado la conexión con TP Monitors.

La **infraestructura de capas intermedias** puede proporcionar las siguientes alternativas de **manejo de mensajes**:

- **Conversacional.** Soporta un dialogo que involucra múltiples interacciones entre el cliente y los componentes en el servidor. Ejemplo: "sockets" en TCP/IP y CPI-C de IBM.
- **Requerimiento – respuesta**. Soporta una sola interacción entre el cliente y los componentes en el servidor. Ejemplo: RPC's (Remote Procedure Calls – Llamados a Procedimientos Remotos) y llamados a métodos remotos por ORB.
- **Colas**. Desacopla las interacciones entre cliente y servidor, los mensajes de requerimientos del cliente son puestos en una cola del servidor. Las colas pueden manejar mensajes con diferentes prioridades. Las colas son un importante concepto en el ambiente de tres capas.
  Los TP Monitors y CORBA 3.0 (Common Object Request Broker Architecture) incluyen el manejo de colas.
- **Publicar-Suscriptor.** Registra el interés de clientes o algunos componentes del servidor en ciertos mensajes en un manejador de eventos. Componentes del servidor o clientes "**publican**" mensajes en el manejador de eventos quien los envía a suscriptores que han registrado su interés en un tipo particular de mensajes. Pueden manejar colas de mensajes para diferentes componentes. MOM (Message Oriented Middleware), TP Monitors y CORBA incluyen esta característica.
- **(Broadcast and datagrams) Transmisión y diagramas de datos.** Establecen un medio de comunicación para uno o varios componentes o clientes. CORBA y TP Monitors la incluyen.

- **Agente de Mensajes (Message Brokers).** Permite a una aplicación notificar a otras que un evento se ha presentado. La aplicación que origina el mensaje lo envía al MOM, quien lo encola para su envío posterior, incluso puede cambiar de formato al mensaje. Ejemplo: Podemos notificar (informar) a varias aplicaciones cuando una persona abre una cuenta de cheques. El manejo de colas de mensajes trabaja bien si las aplicaciones notificadas no pueden impedir que se abra la cuenta, en caso contrario, la aplicación original deberá notificar a todas las aplicaciones involucradas que la cuenta se cerro o no se abrió, por la que cada aplicación debe tener una transacción de reversa (rollback).

Aunque las aplicaciones de tres capas están creciendo en popularidad, las de dos capas siguen siendo buenas para proyectos pequeños.

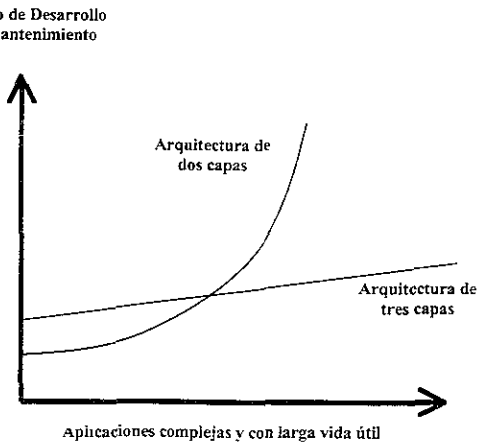

Fig. 7 **Comparando dos capas y tres capas**

Según Gartner Group debemos usar tres capas si la aplicación a desarrollar tiene las siguientes características:

- Más de 50 servicios o clases.
- Uso de diferentes lenguajes en las aplicaciones

- Dos o más Bases de Datos diferentes.
- Vida útil de la aplicación mayor a tres años.
- Posibilidad de muchas modificaciones o adiciones.
- Alto volumen de transacciones (más de cincuenta mil por día)

Los TP Monitors están incorporando el protocolo de **two-phase commit** para asegurar la integridad de las transacciones. el cual detallaremos más adelante.

Los TP Monitors fueron desarrollados para dar servicio a miles de clientes, sirviendo como interface entre clientes y servidores. Pueden manejar transacciones enrutadas a través de diversos sistemas, balancear cargas durante su ejecución y reiniciarlas después de una falla, proporcionando un mejor desempeño del sistema, puede manejar recursos de uno o varios servidores y puede cooperar con otros TP Monitors

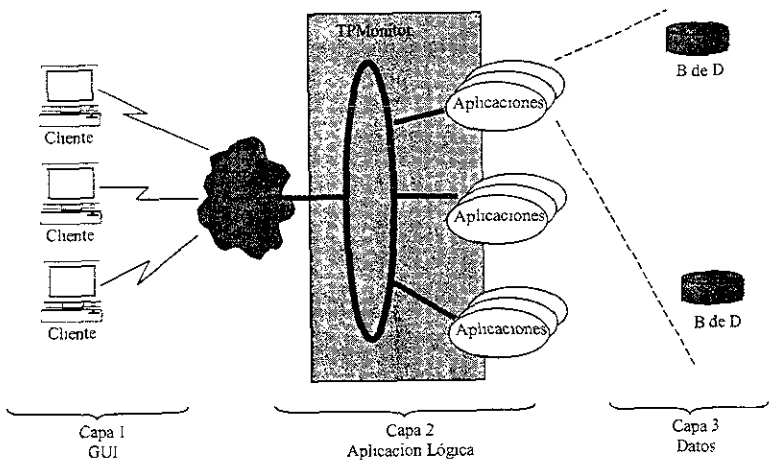

Fig. 8 **Tres capas cliente-servidor con TP Monitor**

¿Cómo trabajan las aplicaciones basadas en TP Monitors?

Podemos decir que los TP Monitors proporcionan infraestructura preconstruida que ayuda a construir, correr y administrar una aplicación cliente-servidor robusta y con alto desempeño.

Los TP Monitors proporcionan del lado del servidor **procedimientos reusables** modulares llamados **servicios** que encapsulan un manejador de recursos.

Un manejador o administrador de recursos es cualquier pieza de software que administra recursos compartidos como un Administrador de Base de Datos, de Colas o de Transacciones. La aplicación adiciona nuevos llamados a funciones y el TP Monitor distribuye la función sobre múltiples servidores.

Los TP Monitors permiten adicionar recursos heterogéneos en el servidor sin alterar la arquitectura de las aplicaciones existentes. Manejan transacciones desde su punto de origen (normalmente un cliente) a través de uno ó más servidores y devuelven la respuesta al punto de origen.

Los TP Monitors controlan todo el tráfico que liga cientos o miles de clientes con programas aplicativos y los recursos que los soportan. Estos procesos tienen así una existencia independiente de la Base de Datos y las interfaces gráficas de usuario (GUI's). Administran todos los aspectos de una aplicación distribuida sin importar los sistemas o recursos usados. Pueden manejar recursos en un servidor o en múltiples servidores y pueden cooperar con otros TP Monitors.

Los TP Monitors al balancear las cargas proporcionan mejor desempeño usando los mismos recursos del sistema y permiten correr aplicaciones en hardware menos costoso y no requieren una solución específica de Base de Datos, además de utilizar interfaces estándares. Lo anterior según Standish Group ahorra hasta un 30% del costo y de un 40 a un 50% del tiempo de desarrollo respecto al desarrollo tradicional.

Del lado del servidor, los TP Monitors posibilitan la creación de procedimientos modulares reusables llamados servicios como ya se ha mencionado, que encapsulan recursos compartidos.

Los TP Monitors proporcionan una cubierta de servicios de propósito general llamada "clase de servicios" para correr los servicios en capas intermedias e introducir un estilo de programación manejado por eventos en el lado del servidor. Se exporta la función "call" (de llamado) y no los datos mismos, lo que permite adicionar nuevas funciones que los TP Monitors distribuyen sobre múltiples servidores , creando aplicaciones altamente complejas por la adición de más servicios.

Los TP Monitors aplican como principio del proceso de transacciones de aplicaciones las propiedades ACID.

ACID Atomicity, Consistency, Isolation & Durability. Término acuñado por Andreas Reuter en 1983 para definir los estándares de **atomicidad, consistencia, aislamiento y durabilidad** para las transacciones.

Atomicidad. La transacción es una unidad de trabajo indivisible, o todas su componentes suceden satisfactoriamente o "todas" fallan (si al menos una falla). Esto es, o todo o nada, la transacción sucede o no sucede.

Consistencia. Significa que después de que una transacción ha sido ejecutada, ella debe regresar al sistema como un estado: satisfactorio o no satisfactorio. Si la transacción no puede alcanzar su estado final, regresa el sistema a su estado inicial (tal como se encontraba al inicio de la transacción).

Aislamiento. Significa que el comportamiento de una transacción no es afectado por otra transacción que se ejecuta concurrentemente. La transacción serializa todo acceso a recursos compartidos para garantizar que las transacciones o programas que corren concurrentemente no alteran unos las operaciones de otros.

Durabilidad. Significa que los efectos de una transacción al terminar esta, son permanentes y sus cambios no se ven afectados por fallas del sistema.

De manera ideal, todo programa cliente-servidor debería ser escrito como transacciones Acid.

Los TP Monitors aparecen primero en los mainframes para proporcionar un ambiente robusto para el proceso de transacciones en línea OLTP (On Line Transaction Processing), aplicaciones que requieren respuesta inmediata, tales como reservaciones de aerolíneas, transacciones bancarias, etc.

Al moverse las aplicaciones OLTP a arquitecturas cliente-servidor y sistemas abiertos, se crea un nuevo ambiente de desarrollo basado en TP Monitors que hacen tres cosas muy bien:

Administración de procesos, incluyendo el arranque del servidor, monitoreando su trabajo y balanceando cargas.

Administración de transacciones, lo que significa que el TP Monitor garantiza las propiedades Acid para todo programa que corre bajo su ambiente.

Administración de comunicaciones cliente-servidor que permiten a clientes y servidores invocar a los componentes de una aplicación de diferentes maneras (requisición – respuesta, conversacional, colas, publicación y suscripción, ó emisión) .

Los TP Monitors y los sistemas operativos como el gran concentrador.

Los TP Monitors fueron introducidos inicialmente para correr aplicaciones que pudieran atender a cientos y a miles de clientes , proporcionando los recursos que requieran del servidor. Funcionan como una capa sobre el sistema operativo que conecta en tiempo real a estos clientes con un pool de servidores, balanceando el uso de los recursos sobre la base de la demanda.

Sin TP Monitor

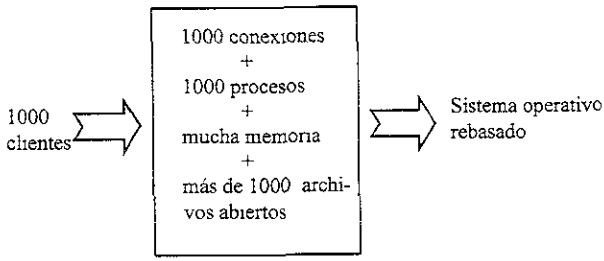

Fig. 9

Con TP Monitor

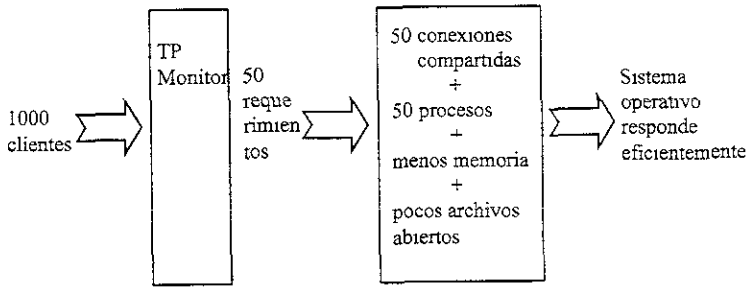

Fig. 10

Los TP Monitors funcionan como un embudo concentrador de requerimientos que atienden a través de un pool de procesos en el servidor. El servidor de procesos liga dinámicamente a las funciones de servicios llamadas por los clientes, supervisa su ejecución y devuelve el resultado al cliente.

Después de completar un requerimiento, otro cliente puede rehusar el proceso de servicio. El servidor conserva los servicios cargados en memoria, donde pueden ser compartidos. Si el número de requerimientos que entran al servidor excede al número de procesos de servicio disponibles en el servidor, el TP Monitor puede dinámicamente levantar o arrancar nuevos. esto se llama balanceo de cargas. Un TP Monitor más sofisticado puede incluso efectuar el balanceo del proceso de servicios a través de múltiples procesadores. Parte del balanceo de carga esta constituido por el manejo de prioridades tanto del lado del cliente como del lado de las funciones llamadas en el servidor.

Con los TP Monitors, las aplicaciones no tienen que manejar la concurrencia, el manejo de fallas, interrupciones de comunicaciones, balanceo de cargas y sincronización de recursos a través de múltiples nodos. Todo esto se hace transparente a las aplicaciones, del mismo modo que el sistema operativo hace transparente el hardware a las aplicaciones.

Los TP Monitors están siendo asociados con el protocolo Two–phase commit para asegurar la integridad de las aplicaciones.

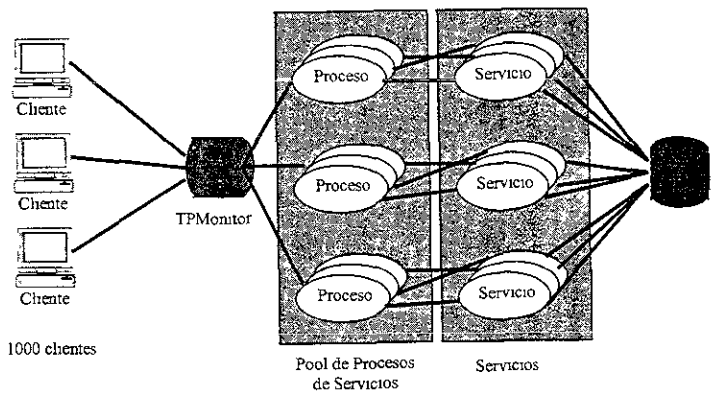

Fig. 11

¿ Qué es el protocolo Two-Phase Commit ?

Es la sincronización de las actualizaciones, o todas suceden o todas fallan. El nodo raíz o nodo coordinador de transacciones, coordina que todos los nodos subordinados o participantes terminen satisfactoriamente su participación, lo cual ocurre cuando recibe dicha señal de todos y cada uno (primera fase del commit). después comunica a los nodos subordinados que termina su parte en la transacción, el nodo raíz espera confirmación y avisa al cliente que la transacción fue completada (segunda fase del commit).

Si alguno de los nodos subordinados regresa una indicación de que su parte no fue completada, es decir, que fallo o no contesta en un tiempo preestablecido, el nodo raiz indica a todos los nodos subordinados que desarrollen un **rollback**, es decir, **que regresen la información a su estado inicial** y avisa al cliente que la transacción no fue realizada.

Monitor de transacciones tuxedo

¿ Qué es Tuxedo ?

Es una plataforma de capas intermedias usada por la mayoría de las aplicaciones cliente-servidor de tres a n capas.

Se basa en una arquitectura que permite identificar los elementos estructurales que pueden ser usados como bloques en la construcción de sistemas cada vez más complejos.

Proporciona una interface para el manejo y la administración de transacciones y servicios a través de funciones que soportan clientes y servidores conocida como ATMI (Application Transaction Manager Interface) y un lenguaje para comunicación entre clientes y servidores llamado FML (Field Manipulation Languaje) .

El ATMI posee las siguientes características:

- Manejo de errores
- Manejo de buffers
- Conexión
- Solicitud / Respuesta
- Conversacional
- Solicitudes almacenadas

- Prioridades
- Notificaciones
- Publicación dinámica
- Terminación de servicios
- Transacciones

Maneja dos esquemas básicos:

El Administrador de aplicaciones Tuxedo

- Maneja la instalación y la configuración inicial
- Configura cada aplicación
- Monitorea aplicaciones
- Inicia y termina las aplicaciones
- Efectúa el mantenimiento a las aplicaciones Tuxedo
- Reconfigura dinámicamente las aplicaciones

El desarrollador de aplicaciones Tuxedo

- Construye aplicaciones Cliente
- Construye Servicios

Tuxedo garantiza la integridad de datos y transacciones, su distribución a las aplicaciones encargadas de resolverlas y el monitoreo y administración de la operación, creando independencia entre las aplicaciones, sus plataformas y lenguajes de programación, permitiendo un servicio más eficiente y ágil hacia el usuario final.

El siguiente cuadro ilustra de manera amplia los servicios proporcionados por Tuxedo

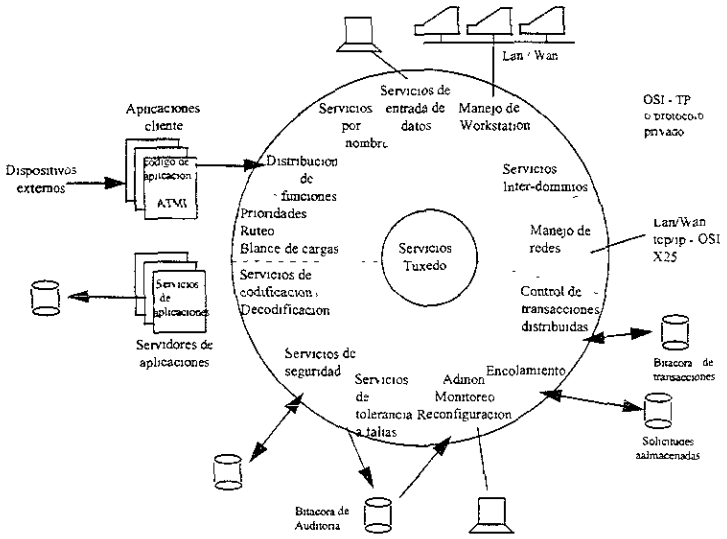

Fig. 12

El Modelo cliente - servidor

El modelo cliente - servidor constituye sin lugar a dudas, la tendencia más importante en el desarrollo de aplicaciones hoy en día. Básicamente tenemos grandes computadoras en un ambiente de redes conectadas con sistemas personales bajo Windows, en las PC's generalmente desplegándose la información al usuario y por el otro lado las máquinas grandes o medianas almacenan cantidades importantes de datos (información) y ejecutan grandes programas.
Los clientes y servidores son entidades lógicas independientes que trabajan juntas sobre una red para completar una tarea.

El modelo cliente - servidor, es un modelo de construcción de programas en el cual un proceso llamado el servidor controla y administra recursos solicitados por aplicaciones llamadas clientes. Este es un modelo de construcción de aplicaciones, donde si una máquina tiene como principal función dar servicios, se le puede llamar servidor (*server*); sin embargo, una máquina puede ser servidor, porque tiene un proceso que proporciona servicios, y cliente al mismo tiempo, porque tiene otro proceso que solicita un proceso a un servidor.

Los servidores pueden clasificarse de acuerdo a la naturaleza del servicio que proporcionen en servidores de:

- archivos (*File Servers*)
- base de datos (*Database Servers*)
- transacciones (*Transaction Servers*)
- grupo (*GroupWare Servers*)
- objetos (*Object Servers*)
- Internet (*Web Servers*)

De acuerdo a la distribución podrán identificarse como Clientes o Servidores.
Entre los beneficios que proporciona este modelo están:

- Modularidad . Independiza funciones lógicas.
- Desempeño. Optimiza el manejo de recursos del equipo central y aprovecha la potencialidad del Cliente.
- Independencia de datos. Permite modificar el esquema físico sin repercusiones en los programas.
- Extensibilidad. Permite la incorporación de nuevas funciones y nodos.
- Envío de datos o de funciones. Incrementa la potencialidad de las aplicaciones.

Tuxedo proporciona una infraestructura de capas intermedias que permite escribir aplicaciones empresariales de misión crítica cliente-servidor como una colección de servicios. Tuxedo administra estos servicios y coordina su uso en transacciones. Los servicios pueden accesar gran variedad de Bases de Datos. Tuxedo asegura la integridad de las transacciones que administra, incluye infraestructura para el manejo de mensajes, acceso a más de 50 plataformas y soporte a Internet.
A continuación se encuentra el modelo cliente - servidor utilizado por Tuxedo

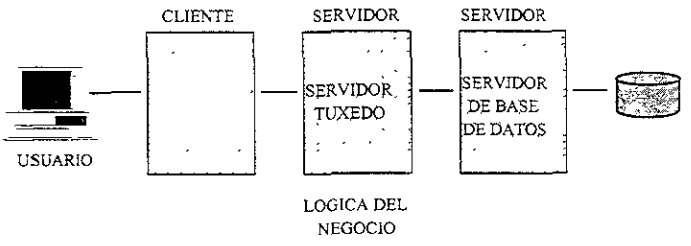

Fig. 13

El **servidor** es un proveedor de servicios, el **cliente** es un consumidor de esos servicios
Un servidor puede atender a muchos clientes al mismo tiempo y regular sus accesos a los recursos compartidos. Se basa en un intercambio de mensajes, dónde éste tiene dos funciones básicas, entrega para la **solicitud** de un servicio, y devolución de la **respuesta**.

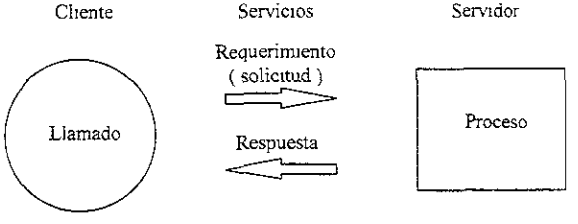

Fig. 14

El código y los datos del servidor son centralizados, en tanto que los clientes son independientes.
El servidor es un agrupador de servicios, que además administra éstos, los recursos involucrados, su acceso e integridad.
El número de servidores activos puede variar dinámicamente para mantener el mejor nivel de respuesta. Si los requerimientos son altos, es factible duplicar el servidor total o parcialmente, de manera que se tengan dos ó más servidores para la atención de los servicios demandados, ó bien disminuir el número de servidores cuando la demanda baje.

Por ejemplo, supongamos que iniciamos operaciones con un servidor atendiendo la consulta de saldo de cuenta de cheques vía telefónica, y que a medio día, la demanda a crecido tanto que los mensajes empiezan a encolarse y el tiempo de respuesta a incrementarse más allá de lo conveniente, entonces levantamos otro servidor con el mismo servicio, y distribuimos los requerimientos de información, esto es, las solicitudes serán atendidas entre los dos servidores (la cola se parte), disminuyéndose el tiempo de respuesta.
Como se mencionó, pueden enviarse datos o funciones, con las siguientes características:

Envío de datos

- El código está contenido en el cliente, maneja la interacción con el usuario y las comunicaciones con la base de datos.
- El servidor es provisto por el manejador de la base de datos y garantiza su consistencia.

Envío de funciones

- El servidor maneja la comunicación con la base de datos.
- El cliente realiza las interacciones con el usuario
- Los servicios son solicitados por nombre

Tuxedo maneja el esquema de envío de funciones, reduciendo el tráfico en la red, manejando prioridades y encolamiento. No requiere conexión con cada usuario.
A continuación se muestran los esquemas que ilustran las principales etapas de una aplicación cliente y de una servidor.

Aplicación cliente

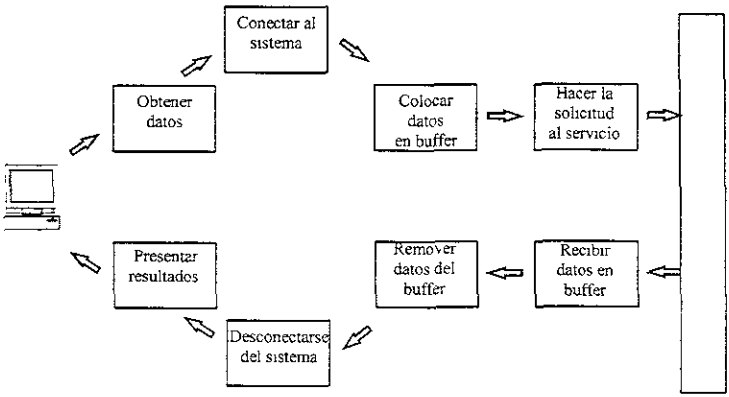

Fig. 15

Aplicación servidor

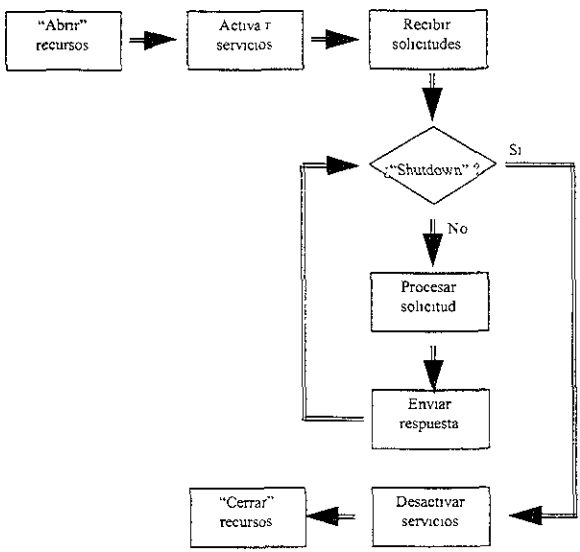

Fig. 16

Algunas de las principales características de este modelo son las siguientes:

- Actúa como un proceso "demonio" que acepta solicitudes y las despacha a los servicios correspondientes
- Un servidor puede contener varios servicios
- Un mismo servicio puede estar contenido en varios servidores
- Varios servicios pueden ser manejados por una misma rutina o estar contenidos en rutinas independientes
- Un servidor puede actuar como un cliente cuando a su vez solicita servicios a otro servidor (reenvío de solicitudes). En este caso, Tuxedo rutea la solicitud al servidor apropiado. Este último caso resulta muy adecuado cuando se tiene un red de área amplia (Wan)

La comunicación de la información se efectúa a través de un área en memoria que tiene un tipo y un subtipo asociados llamada *buffer* [1].

Tuxedo maneja el concepto de dominios. es decir, puede agrupar un grupo de servicios que sólo atienden a un tipo de aplicaciones, restringiendo su uso y facilitando su control, los cuales pueden aplicarse a:

- Aplicaciones Tuxedo
- Aplicaciones en equipos centrales como IBM, Unisys ó Tandem, entre otros.

Este concepto resulta útil para agrupar los servicios por sistemas específicos, como pueden ser cada uno de los que conforman la Banca Electrónica.

Una aplicación Tuxedo bien construida, puede identificar problemas en:

- Aplicación
- Sistema Tuxedo
- Manejador de la base de datos
- Red
- Sistema operativo
- Hardware

[1] Area de memoria reservada para guardar información de entrada - salida

Tuxedo usa el protocolo *Two-Phase commit* [2] pero para ello se debe contar con aplicaciones en base de datos, de lo contrario, el reverso de operaciones incompletas no puede hacerse de manera automática y requiere la implementación de nuevas funciones en la aplicación que finalmente debe resolver el servicio solicitado.

El modelo cliente - servidor con Tuxedo trabaja de la siguiente manera:

1. El cliente consulta al sistema Tuxedo , el cual checa en sus tablas la dirección de la cola del servidor solicitado.
2. La solicitud es enviada a la cola del servidor.
3. El servidor procesa la petición y envía la respuesta a la cola del cliente.

El servidor se queda en un ciclo hasta que llega un mensaje de apagado (shutdown), el cual lo deshabilita.

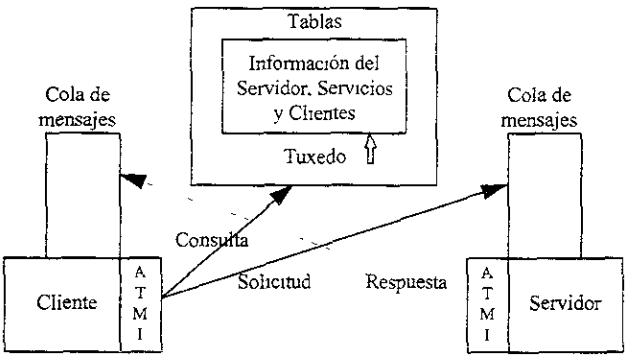

Fig. 17

Una aplicación Tuxedo consta de:

- Un archivo de configuración
- Código del cliente
- Código del servidor

[2] Garantiza la terminación exitosa de una transacción con actualización a BD o en caso de no terminación, deshace la parte efectuada

- Comandos para iniciar y ejecutar la aplicación

Normalmente, los servicios Tuxedo tienen entre sus funciones el acceso a los Datos, sin embargo, cuando éstos no están en una Base de Datos y las aplicaciones tienen un alto grado de control sobre su actualización y acceso, como es el caso que nos ocupa, resulta conveniente mantenerlo en primera instancia, por lo que es necesario incorporar a la aplicación que controla la actualización una parte del código del servicio.

Por su parte, el Cliente debe efectuar los siguientes pasos:

- Efectuar las interfaces con el usuario y determinar que operación debe ser desempeñada.
- Preparar la petición del servicio
- Iniciar la petición del servicio
- Procesar la petición del servicio
- Reportar los resultados al usuario

Tuxedo proporciona los siguientes métodos para comunicación con el cliente:

- Síncrono. El cliente realiza un llamado y se bloquea esperando respuesta.
- Asíncrono. El cliente realiza un llamado, este se inicia, pero no espera su conclusión, para poder continuar con otras operaciones, más tarde recupera la respuesta.
- Conversacional. El cliente establece un diálogo con uno o más servidores conversacionales
- Carga. El cliente manda una petición a la cola, la petición será atendida de acuerdo al administrador de colas y la respuesta recuperada más tarde.
- No solicitados. El cliente puede mandar mensajes no solicitados a otros clientes

El *ATMI* [3] proporciona las siguientes funciones para establecer la interface para el manejo y administración de transacciones y servicios:

- Manda un número de error, tperrno( ) y un mensaje, tpstrerror( )
- Maneja todo lo referente a la creación, tpalloc( ) y tprealloc( ) ; tamaño, liberación, tpfree( ) y verificación, tptypes( ) del tipo de buffer usado para la comunicación de datos.

[3] Application Transaction Manager Interface

- Las funciones tpacall( ) y tpcall( ). mandan los buffer a los servicios solicitados y las respuestas se solicitan con tpgetrply( ), o se cancelan con tpcancel( ).

El cliente utiliza tpacall( ) y tpgetrply( ) para comunicación asíncrona y tpcall( ) para síncrona.

- El usuario puede determinar la prioridad del último mensaje enviado o recibido con la función tpgprio( ), ó agregar prioridad para la petición de los servicios con tpsprio( ).
- La función tpconnect( ) establece una conversación: para mandar un mensaje se utiliza la función tpsend( ) y para recuperar la respuesta. tprecv( ). Finalmemte, para terminar la conversación se utiliza la función tpdiscon( ).
- Para enviar a la cola un mensaje. se utiliza la función tpenqueue( ) y para desencolar un mensaje, tpdequeue( ). Esto es. tpenqueue( ) carga una solicitud y tpdequeue( ) carga una respuesta.

Tuxedo maneja varios tipos de buffer ATMI, según se requiera. éstos son.

- STRING - Buffer utilizado para datos de tipo carácter como un texto.
- CARRAY - Buffer utilizado para datos de tipo carácter, pero donde los caracteres nulos son de mayor tamaño, como es el caso de los gráficos.
- FML [4] - Buffer diseñado para datos que permiten nombrar campos con 0 ó más ocurrencias de datos por nombre de campo, como son los registros de Base de Datos.
- VIEW - Buffer que permite hacer uso de estructuras del lenguaje C

Buffer tipo STRING.

Es conveniente para aplicaciones en línea. mensajes cortos. no tiene terminador nulo y realiza codificación - decodoficación Ofrece compatibilidad muy aceptable entre diferentes tipos de computadoras.

Buffer tipo CARRAY.

Es comúnmente utilizado para grandes mensajes de longitud fija (como imágenes), puede tener varios terminadores nulos como parte de su contenido. No realiza codificación - decodificación.

[3] Application Transaction Manager Interface
[4] Field Manipulation Language

Buffer tipo FML

Puede ser una tabla de atributos y valores

Ejemplo:

NOMBRE	Jesús
DIRECCION	Martín Mendalde 17
TELEFONO	601 12 49

Soporta 7 tipos de datos: *Short, char, long, float, double*, STRING y CARRAY.

Proporciona funciones de interface como son :

alojamiento y liberación de buffers

añadir y cambiar campos en un buffer

extraer valores de los buffers

leer e imprimir un buffer

Ventajas

Proporciona la facilidad de manejar múltiples datos en un mismo buffer.

Los buffers sólo contienen los campos necesarios para cada operación.

Sólo la parte que se necesita del buffer es enviada en la petición.

Soporta múltiples ocurrencias

Desventajas.

La interface FML es la más complicada de las intefaces (FML, VIEW, STRING y CARRAY) porque se debe manejar cada campo individualmente.

El acceso a los campos es lento en relación a otros tipos de buffer.

Buffer tipo VIEW.

Permite gran flexibilidad en el manejo de los servicios. consiste de funciones y definiciones de Vistas (VIEW's)

Componentes del VIEW

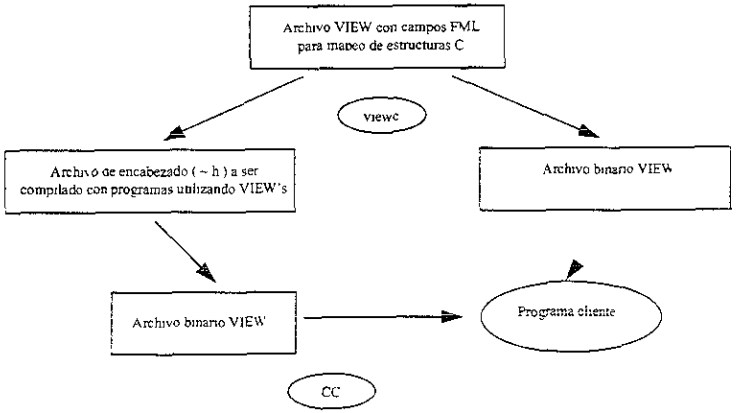

Fig. 18

Estructura de los archivos VIEW

Este archivo contiene la siguiente información:

Nombre da la estructura en C (cname)

Nombre del campo FML (fbname)

Número de ocurrencias (count)

Banderas (flag)

Tamaño (STRING/CARRAY) (size)

valor nulo (NULL) (null)

\# comentario que será ignorado

S comentario que se copiará en el archivo ~.h

Ejemplo:

VIEW aud

S /* VIEW structure for audit information */

type cname fbname count flag size null

long	b_id	Branch_id	1	-	-	0
float	balance	balance1	-	-	0.0	
sting	ermsg	Statlin1	-	80	""	

La salida del comando *viewc* incluye un archivo de encabezado que contiene la definición de la estructura que podría utilizarse por la aplicación

Ejemplo:

```
/* VIEW structure for audit information */


struct aud {
        long b_id;              /* null = 0 */
        float balance;          /* null=0.00000 */
        char ermsg[80] ;        /* null="" */
};
```

Variables de ambiente

VIEWFILES: Variable de ambiente necesaria para que Tuxedo al momento de levantar la aplicación identifique los archivos binarios de los buffers VIEW's

VIEWDIR: Variable de ambiente necesaria para indicar la trayectoria (*Path*) a Tuxedo, para que encuentre los archivos fuente de los buffers View's

TUXCONFIG: Esta variable de ambiente contiene la trayectoria (**path**) del archivo binario TUXCONFIG

A continuación mostramos ejemplos de conversión de FML's a VIEW's y viceversa.

de FML a VIEW

int Fvftos (FBFR *bufptr, char *cstruct, char *view)

*bufptr apuntador al buffer FML del dato fuente a copiar

*cstruct apuntador a la estructura C del dato destino

*view nombre de la estructura

- si un campo FML no está en VIEW , es ignorado
- si son menos ocurrencias en la estructura C que en el FML-VIEW, todo es ignorado
- si son más ocurrencias en la estructura C que en el FML-VIEW, valores nulos llenan los espacios vacios.

de VIEW a FML

int Fvstof (FBFR *bufptr, char *cstruct, int mode, char *view)

*bufptr apuntador al buffer FML del dato destino

*cstruct apuntador a la estructura C del dato fuente

*mode modo a utilizar. Puede ser *FUPDATE, FJOIN, FOJOIN y FCONACT*[5]

*view nombre de la estructura

Estos son los posibles valores de unión entre estructuras C y formato FML -VIEW

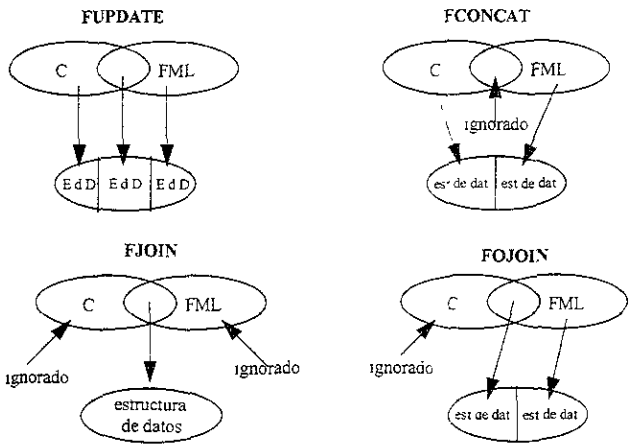

Fig. 19

[5] posibles **valores de unión** de datos entre estructuras C y formatos FML-VIEW

La configuración del Sistema / Tuxedo se realiza en un archivo ascii llamado UBBCONFIG. En este archivo, el administrador de Tuxedo coloca los parámetros para uso del sistema y lista los servidores y servicios con los que se trabajará.

El archivo UBBCONFIG es compilado para uso del Sistema / Tuxedo con el comando tmloadcf y se obtiene el binario TUXCONFIG

Tuxedo se habilita, levanta o se inicia con el comando tmboot y se termina con el comando tmshutdown

Interfaces ATMI para servicios

La interfaz ATMI proporciona diversas rutinas que dan a los servidores un comportamiento funcional dentro del sistema Tuxedo

- Las rutinas tpinit( ) y tpterm( ) son invocadas automáticamente por la interfaz.
- tpsvcinfo es un apuntador al mensaje recibido como parámetro en el servicio.
- Manejo de buffers y funciones conversacionales
- La función tptypes( ) verifica el tipo de buffer.
- Las funciones tpadvertise( ) y tpunadvertise( ) levantan y tiran automáticamente un servicio.
- tpnotify( ) y tpbroadcast( ) envían mensajes no solicitados.
- tpreturn( ) y tpfordward( ) envían mensajes a un cliente u otro servicio respectivamente.
- tpsvrinit( ) y tpsvrdone( ) se encargan de levantar o tirar un servicio opcionalmente.

La estructura de tpsvcinfo es la siguiente:

```
struct tpsvcinfo {
   char name[32];       /* nombre del servicio invocado */
   long flags;          /* atributos del servicio */
   char * data;         /* apuntador al buffer de datos */
   long len;            /* longitud del buffer de datos */
   int cd;              /* identificador de conexión */
```

```
long appkey;        /* llave de autentificación de la aplicación */
CLIENTID cltid;     /* identificador del cliente */
};
```

Los puntos principales a considerar en el manejo de buffers son:

- Las funciones tpalloc, tprealloc y tpfree, alojan, realcjan o liberan buffers
- Si el buffer utilizado en el tpreturn es diferente del buffer de entrada de datos, se debe liberar el buffer de entrada.
- Es posible y conveniente realojar un buffer recibido, para darle mayor espacio en memoria.
- Cuando se envía una respuesta al cliente, tpreturn libera automáticamente la memoria del buffer de respuesta, pero se debe tener cuidado de liberar los buffers, de no hacerlo, se puede agotar la memoria disponible.

A continuación ilustraremos las situaciones más comunes que se presentan en el sistema:
El cliente Tuxedo se comunica con el servidor Tuxedo (Fig. 20) utilizando la función tpcall(), la cual envía una petición a un servicio determinado. El servicio Tuxedo es una función escrita en lenguaje C, la cual atiende al llamado del cliente y envía una respuesta utilizando la función tpreturn( ).

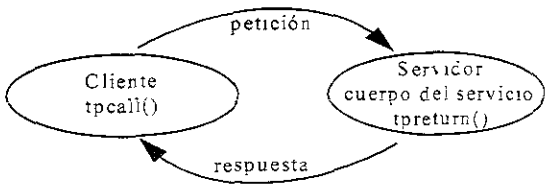

Fig. 20

También se muestra en la figura 21 a un servicio actuando como cliente (servicio invocando a otro servicio) mediante la siguiente secuencia de eventos:

- Un cliente hace una petición a un servicio con la función tpcall( )
- El servidor 1 recibe la llamada del cliente y manda un tpcall( ) al servidor 2 y queda en status de espera
- El servidor 2 recibe la llamada , resuelve la transacción y manda la respuesta al servidor 1 con un tpreturn( )
- El servidor 1 recibe la respuesta y manda la respuesta final al cliente con un tpreturn( ).

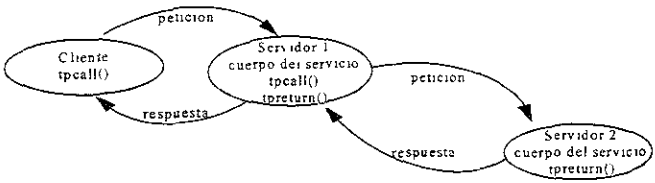

Fig. 21

Servicio conversacional exitoso

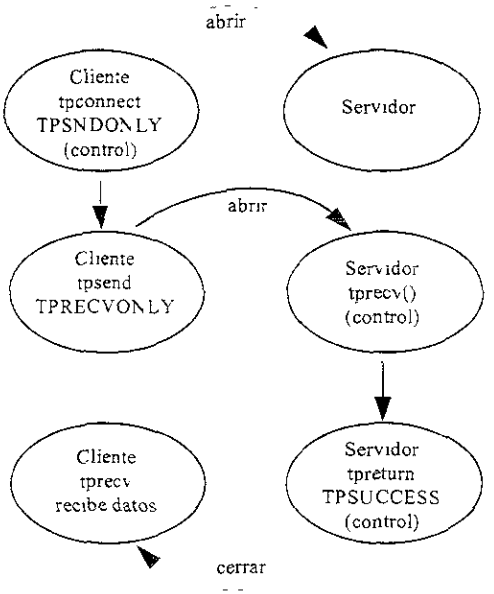

Fig. 22

El diagrama ilustra el uso del tpreturn( ) para terminar un servicio conversacional exitosamente.

1. El cliente abre la conexión utilizando la función tpreturn( ). La bandera TPSENDONLY retiene el control en el cliente.
2. El cliente usa la función tpsend( ) con TPRECVONLY pasando el control al servidor.
3. El servidor usa un tpreturn( ) con TPSUCCESS para enviar al cliente la respuesta y terminar la conversación (cierra la conexión)
4. El cliente recibe el dato con recv( ).

Servicio conversacional no exitoso

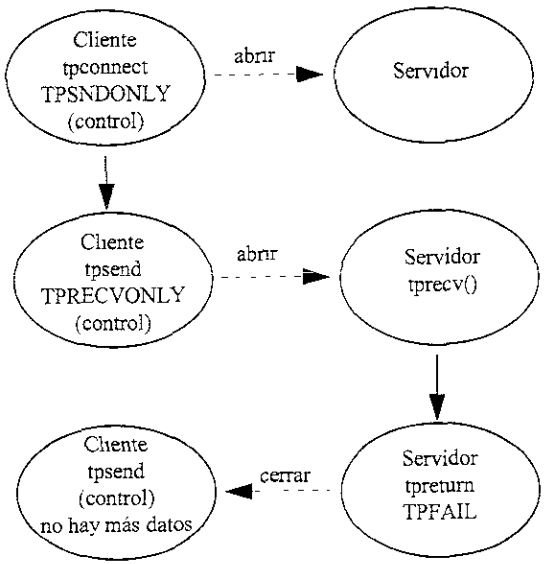

Fig. 23

1. El cliente establece la conexión con tpconnect( ), la bandera TPSENDONLY retiene el control en el cliente
2. El cliente manda un tpsend( ) con TPSENDONLY reteniendo el control.
3. El servidor recibe el mensaje con tprecv( ) y determina que no tiene respuesta que enviar.
4. Cuando el servidor detecta que no tiene el control de la conexión envia un tpreturn( ) con la bandera TPFAIL y termina la conversación.
5. El cliente ya no tiene datos que enviar

Entorno tpforward

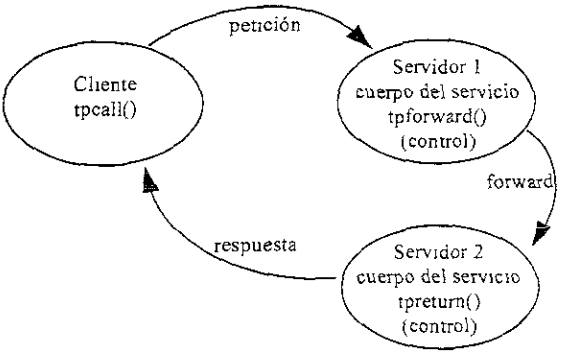

Fig. 24

1. El cliente manda un tpcall( ) al servidor 1
2. El servidor 1 manda el control al servidor 2 utilizando la función tpforward( )
3. El servidor 2 resuelve la transacción y manda la respuesta al cliente con la función tpreturn( ). Envía el buffer de respuesta y el control al cliente que invocó el servicio.

Las partes básicas de un servidor Tuxedo son las siguientes:

1. La función *main* estándar que proporciona Tuxedo.
2. Creación de una cola de mensajes.
3. Llamado a la función **tpsvrinit( )** la cual abre los recursos utilizados por el servicio.
4. El servidor se queda en un ciclo hasta que llega un mensaje de shutdown
5. Llamado a la función **tpsvrdone( )** la cual cierra los recursos utilizados por el servicio.
6. Manejar las relaciones con el Bulletin Board [6]
7. Manejar como argumentos comandos en línea

[6] Registro de avisos para usuarios y administradores

Para crear un servidor se requiere el comando **buildserver** , el cual lo compila. Se deben declarar los archivos de cabecera (.h) necesarios para el servicio. Entre ellos el que se utiliza cuando el servicio intercambia información utilizando un buffer FML, archivo fml.h

El servicio se declara como una función y tiene como parámetro de entrada la estructura TPSVCINFO.

Se declaran las variables del servicio entre ellas el buffer FML.

Para extraer datos del buffer se requiere de las funciones Fget (para cualquier tipo) y Frall (para datos short o long).

Se debe modificar el *ubbfile* donde se especifica cómo se llamará el servidor y generar un nuevo archivo *tuxconfig*.

Dominios en Tuxedo.

Proporcionan la posibilidad de interoperabilidad entre servidores ubicados en diferentes lugares físicos, perimitiendo la definición de áreas de influencia locales denominadas dominios o *domains o /TDOMAIN*, así como un subsistema administrativo que permite crear o cambiar configuraciones, verificar actividades realizadas por una instancia de un dominio o monitorear y ajustar sus parámetros para mejorar su desempeño.

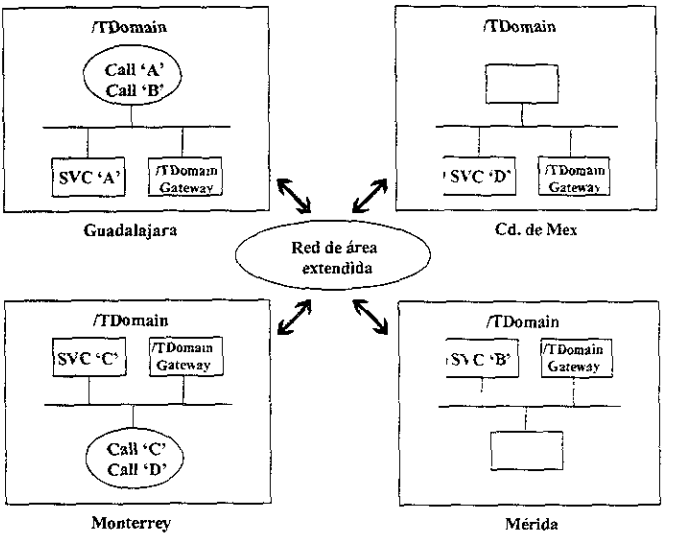

Fig. 25

SVC 'A' Servicio A

La interoperabilidad se produce en base a la especificación OSI[7] para procesamiento distribuido (conocido como OSI /TP) . Es el único protocolo estándar disponible para DTP (Distributed Transaction Processsing) y ha sido seleccionado por X/OPEN como el protocolo para la interoperabilidad del procesamiento de transacciones.

OSI/TP proporciona las funciones de /TDOMAIN además de la interoperabilidad de aplicaciones Tuxedo con el protocolo. Cumple con todos los requerimientos de comunicación con el sistema remoto, asegurando transparencia para los programadores de servicios.

Con una aplicación distribuida sólo se requiere un archivo de configuración *UBEfile* que levante los servicios de todos los lugares donde esté la aplicación, pudiendo levantar cada una con diferentes características. Podríamos decir, que uno sólo consulta, otro actualiza, etc.

El archivo *tuxconfig* se pone en cada máquina.

BDMCONFIG es el archivo de configuración del dominio (*/TDOMAIN*).

- Cada dominio tiene sólo una cola de peticiones, mientras que cada *Gateway*[8] tiene su propia cola de respuestas.

[7] Open System Interconnection
[8] Dispositivos para conectar segmentos de redes y redes a mainframes

Podríamos mencionar como los componentes principales del dominio a:

- El dominio del *Gateway*
- Servidor administrativo del Gateway
- Servidor administrativo del dominio
- BDMCONFIG. Archivo de configuración del dominio

El Servidor administrativo del Gateway permite el monitoreo y control en tiempo real de la interface de comunicaciones, mientras que el servidor administrativo del dominio complementa la función permitiendo la administración de la configuración y sus características.
Se selecciona un Gateway cuando se tienen que interconectar sistemas que se construyeron con base a diferentes arquitecturas de comunicación.

Interface de programación /TxRPC

Cumple con el estándar transaccional RPC [9] de X/OPEN y es compatible con otras plataformas. Constituye una interface simplificada de programación para aplicaciones Tuxedo. Se basa en una definición de especificaciones escrita en un lenguaje conocido como IDL (Interface Definition Languaje).
Podemos decir que Tuxedo constituye la tercera capa en el desarrollo de aplicaciones transaccionales, esto es

Capas	Tuxedo Atmi	
	RPC	aquí estan los tpcall's, etc. (primitivas de Tuxedo)
	Soquets	

[9] Remote Procedure Call

Tuxedo RPC

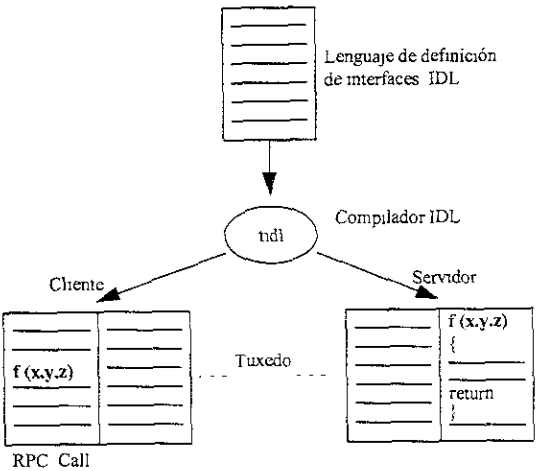

Fig. 26

Las propiedades de la interface simplificada de programación, pueden presentarse en tres grupos:

- Tuxedo Atmi
  - Control de transacciones distribuidas
  - Peticiones asíncronas
  - Forward
  - Manejo de conversaciones
  - Ruteo dependiente de datos
  - Encolamientos
- Tuxedo /TxRPC
  - Control administrativo de la aplicación
  - Control de la configuración
  - Balanceo de cargas
  - Encolamiento por prioridades
- RPC Genérico
  - Procesamiento distribuido
  - Petición / respuesta (esquema cliente - servidor)

Términos y definiciones para el manejo de colas en Tuxedo.

Se identifica como **/Q** . Proporciona un método de almacenamiento de mensajes o peticiones a servicios que garantizan su posterior procesamiento.

El almacenamiento en colas, puede ser :

- FIFO [10]
- LIFO [11]
- Basado en tiempo
- Basado en prioridades.

El Atmi incluye la función tpenqueue( ). la cual permite realizar el encolamiento de mensajes, tanto en el cliente como en el servidor. La función tpdequeue( ) realiza el desencolamiento de mensajes (extracción de un mensaje de la cola) para su atención correspondiente.

Tuxedo también proporciona un administrador de colas a través de qmadmin.

Un ejemplo del uso de colas puede ser el siguiente.

Cuando un nuevo cliente abre su cuenta, el Banco debe enviar un paquete de bienvenida al cliente a su domicilio, sin embargo, el Banco no desea generar una orden de trabajo para cada cuenta nueva, y no es deseable que esto sea parte de la transacción "apertura de cuenta". Cuando la transacción "apertura de cuenta" es exitosa, la aplicación almacena una solicitud para enviar este paquete el cliente posteriormente.

Al final de la semana, la solicitud es desencolada por el servidor a través de *tmqforward* y se genera una orden para enviar el paquete correspondiente.

De manera transaccional y paso a paso se tiene el siguiente esquema:

1. El cliente encola una requisición para el servicio "A" por medio de la función tpenqueue( )
2. El servidor TMQUEUE recibe la petición y la almacena en la cola del servicio "A"
3. Cuando el paso anterior es exitoso, el cliente recibe una respuesta positiva.
4. El servidor TMQFORWARD desencola la petición en el tiempo especificado.
5. TMQFORWARD emite la llamada al servicio "A" en modo transaccional.
6. El servicio "A" regresa a TMQFORWARD vía la función tpreturn.
7. TMQFORWARD coloca la respuesta en la cola REPLYQ

[10] First Input First Output - primero en entrar, primero en salir
[11] Last Input First Output - ultimo en entrar. primero en salir

8. El cliente emite la requisición de respuesta vía un tpqdequeue
9. TMQUEUE recibe la petición y elimina la respuesta de la cola de respuestas.
10. TMQUEUE envía la respuesta al cliente.

Como ya se ha comentado con anterioridad, existen diversas variables de ambiente para especificaciones a Tuxedo, en el caso de las colas, se usa la variable QMCONFIG para definir la ruta del archivo que contiene la cola (Queue Space) . De manera análoga, Tuxedo proporciona facilidades para la administración de colas a través de comandos de *qmadmin*.

En la sección *Servers* del archivo *Ubbconfig*, se especifican los servidores TMQUEUE y TMQFORWARD para el manejo de colas.

Adicionalmente, se requiere de la estructura TPQCTL, la cual es uno de los argumentos de las funciones tpenqueue y tpdequeue para pasar información en las llamadas y recibir información en las respuestas (retornos).

Diseño del Sistema

La solución a implementar adoptada, consiste de:

1. Un servidor tuxedo para cada aplicación general, esto es, un servidor tuxedo para VTA (Vía Telefónica Atlántico), otro para Migesa (caja y plataforma), otro para el SAI (Sistema de Administración de Inversiones), otro para el Sistema de Pago de Servicios, etc.
2. Un conjunto de API's [1] , cuya función es enlazar la aplicación solicitante del servicio cuando ésta no tiene una estructura que permita llamar directamente al servicio Tuxedo de esa aplicación. En el primer caso (requiere API) está precisamente VTA, cuya parte demandante inicial es Periphonics, mientras que en el segundo caso se encuentra el SAI.
3. Un Servidor General o Core Domain, que contiene un directorio de todos los servicios disponibles, el equipo donde se encuentran y la aplicación responsable de resolver cada servicio (requerimiento). Este servidor, es el que recibe los llamados de los servidores de cada aplicación.
4. Una interface en COMS [2] para los equipos Unisys, que recibe el llamado del servidor general para una aplicación específica y que lo convierte al formato que espera la aplicación en este ambiente particular de trabajo.
5. Por último, se implementó un ajuste a la aplicación particular, en este caso cheques, para identificar el origen de la transacción y considerar acciones particulares referentes a acceso, permisos y manejo de errores entre otros.

El esquema que se muestra a continuación, ilustra de mejor manera el proceso descrito en sus diferentes componentes para mayor claridad.

[1] Application Program Interfase
[2] Aplicación del sistema operativo MCP de los equipos Unisys que maneja las comunicaciones

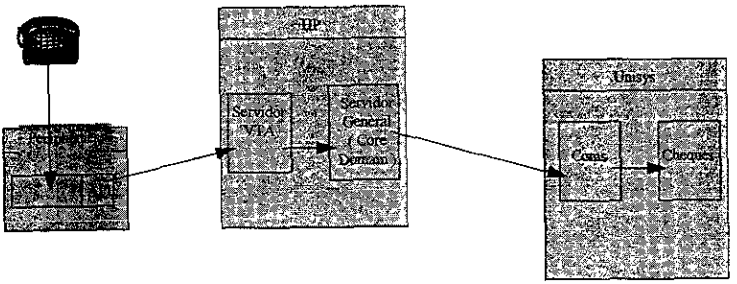

Fig. 27

La misma ruta seguida para la solicitud de un servicio. es seguida para la respuesta, aunque claro está, en sentido contrario.

Es importante aclarar, que en el caso de la interface que nos ocupa, dado que la aplicación de cheques no está en Base de Datos. algunas de las características de tuxedo como son el reverso de una transacción cuando ésta no es resuelta en los términos esperados o en alguna de sus partes, cuando se descompone en el llamado a varios servicios en alguno de los servidores, no puede ser manejada como un rollback [3] ,por lo que debe implementarse en la aplicación o en mismo servidor, un proceso que emule y controle esta operación. Desde luego, en nuestro caso - ejemplo que se refiere a una consulta, esta situación no se puede presentar, pero es importante mencionarlo, para que se dimensione el trabajo requerido para la implementación completa de esta solución.

Convenciones de identificación.

En buffers FML

El prefijo de tamaño uno en minúsculas, indica el tipo de dato. La información posterior en mayúsculas y con separadores "de subrayado" o guión bajo. identifica la descripción semántica del dato como variable mnemónica.

[3] declaración en Base de Datos para cancelar una transacción y deshacer cualquier cambio que haya ocurrido desde el inicio de la transacción

Tipo:

l	long
s	String
d	double

Ejemplo:

lID_CLIENTE	l tipo long
sCUENTA	s tipo string
dSALDO	d tipo double

En buffers tipo VIEW y programas escritos en C

Se continúa con la misma estructura, pero las mayúsculas se convierten a minúsculas y mayúsculas, de manera que las mayúsculas indiquen el inicio de una palabra o abreviatura que se refiera a una idea u objeto diferente y se suprimen los separadores de subrayado o guión bajo.

Ejemplo:

lIdCliente	l tipo long
sCuenta	s tipo string
dSaldo	d tipo double

Identificadores de variables y estructuras

Tipo	Prefijo	Sistema
Buffer FML a utilizarse como argumento de entrada a algún servicio	fbi	Tuxedo
Buffer FML a utilizarse como argumento de salida a algún servicio	fbo	Tuxedo
Buffer FML a utilizarse simultáneamente como argumento de entrada/salida en algún servicio	fb	Tuxedo
View a utilizarse como argumento de entrada a algún servicio. Esto se refiere tanto al identificador de definición – que el compilador **viewc** traduce en "structure tags" de C – como a los identificadores	vi	Tuxedo

que declaran instancias de la estructura.		
view a utilizarse como argumento de salida a algún servicio. Éste se refiere tanto a el identificador de definición como a los identificadores que declaran instancias de la estructura.	vo	Tuxedo
view a utilizarse simultáneamente como argumento de entrada/salida Esto se refiere tanto a el identificador de definición como a los identificadores que declaran instancias de la estructura.	v	Tuxedo
cadena terminada con nulo	s	Tuxedo
cadena de longitud fija (CARRAY)	ca	Tuxedo
short	sh	Tuxedo y C
int	i	Tuxedo y C
long	l	Tuxedo y C
float	f	Tuxedo y C
double	d	Tuxedo y C
dec_t (decimal codificado)	dec	Tuxedo e Informix
apuntador a algún otro tipo	p<prefijo correspondiente al tipo>	Tuxedo y C
serial	se	Informix
integer	i	Informix
smallint	si	Informix
char o character	ch	Informix y C
datetime	da	Informix
variables globales	G_<prefijo correspondiente al tipo>	Tuxedo, Informix y C

Identificadores de servicios.

Distinguiremos dos tipos de servicios, uno que agrupa servidores que apoyan las **aplicaciones** y otra que agrupa servicios asociados a **productos** del Banco.

Aplicaciones	**Acrónimos**
Vía Telefónica Atlántico	VTA
Vía Electrónica Atlántico	VEA
Sucursales	SU
Sistema de Administración de Inversiones	SAI

Productos	**Acrónimos**
Cheques	Chq
Valores	Val
Tarjeta de Acceso	Tja
Tarjeta de Débito	Tjd
Tarjeta de Crédito	Tjc
Inversiones a Plazo	Ipz

Los identificadores de servicios asociados a una aplicación se forman como sigue:

<acrónimo de la aplicación con mayúsculas> + <acrónimo del producto> + <descripción del servicio>

Los identificadores de servicios ofrecidos alrededor de un producto, se forman:

<acrónimo del producto en mayúsculas> + <descripción del servicio>

En ambos casos, la descripción del servicio se formará utilizando minúsculas y mayúsculas, de manera que las mayúsculas indiquen el inicio de una palabra o abreviatura que se refiere a un objeto o idea diferente.

Estos identificadores se deben limitar a 16 caracteres, ya que tuxedo sólo reconoce los primeros 16 para los identificadores de servicios.

Ejemplos:

VTAChqConSdo

Reglas para API's de Servicios.

Adicionalmente a lo expuesto referente a API's , y con el objeto de aislar a las aplicaciones del cliente de algunos cambios en la implantación de los servidores, así como facilitar la labor de programación, los servidores ofrecen una API como un conjunto de funciones que se utilizan entre aplicaciones cliente y servidores.

Identificadores en tipos definidos para API's de servicios.

De manera adicional a las reglas establecidas para identificadores de variables y estructuras, la implantación de API's aprovechan la facilidad del lenguaje C para definir tipos de datos mediante el estatuto **typedef**. Los nombres de tipos definidos en C empiezan con la letra **t** minúscula. De estos tipos se hace la siguiente especificación más fina para los tipos de datos que fungirán de manera uniforme como argumentos a un API.

Tipo	Prefijo	Comentario
Estructura para proporcionar datos de entrada al servicio que invoca el API	te	su nombre se obtiene de reemplazar con "te" el prefijo "vi" del "structure tag" que genera el compilador de *views* de Tuxedo
Estructura para proporcionar datos de salida al servicio que invoca el API	ts	su nombre se obtiene de reemplazar con "ts" el prefijo "vo" del "structure tag" que genera el compilador de *views* de Tuxedo
Estructura para proporcionar datos de entrada/salida al servicio que invoca el API	t	su nombre se obtiene de reemplazar con "t" el prefijo "v" del "structure tag" que ge-nera el compilador de *views* de Tuxedo
Instancia de estructura para proporcionar datos de entrada al servicio que invoca el API	ite	
Instancia de estructura para proporcionar datos de salida al servicio que invoca el API	its	
Instancia de estructura para proporcionar datos de entrada/salida al servicio que invoca el API	it	
Pointer a estructura con datos de	pte	

entrada al servicio que invoca el API
Pointer a estructura con datos de pts
salida al servicio que invoca el API
Pointer a estructura con datos de pt
entrada/salida al servicio que invoca
el API

Identificadores en API's de servicios.

Recordando las reglas expuestas, el identificador de un servicio hacia una aplicación se forma:
<acrónimo de la aplicación con **mayúsculas**> + <acrónimo del producto> + <descripción del servicio>
para su correspondiente nombre de función en el API se tiene la regla:
<acrónimo de la aplicación con **minúsculas**> + <acrónimo del producto> + <descripción del servicio>

Mientras que para identificadores de servicios ofrecidos alrededor de un producto se forman:
<acrónimo del producto en **mayúsculas**> + <descripción del servicio>
para su correspondiente nombre de función en el API se tiene la regla:
<acrónimo del producto en **minúsculas**> + <descripción del servicio>

Especificación genérica del API

Como todos los servicios Tuxedo se invocan con *tpcall* o *tpacall*, la función del API tiene los siguientes cinco argumentos en el orden expuesto:

1. El APPKEY de la aplicación. Es de tipo tAppkey y es un argumento obligatorio, es el mismo para todas las funciones y pasa por valor como argumento de entrada.
2. La banderas del servicio. Son de tipo tTXflags, tambien es un argumento obligatorio para todas las funciones y pasa por valor como argumento de entrada.
3. El apuntador a la estructura con datos de entrada. Necesariamente debe existir un tipo definido para esta estructura.
4. El apuntador a la estructura de datos de salida. Necesariamente debe existir un tipo definido para esta estructura.
5. Apuntador a la estructura con códigos de error de Tuxedo, esto es el apuntador a una instancia del tipo tsErrno que, al regresar la función del API, contiene los valores de Tuxedo

llamadas tperrno y tpurcode.

Adicionalmente. todas las funciones API son de tipo int y regresaran como valor el que le entregue el tpcall o el tpacall del servicio llamado.

Finalmente, existe la siguiente excepción que aplica a las reglas de los argumentos (3) y (4) : Cuando la función del API no requiera de uno de estos argumentos. el apuntador será de tipo tNull y el usuario de la función deberá pasar la constante NULL como su valor.
Ejemplo:
Supóngase que se tiene el servicio llamado CHQCargo y que toma como buffers argumento los views de nombre viChqCargo y voChqCargo. En este caso la implantación de la correspondiente función API es:

int ChqCargo(tAppkey iteAppkey, tTXflags iteTXflags, teChqCargo *pteChqCargo. tsChqCargo *ptsChqCargo, tsErrno *ptsErrno)

Abreviaturas usuales.
Las siguientes abreviaturas se deben utilizar en la formación de nombres de variables y servicios:

Abreviatura	Significado
Cte	cliente
Cto	contrato
Ctos	contratos
Cnn	concentración
Con	consulta
Cpra	compra
Dis	dispersión
Fch	fecha
Fnds	fondos
Fol	número de folio
Gral	general
Hr	hora
Intes	intereses

Mto	monto
Mvto	movimiento
Nom	nombre
Num	número
Id	identificador
Imte	importe
Prt	protección
Ref	número de referencia
Sec	número secuencial
Ser	servicio
Sdo	saldo
Tip	tipo
Tit	titular
Trf	transferencia
Vnta	venta
Pzo	plazo

A continuación se presenta el API utilizado para el llamado desde periphonics al servidor VTA identificada como ChqConSdoTodo.C

Objetivo: Llamar al servicio VTAChqConSdoTodo del servidor VTA

Llamado:

```
Int VtaChqConSdoTodo(tXflags, teVtaChqConSdoTodo*,
                     tsVtaChqConSdoTodo*)
```

Datos de entrada:

(tTXflags)0

y datos en la estructura apuntada por el apuntador a

```
typedef struct {
        long        lCtoVta;
        short       shVpsId;
        tCuenta     chCuenta;
```

```
        tMoneda     shMoneda;
    } teVtaChqConSdoTodo;
```

Datos de salida en la estructura apuntada por un apuntador a

```
typedef struct {
        tTitular chTitular;
        tImporte    decSdoTotal;
        tImporte    decSdoInicial;
        tImporte    decSdoSBCHoy;
        tImporte    decSdoRemesasTránsito;
        tImporte    decDevolBimestre;
        tImporte    decSdoPromMes;
        tImporte    decSdoLineaCredito;
        tImporte    decInteresesDevengados;
    } tsVtaChqConSdoTodo,
```

Aclaraciones adicionales.

A la entrada, **1CtoVta** debe contener el contrato único del cliente de VTA. **shVpsId**, el identificador del VPS en que se ejecuta esta función. **chCuenta** el número de la cuenta sobre la que se desea hacer la consulta de saldo. **shMoneda** el tipo de moneda de la cuenta.

A la salida, si la llamada al servicio VTAChqConSdo fué exitosa, se entregará como valor da la función EXITO, y en la estructura de salida. los 9 campos que se indican a continuación contendrán la información que Periphonics utilizará para entregar al usuario la respuesta correspondiente. De lo contrario, la función devolverá alguno de los valores de error mostrados más adelante.

A la entrada, el campo	contiene
1CtoVta	Contrato único del cliente VTA
shVpsId	Identificador del VPS en que se ejecuta esta función
ChCuenta	Número de cuenta de cheques de la que se requiere el saldo
shMoneda	El tipo de moneda de la cuenta

A la salida, el campo	contiene
sTitular	Nombre del titular de la cuenta
decSdoTotal	Saldo disponible
decSdoInicial	Saldo de mayor o real
decSdoSBCHoy	Saldo salvo buen cobro hoy
decSdoRemesasTransito	Importe de las remesas en tránsito
decDevolBimestre	Importe de devoluciones del mes
decSdoPromMes	Saldo promedio del mes (al día de hoy)
decSdoLineaCredito	Crédito en cuenta corriente
decInteresesDevengados	Rendimiento del mes anterior

Valores de la función:

EXITO	Requerimiento resuelto satisfactoriamente
CTA_NO_EXISTE	Cuenta inexistente
CTA_BLOQ_INTER	La cuenta está bloqueada o intervenida
SRV_NO_DISP	Servicio no disponible
FRACASO	Error no tipificado

Es conveniente indicar, que la información identificada anteriormente, corresponde a la información susceptible de recibir del punto de entrada (teléfono en este caso), pero la aplicación cheques, requiere información adicional que la función debe proporcionar, y que se refiere al área solicitante o Centro de Costos (Banca Electrónica - Vía Telefónica Atlántico), Departamento, Sección y Nivel.

De manera complementaria, el servidor efectúa antes de enviar el requerimiento a la aplicación, una validación de acceso e identificación del origen de la solicitud, y puede devolver la requisición si así lo identifica.

Enseguida, procedemos a especificar el servicio utilizado en el servidor de VTA identificado como VTAChqConSdoTodo.ec

Objetivo: Servicio que realiza la consulta de saldo de cuenta de cheques del Banco del Atlántico.

Llamado: tpcall

Datos de entrada: view viChqConGral

long	lCtoVta
short	shVpsId
string	sCta[Len_cuenta + 1]
short	shMoneda

Datos de salida: view voChqConGral

string	sTitular[Len_Titular+1]
dec_t	decSdoTotal
dec_t	decSdoInicial
dec_t	decSdoSvoBuCo
dec_t	decImpRemesas
dec_t	decImpDevoluciones
dec_t	decSdoPromedio
dec_t	decSdoDispCred
dec_t	decImpIntereses

A la entrada, el campo	contiene
lCtoVta	Contrato único del cliente VTA
shVpsId	Identificador del VPS en que se ejecuta esta requisición
shCuenta	Sufijo del número de cuenta de cheques de la que se requiere el saldo
shMoneda	El tipo de moneda de la cuenta

A la salida, el campo	contiene
sTitular	Nombre del titular de la cuenta
decSdoTotal	Saldo disponible
decSdoInicial	Saldo de mayor o real
decSdoSvoBuCo	Saldo salvo buen cobro hoy
decImpRemesas	Importe de las remesas en tránsito

decImpDevoluciones	Importe de devoluciones del mes
decSdoPromedio	Saldo promedio del mes (al día de hoy)
decSdoDispCred	Crédito en cuenta corriente
decImpIntereses	Rendimiento del mes anterior

Valores de la función:

EXITO	Requerimiento resuelto satisfactoriamente
CTA_NO_EXISTE	Cuenta inexistente
CTA_BLOQ_INTER	La cuenta esta bloqueada o intervenida
SRV_NO_DISP	Servicio no disponible
FRACASO	Error no tipificado

Lo hasta aquí expuesto, sin duda da una buena idea de la solución implementada para el problema planteado, el entorno construido para tal efecto, y las reglas definidas para su construcción e implementación. Profundizar más al respecto, implica la necesidad de entrar al código fuente da cada una de las partes que componen el proyecto, lo cual involucra como se ha expuesto varias plataformas y varios entornos de programación. Aquí se ha tratado de mostrar la arquitectura con la mayor precisión sin entrar hasta donde ha sido posible al código.

Conclusiones

La situación expuesta, además de un reto, es una oportunidad que ofrece el entorno para los que dedicamos nuestros esfuerzos a través del trabajo diario al desarrollo de aplicaciones locales dentro de una gran empresa.

Las áreas de sistemas dentro del sector bancario, cada vez son más agobiadas ante la tendencia mundial de utilizar aplicaciones elaboradas por entidades especializadas. El Banco del Atlántico no es la excepción, aquí también se ha iniciado la ardua tarea de sustituir las aplicaciones existentes por un grupo ya probado y utilizado por varias instituciones bancarias en México y en otros paises. ¿ Por qué entonces, la necesidad de implementar la solución planteada en el presente trabajo ?, si ya se inicio la sustitución de las mismas. La razón es un tanto curiosa, pero válida, es decir, además de permitir la continuidad de las operaciones con los sistemas actuales, como ya se ha mencionado, también permite la sustitución paulatina de los mismos. Esto es, estamos colaborando para su propia eliminación. Esto constituye sin duda, un reto aún más difícil, pero no sólo eso, sino que permite a su vez la adopción del "Core" o cuerpo central del nuevo sistema y la de interfaces con el usuario más amigables y acordes a la tecnología vigente y futura. Hay que entender que la informática se encuentra en un proceso de cambio continuo y acelerado y que las aplicaciones y las instituciones deben ser capaces de incorporarlo, so pena de quedarse fuera del entorno Bancario mundial y posiblemente del negocio.

La solución adoptada se encuentra en posición tal, que permite ambas cosas, y no sólo eso, además ayudará a la incorporación del personal actual al nuevo entorno con la sustitución paulatina de operaciones, aprovechando su experiencia y capacitándolo en las nuevas herramientas y plataformas. ¿ Qué hay más valioso en una institución, que sus recursos humanos? ¿ Cómo incorporarlos al nuevo esquema, si estan agobiados por el trabajo diario ?.

Por lo anterior, es indudable que la solución elegida, va más allá de un esquema local de

desarrollo. más allá de la sustitución de aplicaciones, más allá de la incorporación de nuevos equipos, pone al Banco en la antesala de alcanzar el entorno tecnológico que la competencia local y externa demanda , para que continúe con éxito su desarrollo tal como lo ha hecho desde hace muchos años, pero con la visión de un futuro más promisorio y lo más importante, con la preparación técnica y humana que el mundo cambiante requiere.

Bibliografía

- The Tuxedo System
  Software for Constructing & Managing
  Juan Andrade, Mark T. Carges
  Addison Wesley

- Tuxedo System 6 Course
  For Application Administrators
  BEA Systems, Inc.

- Tuxedo System 6 Course
  For Application Developers
  BEA Systems, Inc.

- Tuxedo System 6 Course
  Architecture
  BEA Systems, Inc.

- 3-Tier Client/Server at Work
  Jeri Edwards
  John Wiley & Sons, Inc.