11
2ej

U.N.A.M.

FACULTAD DE INGENIERIA

APLICACION CLIENTE/SERVIDOR DE 3-CAPAS

TESIS

QUE PARA OBTENER EL TITULO DE:
INGENIERO EN COMPUTACION

PRESENTA:
LUIS ALFONSO AMEZCUA ARAGON

DIRECTOR DE TESIS: ING. ADRIAN ALVAREZ MARTINEZ
CO-DIRECTOR DE TESIS: ING. LAURA SANDOVAL MONTAÑO

MEXICO, D. F. 1996

TESIS

COMPLETA

Dedico este trabajo a ...

Mi abuela.	Estoy seguro que si estuviera todavia aqui con nosotros, estaria muy orgullosa de este logro.
Mi madre.	Gracias a la educación que me dió, he podido lograr todo lo que soy.
Mi hermana.	Se que siempre he contado con su respaldo.
Mi padre.	Un reconocimiento por su ayuda.
Mis tios, tias, primos y primas.	Por lo mucho que ha representado el apoyo que me han dado cuando asi lo he necesitado.
Adrian Alvarez M.	Porque me hizo depositario de la visión de este trabajo proporcionándome todos los medios para que lo pudiera llevar a cabo tan exitosamente.
Laura Sandoval M.	Indudablemente sin su empuje, animo y dedicación desinteresada en el presente trabajo, el finalizar el misma hubiera sido más dificil.

Índice.

Introducción.

Las tecnologías de los sistemas de información de hoy en día deben de responder a una serie de retos. Cambio Rápido.- La globalización y los rápidos cambios en los ambientes de negocios actuales incrementan la demanda en organizacianes a responder más rápida y específicamente tanto a las peticiones de usuarios como a la competencia. Para ésto se necesita un entendimiento hacia adelante del estado del negocio. Se necesita acceso inmediato a la información que nos diga qué productos se están movienda, las demandas recientes del mercado, que tan bien se están alcanzanda las metas de calidad, etc. Este clase de respuesta rápida significa que la tecnologia de información debe de modelar el proceso del negocio. Ambientes Complejos.- Otro reto es cómo ocuparse de la complejidad de los ambientes de la tecnología de información. Los negocios se están moviendo de ambientes centralizados basados en un sólo equipo ("host") a complejos ambientes multi-vendedor heterogéneos de cámputo distribuido llamados ambientes cliente/servidor de alcance empresarial. Aquí las operaciones abarcan departamentos, regiones, e inclusive limites internacionales. Uno de los mayores retos es unir estos diversos ambientes manteniendo un alto desempeño. Aplicaciones Complejos.- Los negocios están pidiendo que las aplicaciones hagan más, de manera que coardinen el flujo completo de un negocio en lugar de una sola operación. Esta propuesta está relacionado con la reingeniería de procesos del negocio donde un negocio es visto como una serie de procesos de negocio interrelacionados, en lugar de departamentos independientes con areas de responsabilidad asignadas. Para entender un negocio desde esta perspectiva consolidada se requiere información que abarque múltiples departamentos y sistemas. Esto significa que se necesitan aplicaciones de alcance empresarial que implementen el proceso completo del negocio. Crecimiento Evolutivo Manejado.- Según los negocios se extienden y actualizan sus sistemas de información, deben de ser capaces de apalancar su inversión en tecnología de información y en recursos humanos. La integración con sistemas existentes es también un factor clave, mientras se cumplan las demandas de desempeño y confiabilidad de un negocio en expansión. El nuevo sistema de alcance empresarial se debe de adecuar al ambiente existente en lugar de substituirlo.

El presente trabajo de tesis es una muestra de implementación de un sistema el cual da una respuesta a los retos que se acaban de mencionar. La aplicación prototipo desarrollada es una obra de Reingeniería de Software la cual se basa en una Arquitectura Cliente-Servidor de 3 Capas expuesta por John J. Donovan. La implementación de la misma se basa en el Ambiente de Desarrollo de Aplicaciones Avanzado Forté.

En México se podria afirmar que hasta la fecha de realización de este trabajo, las oportunidades que se vislumbran en el campo de acción de este ámbito de la computación son muy prometedoras, ya que éste es un campo prácticamente virgen. Por esta razón, este trabajo pretende ser un punto de partida que sirva como un valioso antecedente para futuros desarrollos similares.

El presente trabajo de tesis está organizada en 5 capítulos y 2 apéndices. El capítulo 1 empieza planteando un panorama histórico de los pasos que se dieron para dar origen al modelo Cliente/Servidor. Partiendo de ésto se habla del modelo Cliente/Servidor haciéndose mención de las circunstancias que se presentaron alrededor de su surgimiento. Enseguida se exponen las concepciones del modelo Cliente/Servidor visto desde 3 diferentes puntos de vista, el de John J. Donovan, el de W. H. Inmon y el de David Vaskevitch. Para finalizar este capítulo se hace un estudio de evaluación de las 3 estrategias definiéndose la que sirve de fundamento teórico de la aplicación en desarrollo.

En el capítulo 2, debido al estrecho enlace existente entre el modelo Cliente/Servidor y las Bases de Datos, se proporcionan de inicio algunos conceptos básicos de Bases de Datos para citarse después el entorno histórico en el cual evolucionaron en un principio. Posteriormente se expone lo que está detrás de una Base de Datos pero considerándolo desde un punto de vista reciente.

El capítulo 3 contiene otro aspecto también fundamental en un sistema Cliente/Servidor, las Redes. En la primera parte de este capítulo se abordan las redes locales dándose los elementos principales de ellas. En la segunda parte se habla de las características principales de las redes de área amplia.

El capítulo 4 es la parte práctica de este trabajo. En primer lugar se expone el Análisis de Requerimientos necesario en el desarrollo de un sistema. Este se realizó con base en lo señalado por la Metodología de Programas de Digital. A continuación se da un breve paseo teórico por los conceptos que giran alrededor del diseño lógico de un sistema haciéndose énfasis como es natural en la Arquitectura Cliente/Servidor de 3-Capas. Por último se toca lo referente a la implementación física del sistema planteándose la relación con el producto de software usado para el desarrollo del mismo, el cual es Forté como ya se mencionó. Antes de ésto se habla del producto dándose sus características más importantes.

El capítulo 5 presenta las conclusiones de este trabajo de tesis.

El apéndice A presenta los diagramas de las dos diferentes configuraciones usadas en el desarrollo así como en la ya propia ejecución del sistema.

El apéndice B muestra parte del código más importante generado en la implementación de la aplicación.

1. Análisis de las Metodologías de diversas Estrategias Cliente/Servidor.

El término cliente/servidor fue manejado en los 70s en el Centro de Investigación de Palo Alto de Xerox. La aplicación cliente/servidor más temprana fue la impresión láser. La segunda aplicación cliente/servidor fue la conmutación de terminales sobre LANs. La tercera aplicación cliente/servidor fue el servicio de archivos. Hoy el término cliente/servidor se refiere básicamente a servidores de Bases de Datos y a las máquinas de procesamiento de aplicaciones y de interfase de usuario que las comparten. Este es el cliente/servidor que está gradualmente transformando –sino es que rápidamente eliminando– *mainframes* y minicomputadoras y que está causando grandes problemas en la industria computacional.

La revolución "downsizing" pide reemplazar el *mainframe* con cientos de sistemas más pequeños, todos hablándose los unos a los otros, y cada uno sirviendo las necesidades de equipos locales e individuales. El resultado son sistemas de computo distribuidos que soportan la toma de decisiones descentralizada y son manejados por empleados habilitados que se enfocan en la calidad de los productos y en la respuesta a las necesidades de los usuarios. En los próximos diez años, las computadoras cambiarán finalmente la manera fundamental en que la gente organiza y lleva a cabo los negocios de todos tamaños.

El mundo está cambiando alrededor de nosotros. La Reingeniería de Procesos de Negocios (BPR) y la competencia global están forzando a las compañías a trabajar en nuevas formas. Entender estos cambios es un requerimiento para estar en control personalmente y profesionalmente– durante los años venideros.

1.1 Orígenes del Modelo Cliente/Servidor.

1.1.1 Antecedentes del Modelo Cliente/Servidor.

El alboroto en la industria tiene profundas implicaciones no sólo para las compañías que construyen computadoras, sino también para las muchas organizaciones que las usan. BPR es la herramienta de hoy en día para determinar cómo las gerentes deben estructurar sus organizaciones.

1.1.1.1 Crisis de Media Vida para una Industria.

Ninguna de las tecnologías de reemplazo está lo suficientemente madura para verdaderamente reemplazar los *mainframes*. Hasta hace poco el papel de la computadora en el mundo era fácil de entender. Los *mainframes* daban soluciones de negocios. Tanto los *mainframes* como las PCs eran computadoras, pero fuera de ello, no estaban relacianadas. Una de las escasas constantes de las últimas décadas fue el progreso constante en el mundo de la computación. La tecnología computacional se ha convertido en uno de los costos más grandes en las organizaciones. Mientras BPR le vuelve a dar forma a la manera en que las compañías trabajan, las organizaciones deben invertir en todavía más tecnología de información.

Las computadoras entre los años 1900-1949 fueron construidas para propósitos específicos y estaban muy limitadas en funcionalidad.

En los 50s los lenguajes de programación no habían sido inventados, así que hasta simples aplicaciones tomaban mucho tiempo en desarrollarse. Durante esta completa década, sólo algunas docenas de máquinas fueron vendidas, cada una de las cuales costando cientos de miles de dólares. La función primaria de las computadoras era todavía los cálculos. Las computadaras, antes misteriosas, repentinamente parecieron entendibles. Al final de esta década éstas fascinaron al público.

En los 60s, las computadoras de verdad aprendieron a caminar. La gente empezó a ver las computadoras como inteligencias artificiales que a lo mejor algún día reemplazarían a la gente en todos los tipos de tareas. Las computadoras se convirtieron en importantes máquinas de negocios. Llevando una gran cantidad de este optimismo estaba una nueva forma de ver a las computadoras. Sin embargo esta década también marcó el primer gran período de desilusión con las computadoras. Al final de esta década un nuevo papel estaba claro: la computadora era una herramienta suprema para automatizar procesos de negocios complejos. Los 60s también vieron el desarrollo de sistemas operativos modernos. Estos sistemas operativos le permitieron a la computadora ser un recurso compartido constantemente disponible. Repentinamente las computadoras se convirtieron en el gran negocio. En esta década las computadoras preservaron los negocios. Más que cambiar los procesos existentes, las computadoras los hicieron más rápidos; jugaron un papel importante en el mantener a esas orgonizaciones funcionales según crecieron. Al final de los 60s, las computadoras regimente se habían vuelto indispensables para cualquier organización gronde. Las compañías estaban de repente gastando millones de dólares en lo que era antes una categoría de presupuesto no existente.

El mundo completo de la computación cambió entre 1968 y 1972. Las Bases de Datos, las terminales, las redes, y el almacenamiento permanente (discos) se volvieron prácticos todos casi al mismo tiempo. En los tempranos 70s los conceptos asociadas con Bases de Datos en-línea -Bases de Datos conectadas y alimentadas por terminales en

red– repentinamente estaban siendo descritas en revistas de negocios normalmente leídas sólo por ejecutivos. Esta década generó una visión para el futuro: la información está disponible instantáneamente, está siempre actualizada, está organizada en una base de datos consistente que le permite a una compañía operar más rápidamente, y le permite a todos en la organización estar en contacto con el mundo real.

Para el final de los 80s, las computadoras personales habían cambiado totalmente las expectativas de los usuarios y habían creado una demanda para una clase completamente nueva de aplicaciones y de sistemas de computadoras. Para 1989, aún los *mainframes* tuvieron que admitir que las computadoras personales estaban aquí para quedarse, y al mismo tiempo, los fanáticos de los PCs se dieron cuenta que esas mismas PCs necesitaban crecer y jugar un papel en el manejar del negocio. Así mismo esta década fue un periodo de profunda contradicción. Hubo una continua fermentación y cambio en la gran industria computacional. La tecnología de computadoras grandes al final de los 80s era muy similar a la del principio de la misma década. Las terminales se convirtieron comodidades que costaban algunos cientos de dólares. El modelo de base de datos relacional se volvió dominante. Por 1980, la industria computacional estaba empezando a madurar. El foco y la energía de la gran industria computacional cambió de las computadoras y Bases de Datos al proceso de diseño y construcción de aplicaciones. Así que los 80s fue un periodo en que los negocios se enfocaron en convertir el desarrollo de aplicaciones de un arte a una disciplina ingenieril. Junto con el desarrollo de metodologías vinieron las herramientas de software para soportar tanto las metodologías así como el proceso de diseño: CASE (Computer–Aided Software Engineering). La gran industria de la computación alcanzó madurez al final de los 80s. Así mismo esta década fue la de la computadora personal (PC). Mientras que se encargaban de necesidades especiales, aplicaciones como DBASE y WORDSTAR eran dramáticamente más fáciles de aprender y usar que las correspondientes aplicaciones de *mainframe*. Alrededor de 1985 los profesionales de la computación empezaron a tomar las PCs seriamente por primera vez. No obstante éstas no pudieron ser usadas para construir aplicaciones de negocios importantes, de ninguna manera. Una nueva clase de profesionales de la computación emergió en los 80s: los "gurus" cliente/servidor. El presidente del club de computación de ayer, se convirtió en el administrador de la red computacional de hoy.

En los 1990s nace el concepto de automatización de oficina. Hoy la automatización de la oficina está usualmente asociado con sueños de oficinas sin papeles. Las computadoras personales ayudaron e hirieron la automatización de la oficina. Ayudaron porque hicieron a las computadoras fáciles de usar realmente baratas. La hirieron porque demostraron que la meta de eliminar papel, simplificar procesos administrativos y cambiar la oficina, toma mucho más que sólo tecnología. Las computadoras personales cumplen con las necesidades del individuo, no con las de la organización; producen beneficios personales, pero no en formas que generen un devolución medible. Con las computadoras personales, la simplicidad con el trabajo se salió por la puerta. Los trabajadores de información se encontraron haciendo más trabajo en más tiempo. El problema con la

mayoría del trabajo hecho con computadoras personales es que la mayoría de este trabajo sólo está indirectamente relacionado a las funciones principales que mantienen al negocio funcionando. No sólo las PCs no han proporcionado un retorno en la inversión de capital, peor que eso, han hecho que los usuarios no quieran usar las otras computadoras que sí proporcionan un regreso de dinero. Durante los 90s, las organizaciones deben adoptar una completamente nueva perspectiva en el uso de computadoras. Las PCs ahora cuestan tanto como los *mainframes*, pero sólo sirven necesidades personales. Desafortunadamente, las grandes y sofisticadas aplicaciones basadas en *mainframes* no pueden ser convertidas para correr en computadoras personales. Las arquitecturas de los *mainframes* y de las PCs son muy diferentes. Se dice que las PCs deben de ser reservadas para aplicaciones más pequeñas. No obstante los sistemas de PC son mucho más poderosos de lo que se reconoce. BPR realmente depende en nueva tecnología de información. De hecho el punto central de la reingeniería es usar computadoras para rediseñar la organización. La Reingeniería producirá que un sistema de computadora distribuido combine las fortalezas de los *mainframes* y de las computadoras personales. Propiamente reescritas, las aplicaciones que se requieren para mantener a muy grandes organizaciones pueden funcionar en computadoras personales en red. Los sistemas resultantes tienen todo el performance, desempeño y sofisticación requerido en un ambiente de negocios real. Las aplicaciones son escritas en una completamente nueva forma. Pero una vez escritas, exhiben flexibilidad, escalabilidad y facilidad rara en el gran mundo de las aplicaciones de hoy en día.

1.1.2 Surgimiento del Modelo Cliente/Servidor.

1.1.2.1 La Revolución de Negocios de los 90s.

BPR es una extensión directa del Manejo de Calidad Total. ¿De qué tratan éstos? Las organizaciones y sus equipos de manejo están trabajando en un cambio conceptual fundamental que involucra pensar sobre el trabajo en términos de procesos en lugar de tareas. La fábrica completa debe ser considerada como un sistema. Un sistema es una colección de partes interconectadas, todas unidas por un proceso que se pueda describir. El proceso es la clave. La clave para una mejor producción es el mejoramiento del proceso, y la clave para el mejoramiento del proceso es entendimiento y medición.

La organización típica se puede ver como una gran pirámide con el equipo ejecutivo ocupando el pequeño pico y los empleados de la línea frontal (de ensamble) viviendo en la ancha base. Todos los pasos en el proceso de manipular una orden ocurren en la parte media de esta pirámide; ésta es la tan llamada capa de manejo medio. La manera en que una gran organización arregla todas las reglas de negocios de manera que puedan ser

confiables es creando una larga y sofisticada capa de manejo medio.

La respuesta para construir mejores partes y mejores productos contempla dos aspectos: – no esperar hasta el final – enfocarse en el proceso en lugar de la tarea.

En el centro del movimiento de Manejo de Calidad Total (TQM) está la idea de que los errores no suceden al final de una línea de producción. Se necesitan partes que nunca tengan problemas, algo que un robot no puede producir. TQM afirma que se realicen los productos perfectamente desde el principio. Se deben de hacer a los productores de las partes responsables de que las partes que fabrican sean perfectas antes de que formen parte del producto final. La Calidad significa a su vez consistencia de resultados. Para esto hay una opción: máquinas y procesos que se auto–regulen. Esto es, la información que se deriva como parte del proceso de manufactura se retroalimenta al proceso –en el momento que es generada– para mantener así al proceso auto–regulado.

El cambio más grande generado por TQM es el cambio al rol del trabajador de producción clásico. Nació la necesidad de eliminar la distinción entre gerentes y trabajadores. Esta división no produce productos que sean suficientemente buenas, ni en el mejor de los casos.

Hace veinte años los automóviles nuevos tenían cuando menos una parte que no funcionaba correctamente; ahora simplemente asumimos perfección. Este profundo cambio en la calidad del producto así como en la percepción de esa calidad por los clientes gira completamente alrededor de estos procesos auto–regulados. La auto–regulación significa que los instrumentos del proceso pueden modificar el proceso mientras se está ejecutando. La modificación del proceso requiere pensar. Para efectuar esta auto–regulación es necesario que: –los trabajadores de la producción sean responsables de todas las decisiones básicas que involucren el trabajo que realizan, –toda la información que un trabajador de producción necesita para producir productos perfectos, esté disponible para ese trabajador en cualquier momento. Todas las decisiones relacionadas a la calidad del producto de trabajo deben ser hechas por la gente que está realizando el trabajo al momento en que éste se realiza.

Para que esto sea posible se requiere de dos cambios fundamentales en la organización, uno mecánico y el otro filosófico. La primera que se debe hacer para ello es rediseñar el proceso de producción fundamental y todos los trabajos en él. Una consecuencia básica de este cambio es el enfoque de equipo en lugar del individual. El cambio filosófico involucra el convertir a los trabajadores que históricamente han sido robots de la línea de ensamblado en dueños pensantes del proceso auto–regulado. Este también es un cambio cultural.

Cuando esta misma forma de pensar es aplicada a los proceso de información y de oficina, los resultados son sorprendentes. Los procedimientos de oficina, así como los de

manufactura, históricamente han sido diseñadas alrededor de la filosofía de la línea de ensamble. BPR involucra aplicar el mismo ojo crítico que trabaja en la fábrica a los procedimientos y procesos que se encuentran en el ambiente de oficina. Así como en la fábrica, el cambio es fundamental y simultáneamente mecánico y filosófico.

Aplicada tanto a la fábrica como a la oficina, TQM Y BPR representan un nuevo modelo para pensar acerca de las compañías y la gente que trabaja para ellas. Este modelo mueve la responsabilidad al fondo de la organización. El resultado es organizaciones más planas, eficiencia incrementada, mejor calidad y costos reducidos. La burocracia, factor de baja desempeño, normalmente asociada con tamaño y jerarquía, está desapareciendo.

1.1.2.2 Revolución de Negocios, Revolución Técnica: La Oficina del Futuro Cliente/Servidor.

Existen dos revoluciones que se están extendiendo a través del mundo de negocios que son directamente dependientes una de la otra: 1. La revolución cliente/servidor de computadoras personales está cambiando la definición de computadoras. El uso efectivo de la tecnología cliente/servidor requiere un nuevo estilo de organización –esto es, una nueva forma de hacer negocios. 2. BPR está redefiniendo la manera en que las organizaciones funcionan. Está redefiniendo los trabajos, las responsabilidades, las misiones y las amplias culturas corporativas.

En un lugar de trabajo, el punto central del movimiento hacia el auto-manejo es que los individuos quienes controlan sus propias vidas se sientan mejor y produzcan mejores resultados. La expresión de libertad personal a través de la elección de herramientas de computadoras personales ha hecho el crecimiento del presupuesto de PCs completamente inevitable para las organizaciones de todos los tamaños.

Un cliente es un tipo de consumidor, un consumidor que usa servicios. En el lenguaje cliente/servidor, es obvio que la revolución computacional de los 90s es acerca del cliente. El cliente es sin lugar a dudas el mayor conductor en esta revolución. Está mostrando un futuro que es dramáticamente diferente que el mundo computacional del pasado: un ambiente que promueve la elección personal, la libertad personal, y aplicaciones que son cada vez más fáciles de entender.

La PC es capaz de engañar un *mainframe* y actuar como una terminal. Un "raspador de pantalla" (screen scraper) es una pieza de software que se encuentra dentro de la PC la cual intercepta las formas que se tratan de mostrar al usuario y las pone disponibles a conversión. A primera vista una aplicación propiamente rediseñada de esta forma se ve totalmente diferente. Un programador puede hacer que una aplicación de *mainframe* se vea como que usa una interfase gráfica totalmente moderna. Las apariencias gráficas están diseñadas para engañar a los usuarios. Hacen que las viejas aplicaciones se

vean como nuevas. En el proceso, traen algunos beneficios en términos de simplificación y facilidad de uso. Sin embargo, un cambio realmente significativo requiere de más.

La gente quiere que las computadoras sean caminos a la información localizada en todo el mundo; quieren combinar datos de muchas fuentes rápida y fácilmente. La mayoría de los usuarios no tienen problemas en la manipulación o presentación de la información de una manera efectiva. Sólo quieren un mejor acceso a esa información.

La industria computacional está seguramente enterada de que los usuarios tienen la necesidad de acceder los datos de las Bases de Datos de la corporación. Como éstas generalmente se encuentran en *mainframes*, las consultas que se realizan a las Bases de Datos traen a los *mainframes* a sus rodillas ya que en la mayoría de los casos se requiere acceder grandes partes de éllas. Cuando esto pasa, el negocio literalmente deja de funcionar.

Las PCs se han vuelto casi tan poderosas como los *mainframes*. Una PC compartida con una copia de la base de datos del *mainframe* puede contestar a consultas tan complejas tan fácilmente como un *mainframe*. Cada grupo de trabajo puede tener su propio servidor. Los servidores dan acceso a datos compartidos a varias computadoras de escritorio (desktop), todas conectadas en una red. El uso de servidores proporcionan una manera de hacer el sistema mejor. Aún sin una conexión directa al sistema real, el nuevo sistema de servidores, permitiéndole realmente a sus usuarios hacer preguntas, da la facultad de resolver problemas de clientes. Obviamente, este nuevo sistema depende totalmente en tener muchos servidores baratos. Así se crea un proceso auto-regulado. Este proceso no podría existir sin la tecnología servidor basada en PCs.

El servidor es la primera computadora que puede ser propiedad de tanto el grupo de trabajo así como de la corporación. Se puede programar que el servidor haga cumplir las reglas del negocio. Esto se puede construir de manera que cuando un equipo de trabajo escriba sus propias aplicaciones, cualquier cambio a la base de datos que rompa con las reglas, sea rechazado.

Pero, ¿De qué tratan el cliente/servidor? Tratan primariamente acerca del servidor: 1. Distribución de datos. 2. Distribución de procesos. 3. Interfase de usuario gráfica. El servidor permite que los datos estén distribuidos a través de muchas computadoras y otros servidores para que los grupos auto-manejados y los empleados autorizados puedan hacer preguntas frecuentemente y de maneras complejas. El servidor también permite que el procesamiento sea distribuido hacia los equipos para que puedan personalizar aplicaciones para que cumplan sus necesidades particulares mientras se está salvaguardando a su vez las reglas de negocios de la compañía. El cliente/servidor trata también del cliente. El tener una interfase de usuario gráfica es importante. Los *mainframes* con apariencia gráfica engañan a la gente por un corto periodo de tiempo, pero a la larga, no conducen a ningún negocio o revolución organizacional. El darle a equipos y a individuos un acceso sin límites

a los datos y la habilidad de personalizar procedimientos de negocios para cumplir con sus necesidades locales realmente guía a la revolución.

Como se acaba de mencionar, las compañías necesitan proporcionar a los equipos y departamentos individuales la habilidad para acceder datos de la corporación permitiéndoles cambiar algunos de esos datos para cumplir con sus necesidades particulares. Los servidores con Bases de Datos distribuidas son la clave para que esto se pueda llevar a cabo.

Al pensar sobre la revolución computacional de los 90s, la mayoría de los pronosticadores asumen que los clientes y los servidores reemplazarán a las grandes computadoras existentes. Los servidores del mundo serán los cerebros y los sistemas nerviosos de las organizacianes a su alrededor. Sirvientes de sus equipos locales auto-manejados, estos servidores ayudarán a crear un mundo de procesos auto-regulados y de empleados con poder de decisión.

1.1.2.3 GUI, UI: ¿Pueden ser las computadoras útiles?

El reto aquí es hacer a las computadoras amigables y fáciles de usar como cualquier otra aplicación, y la forma de lograr eso es enfocarse a la interfase de usuario (UI).

Las computadoras todavía tienen un largo camino a recorrer. Las PCs se han vuelto útiles de una forma no antes pensada sino hasta hace recientemente.

La razón para hacer la interfase gráfica es proporcionar un mecanismo poderoso para que los usuarios y las computadoras puedan hablar entre ellos. Las GUIs (Graphic User Interfaces) son entonces una particular poderosa clase de UI. La UI es la frontera entre la computadora y una persona trabajando con la computadora para llevar a cabo un conjunto de tareas. La UI es también un dispositivo o sistema.

Las computadoras se han vuelto más rápidas día a día. Otros dos cambios han ocurrido los cuales han revalucionado la percepción de las computadoras y de lo que pueden hacer: 1. Las interfases de usuario gráficas estándar. 2. Los programadores que construyen aplicaciones han desarrollado un entendimiento de cómo esos elementos de la GUI pueden ser usados para desarrollar una nueva clase de software que le habla a los usuarios en formas diferentes que las del pasado.

Una GUI está basada en las siguientes cuatro características: -Interfase de usuario gráfica. -Display de alta resolución, de color. -Lo que ves es lo que obtienes (What You See Is What You Get -WYSIWYG-) -Manipulación directa.

El moverse al mundo de la GUI requiere tres cosas: -Hardware que haga estas cuatro características posibles. -Software del sistema que le haga fácil a los desarrolladores de aplicaciones el construir aplicaciones gráficas. -La plena disponibilidad de aplicacianes escritas de esta forma.

La interfase de usuario común de una GUI define una forma estándar de darle órdenes a la computadora para que haga cosas. Mucha de la infraestructura requerida para tener aplicaciones comunes viene del subordinado sistema operativa.

Una buena GUI es poderosa e intuitiva en formas que ninguna interfase basada en terminal puede igualar. El poder desplegar información gráfica directamente en la pantalla hace a las aplicaciones existentes dramáticamente más simples, pero también abre nuevas clases de aplicaciones a su vez.

La simulación crea un mundo real virtual. El ambiente gráfica hace trabajar a la aplicación. En un sentido limitado, las GUIs son un caso especial de realidad virtual en el trabajo. La GUI como realidad virtual es frecuentemente referida como WYSIWYG. WYSIWYG y la Realidad Virtual son dos lados de una mismo moneda, ambas importantes al permitirles a las computadoras interactuar más naturalmente con los humanos.

El concepto central atrás de la interacción en la realidad virtual es la manipulación directa. El arrastrar y dejar caer (drag and drop) es particularmente atractivo porque es directamente muy análogo a lo que la gente hace en el mundo real. Hoy en dio, hay varias formas estándar de interactuar con objetos en el mundo virtual de las GUIs. Drag and drop es un enfoque; los otros dos enfoques estándares son el doble click y los inspectores de propiedad. La acción llamada inspección de propiedad, le permite al usuario inspeccianar inmediatamente y fijar las propiedades clave del objeto seleccionado. La inspección de propiedad es mucho más directa y eficiente que su alternativa, la interacción basada en comandos. La manipulación directa tiene un aspecto importante: el usuario explora y controla el mundo gráfico sin tener que encontrar o invocar comandos.

En el mundo de la GUI, el ratón (y quizá una pluma o una pantalla sensible al toque -touch sensitive screen-) puede crear eventos también. Un programa manejado por eventos crea un mundo de tremenda variedad. La primera consecuencia del paradigma del manejo por eventos es que el sólo mantener todas las acciones de las usuarios requiere mucho poder de la computadora. El soporte de ratón gráfico le da un nuevo significado a la programación manejada por evento en a lo menos tres formas. Primero, los eventos ahora ocurren cientos de veces por segundo. Segundo, el soportar todas las formas del ratón y el sobreponer el cursor del ratón sobre la otra información incrementa la cantidad de trabajo que se hace cada vez que un evento acurre. Finalmente, el ratón le da al usuario una nueva libertad para moverse alrededor de la pantalla. Adicionalmete se necesita un considerable poder de la computadora para decidir cómo dibujar, desplegar cada letra en la pantalla (tamaño, estilo tipo, color , etc.).

Una interfase GUI involucra el manejar 500 veces más de información, procesar esa información más frecuentemente en formas más complejas, y hacer todo aparecer instantáneamente todo el tiempo. Todavía, la necesidad de construir interfases humanamente usables va a llevar a cambios aún más dramáticos en los años que vienen. Las GUIs de PCs tienen que mejorar en cuatro áreas: velocidad, resolución, tamaño, y el factor de forma. En sí, la meta es empacar suficiente poder computacional en una caja que sea lo suficientemente pequeña y ligera para ser conveniente.

Las aplicaciones gráficas manejadas por ratón, están lejos aún de ser realmente fáciles de usar. Computadoras portables, baratas y ligeras con pluma de trabajo sustituyendo al no siempre práctico ratón, estarán disponibles dentro de las próximas 3-5 años.

La interfase de usuario misma ha sido quien ha manejado la explosiva necesidad de PCs más rápidas. Las GUIs hicieron al software útil, y la utilidad del software hicieron a las computadoras deseables.

Los sistemas computacionales del futuro dependerán en tener una computadora poderosa en cada escritorio del usuario; el diseño de la UI que aparezca en esa computadora hará o quebrará a las aplicaciones del mañana. Aún las PCs de hoy así como las interfases de usuario están lejos de ser adecuadas. La computadora de escritorio del año 2000 hará que las máquinas de hoy se vean como juguetes lentos y crudos.

1.1.2.4 El Servidor: La Primera Vez Alrededor del Cliente/Servidor.

El término servidor tiene dos significados: 1. Una pieza de hardware que proporciona servicios compartidos en un ambiente de red. 2. Un componente de software que proporciona un servicio funcional generalizado a otros componentes de software. El rol del servidor es facilitar el compartir. El propósito de un servidor es hacer trabajo para todas las PCs que dependen de él. La mayoría del trabajo del servidor involucra el manejar recursos compartidos. Una definición de cliente/servidor gira alrededor de computadoras compartidas llamadas servidores que manejan recursos compartidas y proveen acceso a esas recursos compartidos como un servicio a sus clientes.

Un servidor puede ayudar de las siguientes formas. Puede hacer un spool de la impresión. Esto es, si la impresora está ocupada imprimiendo algo más, el servidor puede salvar la salida a disco hasta que la impresora esté libre. Mientras el servidor maneja la terminación del proceso de impresión, la computadora que hizo la petición está libre para hacer algún otro trabajo rápidamente.

Un servidor puede proporcionar un rico menú de servicios alrededor de esta aparentemente simple función: 1. El servidor puede manejar varias impresoras, mandando la impresión automáticamente a la primera impresora libre. 2. El servidor puede mantener el control de papel especial. 3. Los trabajos de impresión pueden ser calendarizados para efectuarse a intervalos regulares. 4. Si un trabajo de impresión falla, el servidor puede reempezar el trabajo de su archivo de spool sin pedirle al usuario que vuelva a introducir la petición de impresión. 5. El acceso a las impresoras puede ser extendida y restringida, protegiendo de esta forma el mal uso de las impresoras costosas. 6. El servidor puede tener control de la carga de trabajo, de reportes de uso de impresión, e identificar cuellos de botella potenciales.

En el software de red de hoy en día, una función esencial de la red es engañar a la PC del usuario para que parezca que tiene acceso a algunas facilidades locales no existentes. Conceptualmente, la red y el servidor le proporcionan a la computadora del usuario una impresora lógica. La computadora del usuario piensa que está físicamente conectada a una impresora.

Las unidades de disco fueran el segundo recurso en importancia usualmente ligado a los servidores. El gran disco duro del servidor se comportió dividiéndolo en pequeñas pedazos de un tamaño fijo llamados discos lógicos, de manera que según cada usuario se conectaba al servidor se conectoba a su vez a su disco lógico personal. Este modelo de servicio de disco ofreció beneficios significativos a los usuarios: - Caros discos pudieron ser compartidos por muchos usuarios. - El rápido disco del servidor dio un mejor desempeño que los discos más lentos de uno PC. - Extendiendo a los muchos usuarios en un disco más grande, el espacio fue usado más eficientemente. - Asignando un administrador para que cuide del servidor, el respaldo de información pudo ser automatizado. Sin embargo en este modelo ningún usuario podia compartir información con otros usuarios.

Lógicomente, la primera generación de servidores hizo introducir el concepto de dispositivos lógicos.

En el mundo de la computación, la unidad básica de información en un disco es el documento (archivo). En 1983, Novell intradujo el concepto de compartir archivos. Aqui se mantuvo el disco duro como uno solo, y se dejó a todo mundo trabajar en el mismo disco al mismo tiempo. Asi que sin alguna forma de organizar todos los documentos en un disco en una forma con sentido, el compartir el disco no tiene sentido.

En el compartir documentos, el servidor agrega muchos beneficios: 1. Todos los documentos son ahora accesibles de cualquier parte de la red. 2. Coordinación de acceso a documentos. 3. Los procedimientos de respaldo automáticos proporcionan muchos niveles de protección en contra de desastres. 4. Los usuarios pueden crear estructuras de archivos sofisticadas. 5. El servidor puede hacer cumplir procedimientos de seguridad estrictos. 6. La

computadora puede incansablemente implementar procedimientos de búsqueda sofisticados.

Los clientes y las servidores son sólo computadoras. Exactamente estas mismas cajas, diseñadas para ser PCs baratas y fáciles de instalar, pueden funcionar como cliente o como servidor. Aún en un mundo de poderosas PCs hay una fuerte necesidad de compartir información y recursos. El servidor PC, nació con un simple destino: dar un mecanismo para compartir hardware costoso. Al principio, aunque las PCs habían empezado a actuar como servidores, había todavía una clara distinción entre computadoras grandes y pequeñas: las grandes computadoras controlaban información compartida; las pequeñas computadoras, aún cuando actuaban como servidores, operaban sólo con información individual. Con la introducción de compartir archivos y después de las Bases de Datos, aún pequeños servidores se convirtieron en dispositivos donde se compartía información. Simplemente compartir información está lejos de ser suficiente para reemplazar a los *mainframes* y a las minicomputadoras. Al mismo tiempo, el proporcionar un mecanismo para compartir y controlar el uso de esa información, los servidores se convirtieron en muchos sentidos el equivalente de *mainframes* para pequeños grupos de trabajo. Un servidor que está coordinando y controlando la utilización de información para un grupo de gente en este sentido ha hecho una transición funcional crítica de ser un dispositivo de compartir pasivo a ser un sirviente activo. El compartir archivos –y aún más, el compartir Bases de Datos– da fundamentalmente nuevas capacidades a los usuarios, capacidades que son intrínsicamente dependientes en el hecho de que el servidor es una computadora y no sólo un dispositivo de conmutación sofisticado. De hecho, el servidor es actualmente un tipo de sirviente incansable; un sirviente que proporciona servicios a una variedad de clientes, cada uno de los cuales es también una computadora. Lo que realmente hace a este sistema poderosa, es la manera en que las computadoras personales y los servidores trabajan juntos.

1.1.2.5 El *Mainframe* El Cambio Técnico a Sistemas Distribuidos.

Con la introducción de la minicomputadora, la gente empezó a hablar acerca de una nueva clase de sistemas distribuidos. A pesar de los muchos intentos en los últimos 20 años para hacer esto trabajar, la mayoría de los grandes sistemas de negocios de hoy en día están altamente centralizados. Con las PCs, los servidores, las WANs, y las LANs, los sistemas distribuidos se volvieron aun más atractivos que en el pasado.

Desde una perspectiva de hardware, un *mainframe* no es tan diferente de una PC. A lo que más se parece un *mainframe* es a una LAN. Ningún *mainframe* es miles de veces mas rápido que una PC. Cuesta miles de veces más que una PC, pero no hace miles de veces más trabajo. Aún con un presupuesto infinito, hay un límite superior muy real para el poder de la computadora más grande posible.

Posiblemente la clase de trabajo que mejor caracteriza a los *mainframes* es un grupo llamado aplicacianes de procesamiento en lote (batch). Estas aplicaciones procesan datos en grandes lotes. El procesamiento en lote es importante por dos razones. Primero, virtualmente cada gran organización tiene grandes cantidades de procesamiento en lote. Segundo, el procesamiento en lote es manejodo mejor por un *mainframe*. En muchas compañías, los trabajos en lote definen junto con las demandas en línea mandadas a la base de datos central, el tamaño del *mainframe*. El crecimiento del procesamiento en lote ha sido una de las razones de la necesidad continua por los *mainframes*. El procesamiento en lote causa constantemente que la gente pida más computadoras más rápidas y grandes.

La escalabilidad es la habilidad para adaptarse sin problemas a los cambios en tamaño de una computadora a través de una serie de etapas. La combinación de operaciones grandes, medianas y pequeñas crea el reto de escalabilidad. Todo debe ser escalable (discos, aplicaciones, sistema operativo, etc.) para cumplir con las necesidades de todo tamaño de compañías, oficinas y lugares. Aquí hay que recordar que las grandes compañías necesitan tanto de las grandes computadoras como de las pequeñas y de todas las que están entre estas dos tamaños.

Originalmente una diferencia importante entre un *mainframe* y una PC fue el espacio de direcciones (area de memoria) que el set de instrucciones de la computadora podía utilizar. Hasta 1990, las PCs no hacían realmente un uso de la memoria tan efectivo como los *mainframes*. Hasta ese tiempo, la memoria fue una de las razones de importancia para correr grandes aplicaciones en un *mainframe*. Aun ahora, una razón en favor de las *mainframes* sobre computadoras más pequeñas en cuanto a lo que se refiere a grandes trabajos, es que el hardware y el sistema operativo del *mainframe* soportan grandes cantidades de memoria y grandes espacios de direcciones.

Los *mainframes* tienen grandes cantidades de almacenamiento permanente. Los sistemas de almacenamiento de un *mainframe* no son sólo grandes, sino también muy sofisticados. Su aspecto más sobresaliente es la extensión a la que todo trabaja automáticamente: el sistema automáticamente respalda toda la información cuando es creada, modificada o movida. Los beneficios de las sistemas de almacenamiento permanente de un *mainframe* son críticos si una organización depende de sus Bases de Datos como su medio de almacenamiento y de transacción de información primaria. Hasta muy recientemente, las PCs y aún las estaciones de trabajo UNIX no podían acercarse al tamaño y sofisticación de los sistemas de almacenamiento permanente de un *mainframe*. En los últimos cinco años, la tecnología de almacenamiento de una PC ha cambiado a un ritmo muy rápido. A finales de los 80s, la tecnología RAID (Redundant Arrays of Inexpensive Disks) le dió a las PCs el medio de almacenamiento permanente masivo que necesitaban. En un sistema RAID, un disco grande y caro es reemplazado por muchos discos mas pequeños y baratos. Con la introducción de los sistemas RAID, las redes LAN de PCs empezaron a proporcionar a los usuarios formas de almacenamiento altamente estratificadas.

Las ventajas de un *mainframe* están rápidamente desapareciendo con la introducción de servidores de tipo PC más grandes, con el almacenamiento RAID, y con los sistemas operativos de 32 bits para PC. Es más, si se empiezan a considerar las Bases de Datos distribuidas, el almacenamiento basado en PC ha dado un salto adelante de los *mainframes*. Las Bases de Datos replicadas son las que respaldan a este tipo de aplicación, estando esto sólo disponible en las PCs.

Por otra parte, los *mainframes* pueden permitir a gran número de transacciones simultáneas y trabajos en lote muy grandes acceder la mayoría de las partes de la base de datos de una compañía, manteniendo aun así una respuesta adecuada en tiempo y la integridad de la base de datos (aún si el sistema falla).

El pensar y trabajar con la computación orientada a *mainframe* en los últimos años de los 60s era tan excitante como lo es ahora la cliente/servidor.

Las grandes compañías de hoy en día procesan grandes cargas de trabajo con seguridad, pero el costo es una burocracia masiva cuyo mantenimiento es caro y difícil de entender o cambiar. Las *mainframes* tienen exactamente las mismas características. En un ambiente centralizado como éste, existe una gran cantidad de trabajo administrativo adicional para mantener todo en marcha. La era del *mainframe* ha probado lo práctico de la planeación y contral central.

Las computadoras personales se volvieron populares tan rápidamente por la independencia que prometieron a los usuarios. Muchas grandes organizaciones hoy han excedido las limites prácticos de lo que sus *mainframes* centrales pueden hacer. Según una compañía alcanza los límites de sus *mainframes*, la firma puede empezar a hacer cosas que mejoren la eficiencia del *mainframe* pero recortando también así la eficiencia del usuario.

En muchas compañías grandes, la razón principal de instalar Bases de Datos relacionales es hacer la vida del programador más fácil; las Bases de Datos se convierten en una herramienta para simplificar el desarrollo de aplicaciones. Pero aún con toda esta capacidad de desempeño, el *mainframe* puede dar servicio a sólo algunas consultas a la vez. Las consultas parecen más bien trabajos en lote que transacciones. Un *mainframe* que procesa comúnmente cientos de transacciones cada segundo puede tomarse media hora o más en dar la respuesta a una simple consulta. Consecuentemente, en la mayoría de las compañías, aún los bases relacionales no están realmente disponibles para los usuarios finales. El *mainframe* continúa siendo una bestia centralizada debido a sus costos. Y debido a que las empresas insisten en limitar la carga de trabajo total de una organización entera a través de una sola máquina, no se pueden dar el lujo de permitir que esa máquina sirva las necesidades del individuo.

Cuando los sistemas se hacen muy grandes, llegan a un punto donde los humanos no pueden más entenderlos. La solución estándar a este problema es dividir tales sistemas en partes más pequeñas. Mientras las partes tengan caminos bien definidos de comunicarse entre ellas, un sistema grande puede funcionar, pero aquí ya no cabe que alguna persona sola o un pequeño grupo de gente opere solamente ese sistema. Mas bien el sistema más grande resulta de la interacción cooperativa entre los sistemas más pequeños. En términos computacionales, este tipo de sistema es un sistema distribuido.

La forma en que un *mainframe* lleva a cabo su gran desempeño es haciendo muchas cosas a la vez. El que pueda hacer tantas cosas tan rápidamente se debe a que un *mainframe* es esencialmente un grupo de computadoras más pequeñas que están empaquetadas en una caja y conectadas conjuntamente en una red de alta velocidad interna llamada bus. Una de las cosas que un *mainframe* hace mejor es el mantener esas piezas ocupadas tanto tiempo como sea posible. Por esto, el *mainframe* no es una computadora sola, sino que consiste típicamente de varias computadoras más pequeñas. Entonces, si un *mainframe* es un montón de computadoras actuando como una sola gran computadora, ¿por qué no tomar otros tipos de computadoras más pequeñas y hacerlas actuar también como una gran computadora?. De hecho, ¿por qué no hacer que un conjunto de computadoras personales hagan el trabajo?.

Una vez que se ha visto que aún las computadoras grandes son sólo grupos de computadoras más pequeñas, es natural mantenerse en la idea de usar computadoras pequeñas de gran capacidad para construir esos grupos. ¿Por qué no construir el equivalente de un *mainframe* de los mismos microprocesadores usados para construir computadoras personales?.

Muchas transacciones pueden ser fácilmente distribuidas entre muchas computadoras pequeñas. Una aplicación cliente/servidor corre 12 veces más rápido que la de un *mainframe*. La mayoría de las aplicaciones en lote no pueden ser pasadas sin embargo a sistemas cliente/servidor. En lugar de esto, la mayoría de los trabajos en lote tienen que ser reescritos para que corran en un ambiente cliente/servidor. Algunas aplicaciones en lote no pueden ser esparcidas entre múltiples computadoras.

Históricamente, la mejor forma de obtener más poder era comprar una computadora más grande. Ahora la mejor manera de obtenerlo es comprar computadoras más pequeñas. Se tiene que comprar muchas computadoras pequeñas, las que producirán adicionalmente más poder computacional por menos dinero.

Hoy en día aún para procesamiento en lote se obtienen ahorros enormes pasando de *mainframes* a redes de computadoras personales. Ya no se necesitan usar grandes computadoras para combinar trabajo de muchas partes diferentes. El resultado es computación más simple, costos más bajos, y un mayor desempeño. Claro, el llegar a esto tomará aún tiempo. La experiencia del siglo veinte proporciona un argumento obligado para

las operaciones descentralizadas de todo tipo. Aunque los *mainframes* tienen algunas capacidades remarcables, las redes de computadoras personales pueden reemplazar a esas grandes máquinas. No con PCs solas, sino con redes.

Los sistemas cliente/servidor de PC son un modelo obligado de la red como una computadora. La LAN puede ser un mejor *mainframe* que lo que este fue. La LAN puede también eliminar los problemas de complejidad y escalabilidad que sufren los *mainframes*. Los sistemas distribuidos tienen sus propias complejidades, pero cuando menos existe la posibilidad de poder escoger una complejidad sobre la otra.

1.2 Estrategia Cliente/Servidor según Donovan.

Aquí se presenta un punto de partida para agregar valor a los productos y servicios de una organización a través de tecnología. La arquitectura cliente-servidor de 3-capas fue introducida por el Grupo de Tecnología de Cambridge (CTGroup) y después fue adaptada por la industria como un modelo fundamental de una robusta arquitectura cliente-servidor. El Dr. Donovan fue pionero en establecer las bases para esta arquitectura computacional de 3-capas.

La línea fundamental para las organizaciones en todos lados es confusa. Los procesos de Negocios y la tecnología de información (IT) no están en armonía. Todavía la IT permanece como una esperanza importante para agregar valor a productos y servicios.

El Ideograma Chino para "crisis" y "oportunidad" es el mismo. Hoy en todo el mundo muchas organizaciones están luchando mientras otras prosperan. Muchas organizaciones están tratando de corregir los negocios sistemáticos que los afligen, aplicando técnicas que simplemente no trabajarán. A lo mejor se satisfacen necesidades de supervivencia a corto plazo, pero no proporcionan un cimiento para una estrategia exitosa a larga plazo. Lo que trabaja aquí es que las organizaciones crezcan sólo agregando valor continuamente a sus productos y servicios.

Los procesos de Negocios están interconectados, pero las tecnologías de información no lo están. La IT no está en armonía con los negocios. Ante ésto lo único que se puede hacer es armonizar la organización. Aquí se dá un enfoque diferente para agregar valor continuamente al negocio. Esto a través de una arquitectura diferente para la tecnología de información: una arquitectura cliente-servidor de 3-capas basada en sistemas abiertos y herramientas del Ambiente Distribuido Abierto (ODE).

El Negocio –y la IT que la habilita– es un viaje, no un destino. El trabajo es guiar a la organización en ese viaje. Esta decada presenta oportunidades de negocio fenomenales. Esto se vé en todas partes. La rápida metamorfosis de modos tradicionales de pensamiento y maneras de hacer negocio en radicalmente nuevos paradigmas.

La misión aquí es revolucionar la forma en que se hace negocio y soportar esa revolución con una IT en evolución constante. La habilidad para identificar y aplicar las opciones apropiadas en comunicacianes y computación determinará qué organizaciones tendrón éxito en la próximo década.

Las estrategias promovidos en el pasado por vendedores de hardware no han cumplido la promesa de mejora de negocios y ventaja estratégica. Las viejas estrategias emergieran para solucianar problemas que no existen más, y son incapaces de evolucionar lo suficientemente rápido para tratar con los problemas del mundo real que se encoran hoy en día. La llave del éxito es identificar e implementar aplicaciones estratégicos que praduzcan valor agregado.

Según el poder económico cae, el poder ideológica lo sigue. El estándar de vida en una región está directamente relacionado a su praductividad. Una arquitectura cliente–servidor de 3–capas que usa sistemas abiertos basados en herramientas ODE puede entregar una poderosa ventaja sobre los oponentes.

1.2.1 La Realidad de Hoy: Las Organizaciones en Crisis.

Las organizacianes que continuan operando usando las expectaciones y modelos del ambiente de negacios antiguo están luchando por sobrevivir.

El mundo de negacios que se enfrenta hay en dia es un lugar diferente del que existió hace una década. Muchas compañias que se creyó que eran invencibles están de repente luchando por sobrevivir. Existen diferentes rasgos que caracterizan a este nuevo ambiente. No obstante el caos mundial en la actividad de negocios durante los primeros años de esta década, las oportunidades de negacios globales se están presentando por sí mismas con asombrosa regularidad. Pero estas nuevas oportunidades de mercado deben ser enfrentadas con productos y servicios de alto desarrollo, correctos, y culturalmente enfocadas. Estos cambios crean una grande y constante agitada alberca de nuevos mercados y oportunidades de negocios esperando ser capturadas por organizaciones con suficiente flexibilidad y visión. Los nuevos mercados tienen un vasto potencial para crecimiento, pero presentan retos únicos que muchas organizaciones no pueden enfrentar. Los productos y servicios deben de ser adaptados a los requerimientos culturales y de lenguaje de cada país.

Mientras las organizaciones examinan las nuevas oportunidades que los mercados que emergen ofrecen, también deben concursar con transformaciones rápidas en las relaciones de negocios tradicionales así como en los modelos operantes. Algunas organizaciones han divisado reenfocar sus esfuerzos en sus funciones de negocio básicas. Tal enfoque ha llevado a una reestructuración general, a un "downsizing", y a reducciones de personal según las funciones del negocio se consolidan y simpifican. Desafortunadamente los procesos "simplificados" son usualmente difíciles de llevar a cabo, debido a que los anticuados sistemas de información que los soportan no puden ser rápida y facilmente cambiados para enfrentar los nuevos requerimientos del negocio.

Todas las organizaciones deben responder a los cambios significativos en las expectativas y realidades del mercado global. Los más prominentes son los que se refieren a la calidad, tiempo de mercadeo y potenciales de beneficio. Los clientes en todos lados ahora esperan calidad total de los productos y servicios de una organización. Las organizaciones que puedan detectar una oportunidad de mercado, implementar una solución, y entregarla rápidamente, son las organizaciones que triunfarán. Así mismo se deben de mejorar los productos y procesos con menos dinero que antes, debido a la presión en los márgenes de beneficio.

Las nuevas oportunidades, relaciones y expectativas de mercado demandan nuevas estrategias para el éxito. Solo algunas organizaciones pueden actualmente demandar que poseen tales estrategias.

1.2.2 Soluciones de "Sentido Comun": Prescripción para el Fracaso.

Las respuestas tradicionales a las fuerzas de cambio que recorren las ambientes de negocio de hoy en día son inefectivas y están condenadas al fracaso.

Para corregir el crecimiento en la productividad y para crear barreras a largo plazo de entrada para competidores, algunas organizaciones tomaron uno o más de los siguientes pasos:

. Recortar costos
. Buscar legislación proteccionista
. Ejecutar reestructuración financiera
. Promocionar nombres de marcas

Estas respuestas no proporcionan más que una barrera a largo plazo para entrar y cuando mucho proporcionan salamente un alivio competitivo a corto plazo. El

proteccionismo produce un falso sentido de seguridad en las organizaciones suprimiendo artificialmente la presión de los competidores. Tal satisfacción lleva a estancamiento y otrofia el desarrollo e innovación de productos.

La reestructuración financiera es finalmente sólo una distracción ociosa del trabajo real de proporcionar productos útiles que satisfagan las necesidades del cliente. Por otra parte no se puede contar más en identidad de marcas para asegurar el éxito del producto en el mercado.

A pesar de los mejores esfuerzos de ejecutivos corporativos de buen sentido, las soluciones tradicionales de "sentido común" a los retos del ambiente de negocios de hoy en día están mal adaptadas a la tarea. Los líderes exitosos necesitan una nueva estrategia: una que le dé poder a sus organizaciones para enfrentar las necesidades de sus clientes de una manera que genere una ventajo de negocios estratégica sobre la competencia. La estrategia debe también proparcionar una infraestructura que continuamente se acomode al cambio.

1.2.3 La Solución Correcta: De Gran Volumen a Gran Valor.

Las organizaciones deben continuar agregando valor a sus productos y servicios. La Información de Tecnología (IT) juega un papel fundamental en la habilidad de una organización de agregar valor.

¿Cuándo es un sistema de informacion estratégico? Es estratégico cuando le permite a una organización enfrentar sus más altas metas, dándole a esa organización una ventaja competitiva. Desde un punto de vista de orquitectura de la información, la solución involucra tomar diferentes tipos de informacion de adentro y fuera de la organización, procesándola y presentándola en una forma que es facilmente accesible a la gente correcta. Tal solución es pasible usando una arquitectura cliente-servidor.

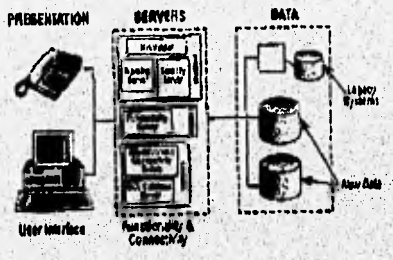
Aún cuando una organización es la primera en introducir una aplicación estratégica, un retardo de varios años de desarrollo y pruebas significa que el nuevo sistema está desactualizado para el tiempo en que es instalado. Desafortunadamente las arquitecturas de información de la mayoría de las organizaciones no son lo suficientemente flexibles y robustas para integrar diferentes sistemas de computadora facilmente.

La raíz de los problemas no es los Sistemas de Manejo de Información (MIS). El problema real es que los procesos de negocios están organizados de una manera diferente que la IT. Para cerrar este hueco, las organizacianes deben reconocer que necesitan entrar en contacto con aplicaciones estratégicas de una forma por mucho extirpada del desarrollo

de aplicaciones tradicional. Sólo recientemente tales armas y tácticas especiales han empezado a emerger. Un número creciente de organizaciones han volteado a esta nueva propuesta que aquí se plantea. Con ésta se construyen aplicaciones estratégicas en 2 meses, en lugar de 2 a 5 años. Y lo más importante es que las aplicaciones fueron construidas con una nueva infraestructura– una arquitectura que se puede adaptar continuamente al cambio en el negocio y en la tecnología.

La única solución que le dá poder al negocio para llevar a cabo completamente las ventajas competitivas que se necesitan, es una arquitectura cliente-servidor basada en sistemas abiertos estándares de industria. La arquitectura cliente-servidor involucra el separar las siguientes funciones de la aplicación en componentes intercambiables o "capas":

. Presentación (o "interfase de usuario")
. Funcionalidad, conectividad y servidores de bases de datos
. Datos, sistemas y aplicaciones existentes que han sido encapsulados



Arquitectura Cliente/Servidor de 3-Capas.

Cuando los usuarios interactúan con una nueva aplicación estratégica construida sobre esta arquitectura, éllos solo tratan con la capa de presentación; los servidores interactuan con las bases de datos o con los sistemas heredados existentes para manipular la información que el usuario requiere. Al separar una aplicacion en 3 capas (presentación, servidores de funcionalidad y datos), la organización gana lo siguiente:

. Libertad para seleccionar cualquier sistema de manejo de base de datos
. Respaldo dinámico
. Libertad para seleccionar cualquier interfase de usuario gráfica
. Flexibilidad para agregar nuevas tecnologías según se necesiten
. Acceso en línea a cualquier fuente de datos
. Costo más bajo de hardware
. Libertad para seleccionar el hardware del tamaño apropiado
. Abertura para valor agregado continuo
. Soporte para una estrategia de migración de sistemas viejos
. Preservación de la inversión actual de hardware y software

. Desarrollo paralelo de aplicaciones
. Diferentes vistas de usuario de datos
. Seguridad
. Robustez y tolerancia a fallas

La Tecnología de Información basada en una arquitectura cliente-servidor de 3-capas y los sistemas abiertos representan la clave que puede desbloquear el vasto potencial para nuevos negocios escondidos dentro de los sistemas computacionales actuales. Mejor que tado, ese potencial puede llevarse a cabo en un sorprendente corto tiempo, debido a que la propuesta cliente-servidor trae consigo velocidad de implementación así como flexibilidad.

1.2.4 La Solución Correcta: Proceso para Progreso.

Para construir sistemas estratégicos exitosos se debe:

1. Implementarlos usando una arquitectura cliente-servidor.
2. Implementarlos en-casa o con un socio.
3. Direccionar los factores de éxito críticos de la organización.
4. Establecer un caso de negocios para justificar, disciplinar y monitorear la aplicación estratégica.
5. Involucrar a los usuarios en todas las fases del desarrollo de la aplicación.
6. Buscar y explotar las "tesoros escondidos".

Es fácil hacer declaraciones de la elegancia y poder de las aplicaciones estratégicas. Es mucho más difícil identificarlas y construirlas, y después superar actitudes entrincheradas e inercia para desplegar tales sistemas en la organización.

El darle el control de una aplicación estratégica a una empresa externa es como dar la llaves de la compañía a un extranjero. Para que una organización sea exitosa en ésto, la IT debe desarrollar sistemas más rapidamente que nunca antes. Una vez más, una arquitectura cliente-servidor hace ésto posible.

La nueva variedad de aplicaciones estratégicas están dirigidas a funciones de "misión crítica" de alta recompensa, las cuales le agregan valor a los productos y servicios, expanden los mercados compartidos, impulsan la ganancia, reducen los costos, y llevan a la organización a nuevas líneas de negocio.

Hoy en día los ejecutivos deben examinar los proyectos de su organización cuidadosamente. Pocas organizaciones están trabajando en proyectos de sistemas de

información que puedan ser considerados verdaderamente estratégicos. Sólo una fracción de las aplicaciones dignas de consideración de una empresa pueden ser estratégicas.

Las mejores compañías gastan 80% de su presupuesto de IT en sistemas operacionales y 20% en sistemas de oficina. Esta nos dice que se debe de gastar el dinero de IT en aplicacianes estratégicas.

Cuando un sistema "remendado" promete restaurar los beneficios perdidos y calmar a clientes enojados, no obstante, es estratégico. Otro tipo de aplicación de alta-recompensa es la que ayuda a los gerentes a hacer decisiones efectivas críticas para el éxito de la organización.

Muchas ideas de aplicación que inicialmente parecen estratégicas después quedan cortas en tener un impacto significativo en esos elementos del negocio que realmente necesitan ayuda. Una aplicación estratégica debe soportar las metas de una organización, las cuales a su vez están soportadas por los factores de éxito críticos de la organización, dándole asi a la misma una ventaja competitiva. Una meta es un final fundamental el cual una organización quiere llevar a cabo; un Factor de Exito Crítico (CSF) es un medio para un final. No hay reglas sobre lo que constituye un CSF, tal y como no hay reglas sobre lo que constituye las metas de una organización. Ambos deben de ser determinados por altos gerentes familiares con las productos de la organización, aperación, clientes, y competidores.

Se deben identificar las metas y CSFs de las organizaciones que representan fuerzas competitivas, así como examinar formas en las cuales la aplicación propuesta puede mejorar la relación con estas fuerzas. Es como encontrar formas para ayudar a que estas organizaciones alcancen sus propias metas incluyéndolas en la aplicación que se está diseñando.

La información es valiosa para todos. Uniones creativas entre organizaciones pueden mejorar la posición competitiva y son dignas de ser buscadas.

Hay 3 razones por las que un caso de negocios es un paso necesario en la construcción de un sistema estratégico. Primero, proporciona una justificación de costo valiosa. Segundo, el análisis completo disciplina a los usuarios y a la gente técnica fijando metas presupuestarias. Finalmente los beneficios proyectados sólo pueden ser realizados monitoreando el sistema durante la operación.

Una lista de beneficios de usar sistemas cliente-servidor, es la siguiente:

. Ganancias incrementadas por llamada de ventas.
. Ganancias incrementadas por minuto de teléfono.
. Ganancias incrementadas por venta cruzada.
. Ganancias incrementadas de servicios no relacionados.

. Costos de entrenamiento reducidos.
. Errores reducidos.
. Redundancia de datos reducida.
. Costos de inventario decrementados.
. Tiempo extra eliminado.
. Costos de hardware reducidos.
. Costos administrativos reducidos.
. Tiempo de empleados redireccionado.
. Desperdicio de recursos reducido.
. Mercados compartidos recapturados.
. Costos operacionales decrementados.
. Satisfacción de cliente incrementada.

Las aplicaciones basadas en una arquitectura cliente-servidor de 3-capas y en herramientas ODE son fáciles de cambiar, tanto en respuesta a cambios en tecnología como en peticiones de usuarios de caracteristicas adicionales. Asi mismo con éstas emergen inadvertidamente "tesoros escondidos", ésto es oportunidades de negocio provechosas, las cuales nacen del firme fundamento de estas nuevos sistemas de información.

Una vez que se identifica a la aplicación estratégica y se establece el caso de negocios necesario, se pasa a construirlo y desplegarlo. La metodología del Grupo CT permite construir un sistema fuerte de industria en no más de 20 semanas lo que cual es un tiempo de desarrollo muy corta si se compara con el 1 a 6 años que los sistemas tradicionales requieren para su desarrollo y prueba.

Muchas organizacianes encuentran que son capaces de concebir estrategias únicas basándose en sus aplicaciones. Tales oportunidades no esperadas pueden ser el aspecto más exitante de planear y desplegar una aplicación estratégica.

Algunas gerentes se llegan a preguntar si sus organizaciones son capaces de llevar a caba tales recompensas no esperadas a partir de una aplicación estratégica. La respuesta a ésto es que se ha encontrado que tal pago es más la regla que la excepción.

1.2.5 Soluciones Propietarias: Arquitectura de Parálisis.

Hay que tener cuidado de los vendedores que buscan seducir con viejas arquitecturas. En particular, hay que cuidarse de vendedores que ofrecen una arquitectura "cliente-servidor" compuesta sólo de 2 capas o construida con herramientas propietarias. Estas sólo encerrarían en soluciones propietarias.

Los sistemas *mainframe* actuales han servido, sin lugar a dudas, razonablemente bien por muchos años, y poseen beneficios especificas que se deben proteger y apalancar. Sin embargo, también sufren de severas limitaciones cuando se encuentran con los problemas de los ambientes de negocio de hoy en dia.

Sus ventajas incluyen:
. Estabilidad.
. Seguridad.
. Soporte.
. Base de Aplicaciones.

Así misma, esta arquitectura sufre de igualmente profundas limitaciones:
. Resistencia al cambio.
. Complejidad.
. Costo.

Con una arquitectura de este tipo, las ganancias en la productividad de la programación son difíciles de lograr. En lo que se refiere al "performance", en promedio cada transacción que tales sistemas ejecutan debe pasar a través de 400,000 lineas de código del vendedor antes de llegar a la aplicación que realmente hace el trabajo real.

Las siguientes acciones se deben de realizar para maximizar las inversiones en *mainframes* actuales:
1. Parar de desarrollar nuevas aplicaciones para los *mainframes*.
2. Mantener sistemas existentes como servidores en la arquitectura cliente-servidor de 3 capas.
3. Empezar una estartegia "right-sizing" para migrar las aplicaciones apropiadas de los *mainframes* a sistemas abiertos.

La arquitectura cliente-servidor separa varias piezas funcionales de una aplicación en unidades discretas, permitiendo que los cambios sean hechas dentro de esas unidades mientras otras unidades son protegidas.

Los vendedores han reconocido desde hace mucho tiempo las limitaciones de sus viejas arquitecturas y han buscado mejorarlas. El impulso es bueno; es la implementación la que ha fallado.

Las arquitecturas que se proponen que son solamente de 2-capas se deben de examinar cuidadosamente. En estas se pueden estar separando clientes y servidores, pero ambos componentes deben de correr en ese mismo hardware del vendedor. Se debe de demandar que la solución abarque las 3 capas completas usando Llamados a Procedimientos Remotos (RPCs) del Ambiente de Computo Distribuido (DCE) como lo definen

los cuerpos estándares neutrales de vendedor tales como La Fundación de Software Abierto (OSF) y UNIX Internacional.

Adicionalmente se deben de tomar las siguientes acciones para maximizar la inversión actual:
1. Mantener a los sistemas existentes como servidores dentro de la arquitectura cliente-servidor de 3-capas.
2. Enfocarse en implementar los procesos de negocios como clientes y capturar las nuevas funciones como servidores.

La arquitectura cliente-servidor de 3-capas con una interfase de usuario común une a todos los sistemas existentes y adquiridos en solo algunos meses, y asegura que las adquisiciones futuras puedan ser integradas en sólo días.

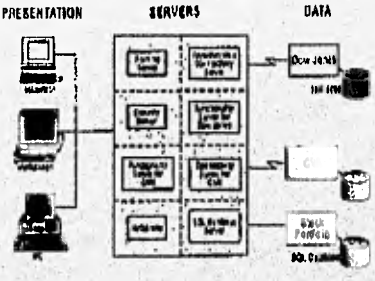
Existen dos razones adicionales para andar con cuidado con las estrategias de un solo vendedor propietarias. La primera de ellas es que es una mal práctica de negocio atar a la compañía muy estrechamente a la fortuna de un solo distribuidor. Existen alternativas mejores donde los sistemas son compatibles con los de otros vendedores, permitiéndose así una fácil transición a un nuevo distribuidor si se siguen las guías de sistemas abiertos. La segunda de ellas es que la propuesta de un solo vendedor pone en una posición de precio desfavorable. Aquí se necesita el poder de la competencia para asegurar que se reciba la mejor tecnología al mejor precio posible. Una vez más, una arquitectura cliente-servidor basada en sistemas abiertos estándares de industria asegura que se reciba precisamente éso. Esta además libera de las limitaciones de las viejas arquitecturas de *mainframe* y permite disfrutar de los beneficios de la competencia multi-vendedor.

1.2.6 Tecnología Cliente-Servidor: Arquitectura de Poderío.

La arquitectura de 3-capas permite construir aplicaciones rápidamente, modificarlas fácilmente según las circunstancias del negocio lo dictan, e involucrar usuarios directamente en el diseño y desarrollo para asegurar máximo uso y eficacia.

Los sistemas viejos no tienen que ser alterados de ninguna forma. El nuevo sistema puede ser construido rápida y baratamente porque usa una construcción "prefabricada", genérica que no requiere reconstruir las bases de datos de los sistemas viejos -la parte que más consume tiempo en la construcción de una nueva aplicación. En la mayoría de los casos, una aplicación estratégica basada en una arquitectura cliente-servidor puede ser diseñada, construida, e instalada en meses en lugar de años.

La arquitectura cliente-servidor involucra separar las siguientes funcianes de la aplicación en componentes intercambiables o "capas":
. Presentación.
. Funcionalidad, conectividad y servidores de bases de datos.
. Datos. Esta capa puede incluir a sistemas existentes y aplicaciones que han sido encapsuladas para tomar ventaja de esta arquitectura.



Aplicación Financiera.

Los usuarios sólo interactuan con la capa de presentación.

La capa de presentación proporciona los clientes de la interfase de usuario que soportan la interacción humano-computadora a través de dispositivos como el teclado, ratón, y monitor.

La capa de servicios es el corazón de la aplicación, donde el cómputo crítico se lleva a cabo. Los servidores realizan muchas funciones.

Aquí el agregar una conexión de datos adicional a una aplicación es tan simple como conectar un servidor más que se pueda comunicar con los nuevos datos.

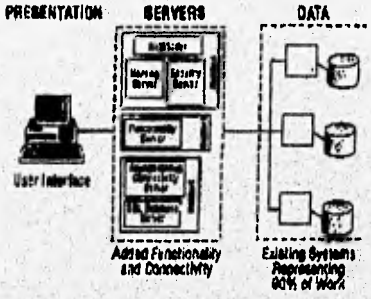
La capa de datos manipula toda la información que la aplicación finalmente busca alcanzar y usar. Los clientes de la interfase de usuario en la capa de presentación sólo se comunican con los datos a través de los servidores intermedios apropiados. Los tipos de datos que pueden poblar esta capa están completamente sin restricciones. Cualquier aplicación puede ganar acceso a cualquier tipo de datos, en cualquier sistema, en cualquier lugar, siempre y cuando el servidor apropiado esté en su lugar.

Cinco de las ganancias más importantes para todo tipo de organizaciones son las siguientes: velocidad en el desarrollo de aplicaciones, migración a sistemas abiertos y

"downsizing" gradual, conectividad a sistemas, evolución a una arquitectura de gateway, y construir apliacacianes de "scratch".

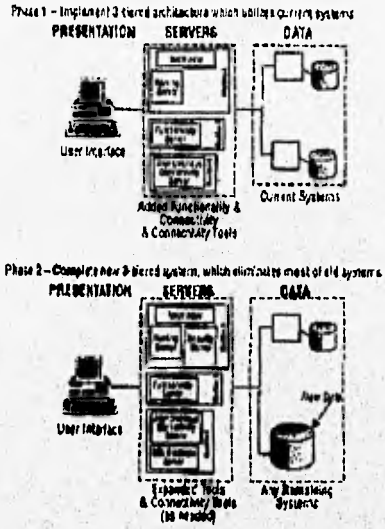
La arquitectura cliente-servidor puede ser aplicada a casi cualquier aplicación estratégica.

Las metodologías tradicionales toman más tiempo y cuestan más. Solamente el tiempo puede hacer la diferencia entre estar adentro y afuera del negocio en un momento dado. La tecnología cliente-servidor proporciona la integración requerida sin duplicar el esfuerzo encapsulado en el sistema existente.



Desarrollo Rápido.

La arquitetcuro cliente-servidor puede ser un puente crucial entre los ambientes viejos y nuevos, praporcionando acceso inmediato y continuo a los nuevas capacidades estratégicas mientras la arganización reemplaza gradualmente a los sistemas existentes. Una arquitectura cliente-servidor puede ser instalada en piezas sin interrumpir el funcionamiento de la nueva arquietctura.



Migración a Sistemas Abiertos.

La arquitectura cliente-servidor es capaz de obtener e integrar información de uno o dos sistemas que permanecen fuera de la organización. Esta a su vez no interfiere con las funciones normales de los sistemas.

El advenimiento de poderosas máquinas de bases de datos de propósito especial ha hecho posible enrutar datos a través de una sola máquina dedicada solamente a este propósito. Con la arquitectura cliente-servidor esta máquina sirve como un "gateway" o una puerta de datos permitiendo que los datos sean movidos entre cualquier aplicación, sistema, o interfase de usuario.

La arquitectura cliente-servidor ofrece una metodología modular, flexible y fácil para construir nuevos sistemas. La habilidad de esta arquitectura para unir muchos sistemas también se aplica cuando se habla acerca de subsistemas dentro de un mismo proyecto. Se puede construir un paquete en piezas y ver cada una de estas partes corriendo, mucho antes que el sistema final se complete.

Una arquitectura cliente-servidor de 3-capas proporciona la base sobre la cual construir las aplicaciones de información estratégicas que pueden energizar y dar poderío al negocio. Una vez que se ha adaptado esta arquitetcura el próximo paso es asegurarse que las herramientas de software que se emplean estén basadas en sistemas abiertos estándares de industria.

1.2.7 Herramientas Cliente-Servidor: Ambiente Distribuido Abierto.

Al conjunto de herramientas de software desarrollado específicamente para soportar el desarrollo de aplicaciones para esta arquitectura, es llamado el Ambiente Distribuido Abierto (ODE), el cual es ofrecido por la Corporación de Ambientes Abiertos (OEC). Hoy en día las herramientas ODE están siendo usadas por todo el mundo y están disponibles y soportadas en la mayoría de las plataformas de hardware y sistemas operativos.

La Fundación de Software Abierta (OSF) y UNIX Internacional han especificado los estándares para un ambiente de cómputo distribuido de objetos abiertos. Estos estándares incluyen:

. un método para que las aplicaciones se comuniquen una a otra a través de Llamadas a Procedimientos Remotos (RPC).
. un método para mantener control de las clientes y servidores a través de "servicios con nombre".
. un mecanismo de seguridad, Kerberos.
. Estándares de Ambiente de Aplicación (AES).
. un Ambiente de Manejo Distribuido (DME) para el manejo del ambiente.

ODE es un conjunto de herramientas de desarrollo de aplicaciones que permite utilizar el software y hardware de hoy en día en un ambiente cliente-servidor de 3-capas. Las herramientas de ODE proporcionan servicios en 5 categorías generales:

. Servicios Básicos.
. Puentes.
. Generación de Código y Servicios de Desorrollo de Aplicaciones.
. Servicios de Manejo de Aplicaciones.
. Conectividad y Servidores de Bases de Datos.

Cualquier arquitectura cliente-servidor tiene algun método para mandar información entre clientes y servidores. La arquitectura cliente-servidor de 3-capas de ODE usa un mecanismo de comunicación cliente-servidor estándar para lograr ésta meta. La tecnología RPC es uno de los estándares que permite a los programadores distribuir módulos de aplicación entre varias máquinas en una red. Una aplicación puede consistir de un módulo de interfase de usuario en una PC (cliente), y un programa que proporcione información (servidor). Los estándares RPC le permiten a un programa en la PC activar al servidor simplemente llamándolo remotamente. La tecnología RPC permite que 2 módulos se comporten como si ambas residieran en la misma máquina. Los servidores realizan servicios para cualquier cliente interesado, mientras los detalles de lugar e implementación son transparentes para todos los clientes y manejados por los servidores de nombres.

Las versiones iniciales de ODE usan la librería de RPCs de ODE. Las versiones posteriores de ODE le permiten a los desarrolladores usar los mecanismos RPC estándares de industria proporcionados por DCE de OSF. En los dispositivos de los clientes, las librerías

de ODE adaptan constructores de pantallas de GUIs para Microsoft Windows, OS/2, el Sistema 7 de Macintosh y Motif. En los dispositivos servidores, las librerias ODE adaptan máquinas de bases de datos. Los servidores ODE proporcionan una liga en las bases de datos de IMS y DB2 para los *mainframes* de IBM, y DMS1100 y DMS2 para los *mainframes* UNISYS. ODE proporciona los servicios que generan el código de procedimientos necesario para tanto el cliente como el servidor. Asi mismo éste permite a los administradores del sistema y a los desarrolladores de aplicaciones automatizar muchas de la funciones que deben ser realizadas para manejar tales aplicaciones.

El control de versión para el procesamiento de transacciones, la seguridad, y otros servidores administrativos también han sido implementados como parte de la arquitetcura ODE.

Aquí hay tres formas de acceder o actualizar datos: 1. SQL proporciona acceso directo a la base de datos. 2. Servidores uno-a-uno le permiten a un programa servidor invocar transacciones. 3. La emulación de terminal accede datos emulando una terminal y actuando como un usuario en una máquina remota.

La arquitectura cliente-servidor de 3-capas tiene 3 capas lógicas: -Presentación (o interfase de usuario) -Funcionalidad, conectividad y servidores de bases de datos. -Datos. Las herramientas ODE soportan cada capa.

Las herramientas de presentación de ODE consisten de herramientas y utilerías que residen en el dispositivo del cliente y le permiten a las aplicaciones constructoras de pantallas realizar RPCs para acceder datos o realizar procesamiento.

Las herramientas en la capa de funcionalidad le permite a los clientes y servidores interactuar entre múltiples ambientes de hardware y software. En el corazón de la capa de funcionalidad ODE está la herramienta ODE básica. Esta tiene los siguientes componentes: -"Broker" ODE. -"Make" RPC. -"Test" RPC. -"Debug" RPC. -"Netminder".

El resto de las herramientas de funcionalidad de ODE son conocidas como servidores de conectividad porque permiten acceso a datos de una variedad de fuentes.

La herramienta de canectividad remota proparciona servicios de canectividad y de adaptación adicionalmente a los servicios de generación de código y desarrollo de aplicaciones. Con esta herramienta, un desarrollador puede reducir el tiempo que se necesita para crear servidores de emulación de terminal.

La herramienta de conectividad Open3270 proporciona servicios de conectividad y adaptación asi como servicios de desarrollo de aplicaciones y generación de código. Con esta herramienta los desarrolladores pueden crear servidores de emulación de terminal que acceden *mainframes* IBM sobre redes SNA.

La herramienta de conectividad de bases de datos le permite a los desarrolladores construir servidores que acceden y manipulan sistemas manejadores de bases de datos relacionales.

La herramienta avanzada de conectividad de bases de datos le permite a los desarrolladores crear servidores que implementan servicios de procesamiento de transacciones en ODE.

Hay que tener cuidado de las soluciones de 2-capas. La capa media o de funcionalidad es crítica para implementar una solución. No hay que caer en la trampa de construir la funcionalidad del negocio en la capa de presentación. Hay que tener cuidado también de cualquier solución que no cumpla con los estándares ODE.

1.2.8 Paso 1: Sistema Piloto.

Un sistema piloto se puede construir en 1 a 3 semanas usando una arquitectura cliente-servidor y herramientas flexibles. El "staff" puede diseñar el sistema piloto en 5 días ó menos, y entonces implementarlo inmediatamente tomando ventaja de la flexibilidad de esta arquitectura usando las herramientas expuestas anteriormente. Es difícil creer que algún sistema con utilidad pueda ser construido tan rápidamente. Este rápido proceso de desarrollo depende de: 1. una arquitectura cliente-servidor de 3-capas. 2. "buses" estándares. 3. herramientales ODE que empleen todo lo anterior. 4. el equipo de las "Armas Especiales y Tácticas".

El sistema piloto no es un fin en sí. Este no se debe confundir con sistemas "prototipo" que se hayan podido construir en el pasado. Los sistemas pilotos que aquí se construyen son aplicaciones con funcionalidad pegados a sistemas existentes vivientes. El construir una versión piloto acelera, simplifica y enfoca el proceso de desarrollo de manera que un equipo de gerentes no técnicos pueda construir los elementos más estratégicos del sistema en una sola semana.

La arquitectura del sistema piloto le permite a gente no técnica que conoce las necesidades de la organización manejar el proceso. Sus esfuerzos están soportados por las herramientas ODE.

El equipo de trabajo debe de estar constituido por gerentes y usuarios. La gente técnica también juega un papel crucial ya que entienden a los sistemas existentes o los que la nueva aplicación se debe conectar. El equipo así mismo debe de incluir 2 o 3 personas que hayan tenido contacto con los conceptos de la arquitectura cliente-servidor. Toda esta gente debe de ser rotado en el equipo.

Se debe de checar continuamente que el sistema piloto siga cumpliendo con las metas de la organización así como con los factores de éxito críticos. En el desarrollo la gente técnico se ocupa de construir las partes del sistema que están atrás de la interfase de usuario mientras los usuarios construyen la interfase real.

El sistema piloto es una herramienta muy valiosa para vender la aplicación.

Un sistema piloto existoso, desarrollado rápidamente por usuarios y gerentes de negocios no técnicos en cooperación con un pequeño número de gente técnica, representa la base para un sistema en producción totalmente a escala. Una vez que se tiene un sistema piloto funcional, se está listo para desarrollar toda la aplicación.

1.2.9 Paso 2: Sistema de Producción.

Usando arquitectura cliente-servidor y herramientas de sistemas abiertos, los sistemas de producción toman de 10 a 20 semanas en implementarse en lugar del 1 a 6 años que toma con arquitecturas y técnicas tradicionales.

El sistema piloto es una versión sofisticada del sistema de producción el cual es funcional desde el principio.

El proceso de construcción de un sistema de producción es paralelo al de construcción de un sistema piloto. Se usa la misma arquitectura de aplicación así como la misma robustez. La arquitectura consiste de módulos que sólo necesitan ser adecuados para proporcionar las características requeridas por la aplicación estratégica.

El sistema de producción debe proporcionar un alto desempeño y un tiempo de respuesta rápido si es que va a ofrecer servicios de misión-crítica. Si el hardware de la computadora es lento, porciones de la aplicación pueden ser portados a una computadora más grande o distribuidas entre múltiples computadoras para elevar el desempeño. La arquitectura cliente-servidor realmente apoya esta distribución de la carga de cómputo. Esta ganancia en el "performance" es posible cuando se distribuye múltiples copias de un servidor en varias máquinas para así proporcionar procesamiento paralelo.

En el desarrollo del sistema de producción se debe de mantener a los usuarios involucrados. El checar puntos con usuarios clave es vital para el éxito. Según el desarrollo del sistema de producción se acerca a su finalización, los componentes deben de ser provados e integrados. Se debe de estar comprometido a un proceso de mejora constante en cada una de las aplicaciones estratégicas si se quiere alcanzar la meta de tales cambiantes circunstancias. Afortunadamente el modificar y mejorar una aplicación

estratégica es bastante fácil con una arquietectura cliente-servidor que con metodologías tradicionales.

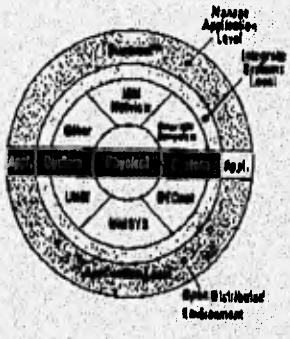
Los equipos de seguimiento deben revisar las metas y los factores de éxito críticos que definieron la idea de la aplicación estratégica original. Los equipos pueden determinar si las metas han cambiado o si nuevas metas están emergiendo. Según una organización continua modificando y expandiendo una aplicación, eventualmente tendrá sentido reestructurar tal aplicación y reemplazar los sistemas originales.

La arquitectura cliente-servidor le da a una organización 2 opciones: dejar los sistemas estándares originales intactos o reestructurar el ambiente para absorber y reemplazar los sistemas existentes. La flexibilidad de la arquitectura cliente-servidor hace a la última apción la opción más inteligente. Las organizacianes pueden usar a la arquitectura cliente-servidor como un puente estratégico entre las aplicaciones originales y las nuevas.

Una consecuencia de adoptar una arquitetcura cliente-servidor para sistemas de información estratégicos es la importancia incrementada de las redes de cómputo. Las redes proporcionan el elemento principal de comunicación para aplicaciones distribuidas. Con esta arquitectura las fallas en la red pueden significar la diferencia entre la vida y la muerte de la empresa.

El manejo de la red tiene que ver con 3 niveles de hardware y software: . Nivel físico . Nivel sistema . Nivel aplicación.

El diseño modular de la orquitectura cliente-servidor permite rápidas modificaciones a los servidores especiales que traducen información entre sistemas. Debido a que estos filtros son demasiado compactos, las modificaciones se pueden hacer en cuestión de minutos. En el futuro los sistemas expertos serán capaces de detectar y hacer correcciones requeridas por cambios a la aplicación, automáticamente.



La Arquitectura Cliente-Servidor de Manejo de Redes.

Con esto se concluye que se puede construir un sistema de producción fuerte de industria en un período de tiempo asombrosamente corto.

1.2.10 Nueva Base: Enfrentando el Reto del Cambio.

Varias fuerzas están conduciendo a la computación y a las comunicaciones hacia la descentralización:
. Sistemas personalizados que son fácilmente desarrollados en pequeñas máquinas, pero que requieren conexiones a datos que residen en otros sistemas.
. Productividad en el desarrollo de software la cual es más alta en máquinas pequeñas para muchas aplicaciones.
. Nuevas tecnologías de procesadores que tienen que ser integradas con sistemas existentes.
Generalmente se tiene alguna porción del procesamiento de datos centralizada, pero es inevitable el que algún componente esté descentralizado.

Los cambios en tecnología incluyen: – Funciones de soporte de decisión que están siendo reemplazadas por sistemas expertos que realmente toman decisiones. – Computadoras tradicionales que están siendo superadas por procesadores paralelos que proparcionan 100 veces mayor rapidez de procesamiento.

La influencia de un solo vendedor en las organizaciones le está dando camino a las arquitecturas multi-vendedor con el advenimiento y aceptación de estándares de sistemas abiertos.

El usar sistemas de comunicación y sistemas operativos no-propietarios da la libertad de seleccionar con las mejores tecnologías, precios y características de performance de una amplia variedad de vendedores. Los sistemas propietarios le dan a los vendedores control de la empresa. Los sistemas abiertos ponen a la empresa de regreso al asiento de conducción.

Dos fuerzas manejan el comportamiento de compras de los gerentes de información: – Rápidos avances en desempeño y correspondientes disminuciones en costo de los componentes de la tecnología de información. – Movimiento de alcance industrial hacia estándares basados en especificaciones de sistemas abiertos.

La competencia le proporciona al comprador el lujo de ser selectivo. Las mejoras en la relación precio/desempeño fortalecen el descentralizamiento del cómputo. Los estándares de los sistemas abiertos se están volviendo una fuerza dominante en el mercado. La arquitectura cliente-servidor le permite a las compañías tratar al *mainframe* como un

servidor de archivos gigante. El cómputo de misión crítica puede ser distribuido entre más estaciones de trabajo y PCs de buen precio, cuyas operaciones están basadas en estándares.

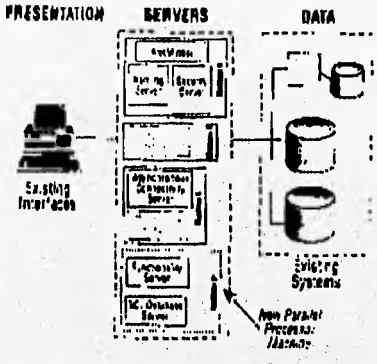
1.2.11 Una Mirada hacia Enfrente: Armas Tecnológicas del Futuro.

Cinco tecnologías emergentes para incluir en cualquier empresa son: la tecnología de objetos, los sistemas expertos, las computadoras capaces del procesamiento paralelo, las nuevas técnicas para las interfases de usuario y el cómputo colaborativo. Estas tecnologías pueden ser integradas fácil y económicamente con los sistemas existentes usando una arquitectura cliente-servidor cuando se usan herramientas ODE.

El cómputo orientado a objetos representa el futuro del desarrollo de software. El uso de la tecnología de objetos incrementa significativamente la productividad del programador y mejora el mantenimiento de programas. De esta manera los programas pueden ser construidos rápidamente reusando objetos existentes en nuevas formas.

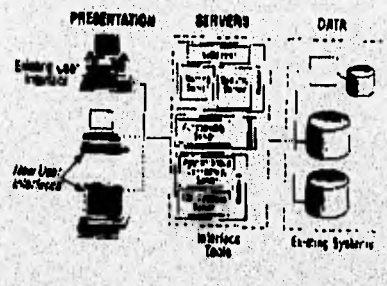
Los sistemas expertos tratan de modelar el juicio de los expertos humanos representando las técnicas de toma de decisiones del experto usando una colección del reglas "si-entonces". Desafortunadamente éstos sólo trabajan bien para ciertos problemas que pueden ser resueltos por estricta aplicación de decisiones si-entonces. Un sistema experto sin acceso a la mejor información nunca llevara a cabo todo su potencial. La arquitectura cliente-servidor puede proporcionar una manera flexible y confiable de obtener datos de sistemas existentes en la nueva aplicación del sistema experto.

Las computadoras de procesamiento paralelo proporcionan gran poder de cómputo para situaciones en las que la velocidad es crítica. Estas son más rápidas y económicas que cualquier otra máquina convencional. En lugar de hacer todo el trabajo en un cara procesador, el trabajo se divide entre procesadores unidos más simples los cuales trabajan concurrentemente. Se puede elevar la velocidad de procesamiento dramáticamente cuando una tarea puede ser dividida en más de una subtarea paralela. Las máquinas de procesamiento paralelo ofrecen gran velocidad usando tecnología convencional. Una vez más aquí la arquitectura cliente-servidor permite ligar la máquina de procesamiento paralelo y sus aplicaciones críticas en tiempo a las aplicaciones existentes con un mínimo de esfuerzo e interrupción. UNIX juega aquí un papel importante. Este sistema operativo minimiza el trabajo involucrado en portar aplicaciones a máquinas más rápidas y asegura el rango más amplio en opciones de vendedores.



Integración Tecnologías de Procesamiento Viejas y Nuevas.

Las interfases de usuario son la llave para usar los sistemas de información efectivamente. Estas se crearon debido a que las interfases convencionales entre los seres humanos y sus computadoras son terribles. De esta forma los días de las terminales tontas con sus hipnotizantes renglones de caracteres verde lima están contados. Las interfases de usuario gráficas son ya las clientes en la arquitectura de 3-capas. Estas interfases gráficas mejoran obviamente la comunicación entre las computadoras y los usuarios.



Integración de Nuevas Interfases de Usuario.

La tecnología de objetos, los sistemas expertos, las máquinas de procesamiento paralelo y las nuevas técnicas de interfase de usuarios mejoran de esta manera sin lugar a dudas el poder y la utilidad de los sistemas de información. Y al aplicar la arquitectura cliente-servidor a las aplicaciones estratégicas, se pueden integrar estas tecnologías, como ya se citó, de una forma fácil y económica con los sistemas existentes.

1.3 Estrategia Cliente/Servidor según Inmon.

1.3.1 Arquitectura en el Ambiente Cliente/Servidor.

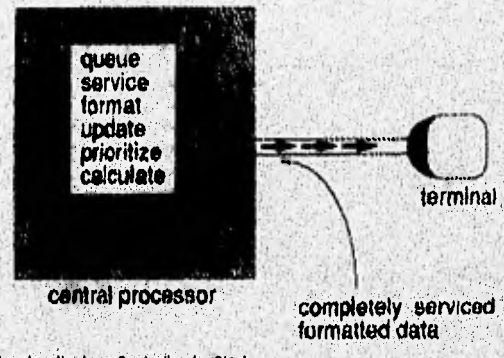
Arquitectura es preparar todos los cambios principales, y después construir la estructura en una secuencia en que las partes se conjunten en una forma congruente.

Cuando no se anticipan los requerimientos, el desarrollador produce una pesadilla en el mantenimiento y operación del sistema.

El que los sistemas cliente/servidor sean más chicos que los de tipo *mainframe*, no significa que sean más fáciles de construir o mantener. En algunos aspectos los sistemas cliente/servidor son más complejos que los de tipo *mainframe*. Por ésto, es muy importante tomar una metodología con arquitectura para el procesamiento y desarrollo cliente/servidor.

1.3.1.1 Procesamiento Cliente/Servidor–Las Bases.

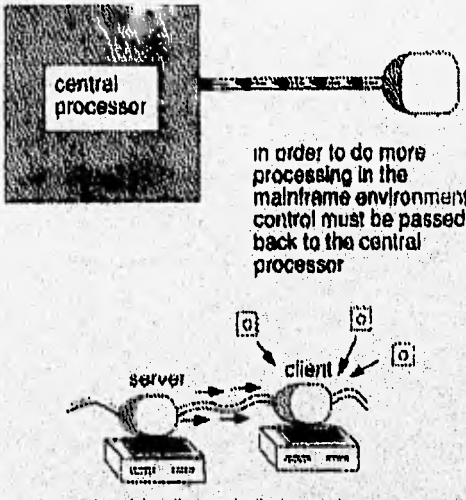
Los más importantes elementos de la arquitectura cliente/servidor se centran alrededor del procesamiento que ocurre dentro de la arquitectura y los datos almacenados que fluyen a través de la red misma. Indudablemente el desarrollador/diseñador necesita estar bien ligado con la tecnología en la que el ambiente cliente/servidor va a correr. Pero la atención apropiado del desarrollador no debe estar en la tecnología, sino en la(s) aplicaciane(s) que están siendo construidas y en las grandes estructuras de procesamiento y datos.



Una Arquitectura Centralizada Clásica.

Se debe de recordar que las arquitecturas centralizadas de tipo *mainframe* existen desde mucho antes que existiera el procesamiento cliente/servidor. En una arquitectura de este tipo todo lo que ocurre en la terminal es el desplegado de datos prefabricados y

preformateados, así como de colecciones de datos preformateadas. A diferencia de esta arquitectura, en la cliente/servidor una vez que los datos llegan a un nodo, éstos pueden ser recalculados, reformateados, reanalizados cualquier número de veces y formas, y combinados con otros datos sin regresar necesariamente al servidor.



Una vez que los datos llegan al cliente, pueden ser procesados muchas veces sin tener que regresar el control al servidor.

1.3.1.2 Costos.

Las costos de mantenimiento en la arquitectura cliente/servidor pueden ser menores porque las aplicaciones cliente/servidor usualmente son más pequeñas y más simples.

1.3.1.3 Uso de la Aplicación.

El uso del procesamiento cliente/servidor puede ser clasificado ya sea como uso operacional o uso DSS(Sistemas de Soporte de Decisiones).

El procesamiento operacional es aquél que afecta las decisiones de negocios diarias que son hechas y es conducido por los requerimientos. El procesamiento DSS es usado para soportar a la comunidad gerencial y es conducido por los datos. La más obvia y profunda diferencia entre estos dos tipos de procesamiento en el ambiente cliente/servidor, radica en la manera en que los recursos de hardware son usados.

1.3.1.4 Autonomía VS. Integración.

Uno de los grandes atractivos del ambiente cliente/servidor es el de autonomía de procesamiento, el cual es permitido por la independencia en cada nodo. Bajo todas las condiciones necesita haber un grado de libertad en cada nodo. Sin embargo, hay una necesidad acompañante de uniformidad y consistencia de datos y de procesamiento cuando los datos y/o procesamiento son "corporativos" los que parecen residir en un nodo pero son necesitados a través de la red. El no hacer la distinción entre datos autónomos, privados y datos corporativos, públicos así como el mezclarlos libremente, puede provocar graves problemas colectivos en el ambiente cliente/servidor.

1.3.1.5 Estructura de Datos.

La estructura de datos siempre ha sido importante en todos los ambientes de procesamiento. La primera gran división de la estructura de datos se lleva a cabo dentro del nodo mismo- qué datos son públicos (corporativos), y qué datos son privados debe ser claramente delineado. La segunda división de datos es entre los nodos operacionales y los de tipo DDS, de la red.

1.3.2 El Ambiente Cliente/Servidor-Algunos Elementos.

Hay varios grandes atractivos en el ambiente cliente/servidor: el costo de construir los sistemas, los costos de operarlos, y el control y autonomía que se permiten el desarrollador y el usuario en este ambiente. El elemento de individualidad y control hacen de este ambiente algo muy seductivo.

El "Performance" es el espacio de tiempo desde el momento en que un usuario inicia una petición hasta que ésta ha sido satisfecha y los resultados son obvios y utilizables. El Performance es un elemento en el ambiente cliente/servidor que el desarrollador enfrenta y lo comienza a ser cuando las aplicaciones cliente/servidor maduran.

Por otra parte para mantener control y disciplina sobre los nodos en la red cliente/servidor, debe ser definido un "sistema de registro", otro elemento importante.

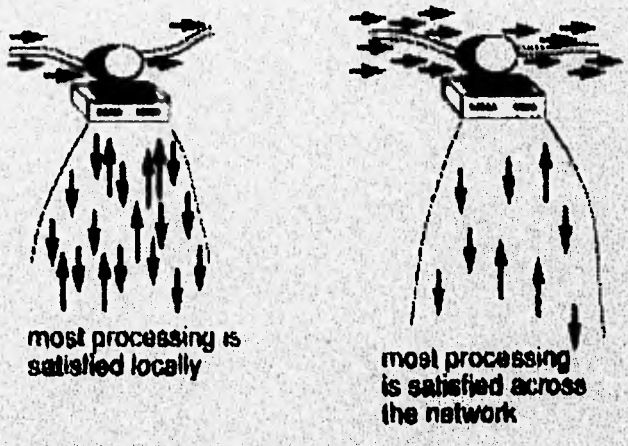
1.3.2.1 Datos de Valor Actual VS. Datos Archivados.

Los datos de valor actual son datos cuya precisión es válida por el momento de acceso. Son el reflejo de información o estado en ese momento. Los datos archivados son datos cuyo valor es relevante o exacto en algún momento de tiempo.

Hay un trato y procesamiento muy diferente de los datos en lo que se refiere al sistema de registro cuando se trata con datos archivados y datos de valor actual. Este es un elemento a considerar seriamente.

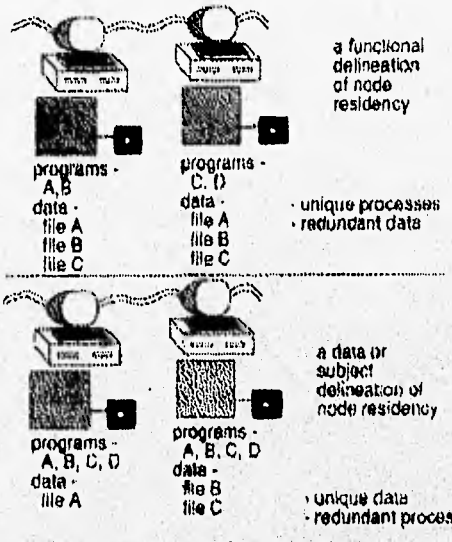
1.3.2.2 Residencia en Nodo -Un Importante Elemento.

Otro elemento de diseño muy importante en el ambiente cliente/servidor es el de residencia en nodo de datos y la probabilidad de acceso de datos. Cada una de las posibilidades para residencia en nodo tiene sus fortalezas y debilidades. Como una regla, la residencia en nodo es decidida por la orientación de la organización que controla el nodo.



Diferencia entre procesamiento local y procesamiento corporativo.

Cada uno de estos elementos que se acaban de mencionar, tiene un profundo efecto en la manera en que el ambiente cliente/servidor es moldeado y en la forma en que las aplicaciones son desarrolladas.



El cambio entre un diseño funcional de residencia de nodo y un diseño de datos o tema de residencia de nodo.

1.3.3 El Sistema de Registro y el Procesamiento Operacional/DDS.

El establecimiento e implementación del sistema de registro constituye una de las bases de un ambiente cliente/servidor exitoso.

1.3.3.1 El Sistema de Registro Operacional para el Procesamiento Cliente/Servidor.

El sistema de registro en este caso es definido por residencia de nodo, donde el control de actualizar reside precisamente en cada nodo. La residencia de nodo es definida por un alineamiento funcional con la organización. La razón de que la residencia de nodo sea naturalmente extendida a lo largo de líneas políticas o funcionales, es que la esencia del procesamiento cliente/servidor es de control y autonomía. Si la organización es dividida por las líneas de diferentes departamentos que realizan diferentes funciones, como contabilidad, mercadeo, etc., entonces la residencia de nodo es definida de esa forma. Si la organización tiene una división geográfica donde diferentes unidades geográficas están involucradas, entonces la residencia de nodo se realiza por las líneas de divisiones geográficas.

Una definición formal e implementación del sistema de registro se necesita si va a existir control e integridad del procesamiento corporativo cliente/servidor. ¿Cómo, exactamente, puede ser implementado el sistema de registro en un ambiente cliente/servidor residente en nodo orientado funcionalmente? Una manera es definir, basándose en el uso común de datos, que nodo es el nodo residente para qué datos.

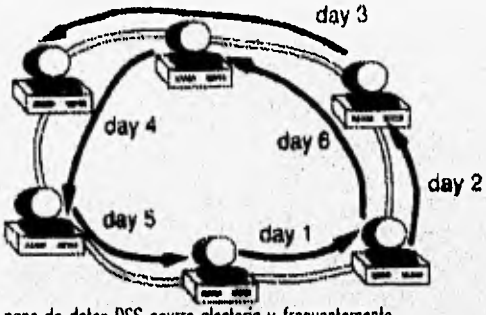
Otra alternativa es esparcir los datos a través de la red o en la configuración "pura" cliente/servidor para que la cuestión de residencia de nodo ante todo permita que los datos sean actualizados y accedidos en el nodo y sólo en el nodo en el cual su sistema de registro fue asignada.

Otra opción es crear un nodo que maneje todos los datos, y nodos separados que procesen los datos.

1.3.3.2 Sistema de Registro-Procesamiento DSS.

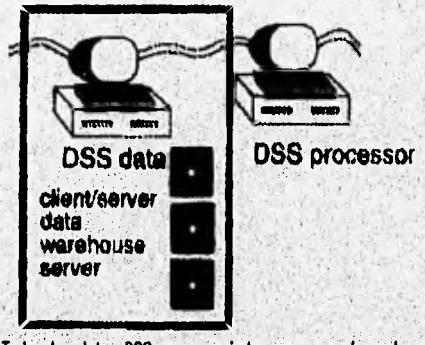
Existe una diferencia esencial entre el sistema de registro para datos operacionales y el sistema de registros para datos de tipo DSS: Los datos operacionales tienen valor actual y pueden ser actualizados, mientras que los datos variantes con el tiempo archivados, una vez que están registrados correctamente, no pueden ser actualizados.

Los factores que contribuyen a la necesidad de un sistema de registro para datos de tipo DSS son varios. Uno de ellos es el procesamiento analítico entre los diferentes conjuntos de datos. Otro es la carencia de una base de tiempo para los datos. Uno más son las múltiples niveles de extracción, cada uno de los cuales exagera las probabilidades de desintegración de datos. Los datos externos también contribuyen al problema de los datos de tipo DSS en el ambiente cliente/servidor sin un sistema de registro. Estos y la pérdida de la fuente de origen, entonces son factores adicionales del por qué el procesamiento de tipo DSS en un ambiente cliente/servidor sin un sistema de registro específicamente definido lleva a la falta de credibilidad.



El paso de datos DSS ocurre aleatoria y frecuentemente.

Una razón final para la falta de credibilidad del procesamiento de tipo DSS en un ambiente cliente/servidor sin un sistema de registro es que no hay fuente común de los datos con la que se pueda empezar.



Todos los datos DSS son manejados por un nodo en la red.

1.3.3.3 El Almacén de Datos del Sistema de Registro de tipo DSS.

La opción de usar un *mainframe* para el almacen de datos cliente/servidor es una excelente forma de realizar una transición suave entre el ambiente *mainframe* y el ambiente cliente/servidor.

A lo anteriar se suma que una característica de los datos del sistema de registro de tipo DSS –el almacén de datas– es que consumen mucho espocio de almacenamiento.

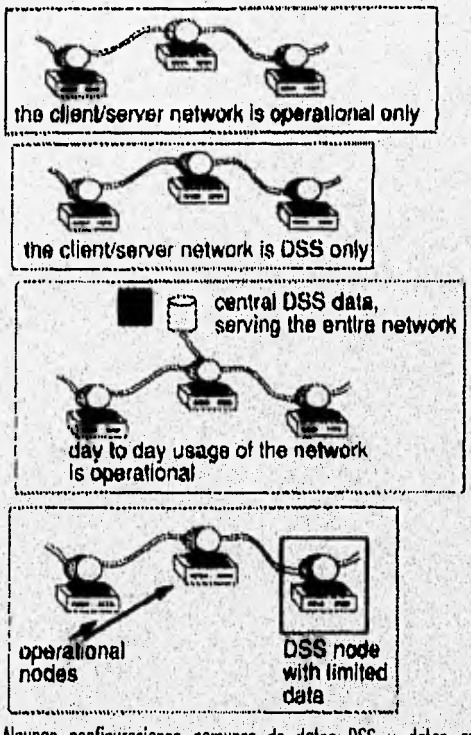
La granularidad de los datos es la primera consideración para canstruir el sistema de registro de tipo DSS – el almacén de datos cliente/servidar. La segunda consideración

es la de integración de los datos.

En general las características del almacen de datos cliente/servidor son:
. orientación a tema,
. integración,
. sin volatilidad una vez que la "foto" fue tomada y,
. variancia con el tiempo.

1.3.4 Configuraciones.

Para cumplir la necesidad de una separación de datos operacionales y de tipo DSS, existen múltiples opciones de implementación para el ambiente cliente/servidor.



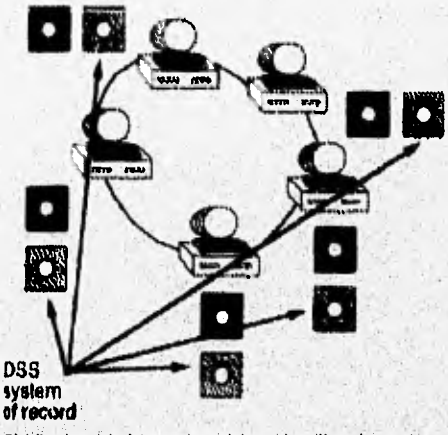
Algunas configuraciones comunes de datos DSS y datos operacionales y procesamiento en el ambiente cliente/servidor.

Hay muchas posibles configuraciones para el manejo de procesamiento operacional y de tipo DSS en el ambiente cliente/servidor. Algunas de las configuraciones más populares son:

. procesamiento operacional unicamente,
. procesamiento de tipo DSS unicamente,
. procesamiento DSS centralizado, procesamiento operacional distribuido, y
. procesamiento DSS descentralizado, procesamiento operacional distribuido.

Una configuración popular, es la red operacional cliente/servidor y el gran depósito central DSS del sistema de registro DSS -almacén de datos cliente/servidor.

Cada configuración como es obvio tiene sus ventajas y desventajas.



Distribución del sistema de registro entre diferentes nodos.

1.3.5 Performance en el Ambiente Cliente/Servidor.

El "performance" es también un tema importante en el ambiente cliente/servidor.

Aquí cabe mencionar que en lo que se refiere al mejoramiento del "performance" en un momento dado, dirigir ésto solamente en el nivel de hardware es un error tanto en el ambiente cliente/servidor como en cualquier otro ambiente.

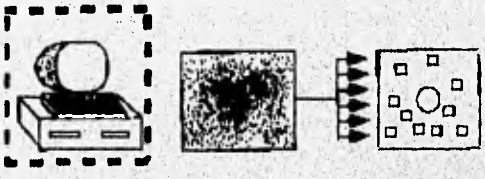
1.3.5.1 Manifestación del Problema de "performance".

Las formas en que los problemas de "performance" se presentan son:
- "Inundación" de la red debido a muchos pequeños mensajes mandados.

- "Inundación" de la red debido a muchos mensajes grandes mandados.

- "Encierro" de un nodo debido a acceso masivo de datos.

- "Encierro" de un nodo debido a bloqueado de datos.



Acceso masivo de datos.

Pero, ¿Qué se puede hacer para evitar estos problemas de "performance"?

Las siguientes son algunas sugerencias que deben ser aplicadas al momento de diseño:

. Las peticiones deben pedir siempre cantidades de datos limitadas.

. El espacio de tiempo por el que los datos son bloqueados debe ser mínimo.

. Es una buena práctica dividir el programa en programas más cortos.

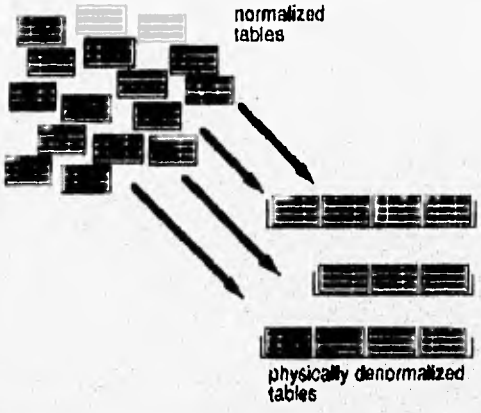
. Cuando un programa accede datos, éstos deben de estar indexados.

. Cuando un programa accede más de un registro, el acceso a los datos y el almacenamiento se debe de realizar mediante "buffers".

Entonces la estrategia para llevar a cabo altos niveles de "performance" en el ambiente cliente/servidor es realizar tan pocas entradas/salidas (I/O's) como sea posible.

Un enfoque general para organizar datos para el diseño de una base de datos es la práctica de la normalización de los datos. La normalización de los datos es una camino natural y bueno de formar las bases del diseño de bases de datos. Esta normalización trae consigo la desnormalización física, la cual puede llevarse a cabo creando selectivamente datos redundantes, o creando arreglos de datos dentro de un mismo "renglón", o esparciendo datos en tablas más pequeñas cuando hay una gran discrepancia en la

probabilidad de acceso de datos, o precalculando datos que son accedidos frecuentemente.



Agrupación de datos para fines de performance.

1.3.5.2 Otras prácticas de "performance".

Otra práctica obligatoria en el ambiente cliente/servidor es la de reorganizar los datos periódicamente en el disco duro.

Adicionalmente, de vez en cuando los datos necesitan ser purgados, de acuedo a algún calendario que se establezca.

1.3.6 Metadatos y el Ambiente Cliente/Servidor.

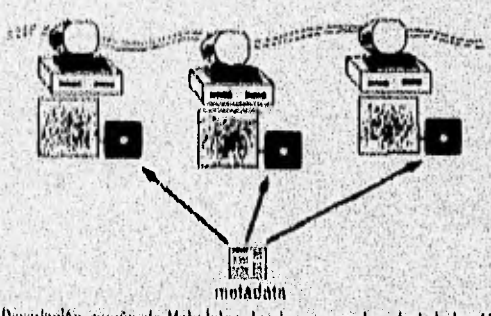
Los metadatos son datos acerca de datos. Estos son una parte esencial del mecanismo de control requerido para el ambiente cliente/servidor. El control de los metadatos es el pegamento que montiene junto al ombiente cliente/servidor. Los metadatos son necesarios paro el establecimiento y control del sistema de registro. Cuando la estructura de datos comunes permanece consistente sobre los diferentes nodos, el código escrito para un nodo es oplicable y completamente usable en otros nodos.

```
01 part file.
  02 partno              char (16),
  02 partdesc            char(30),
  02 qtyonhand           pic '99999',
  02 unitmeasure         char(1),
  02 lastdeldate         pic '999999',
  02 pctlanalyst         char(2),
  02 storeloc            char(1),
  ..........................................
  ..........................................
```

Ejemplo de Metadatos.

1.3.6.1 Almacén Central.

Es necesario tener un depósito común o central de metadatos para controlar datos compartidos comúnmente. Una buena práctica para ayudar que el ambiente esté en sincronía y así minimizar problemas es ponerles a los programas y al código una estampa de tiempo tanto al nivel de nodo como al de metadatos. Donde hay un ambiente servidor "puro", el servidor se convierte en un buen lugar para almacenar el software de control de los metadatos así como la administración.



Descripción común de Metadatos, la clave para el control de los Metadatos.

1.3.7 Una Metodología de Desarrollo Cliente/Servidor.

Una metodología significa dirigir al desarrollador a través de un camino lógico, señalando lo

que necesita hacer, en qué orden, y cuánto tiempo la actividad debe de tomar. En general hay una aceptación hacia las metodologías.

1.3.7.1 Dos Metodologías.

Existe una metodología diferente para los sistemas operacionales y otra para los de tipo DSS. Lo que se debe llevar a cabo en cada caso será expuesto a continuación. Lo que se requiere para conseguir los resultados esperados se deja completamente a cargo del desarrollador.

SISTEMAS OPERACIONALES CLIENTE/SERVIDOR.

M1 –Actividades de Inicio de Proyecto.

. Entrevistas. El resultado de las entrevistas es la descripción de lo que el sistema debe de hacer, reflejando usualmente la opinión de un manejo mediano.

. Recolección de Datos. Los requerimientos usualmente detallados que no son tomados en ninguna otra parte son recolectados aquí.

. JAD (Diseño Conjunto de la Aplicación). El resultado de una o más sesiones JAD es un conjunto de requerimientos formalizado que colectivamente representan las necesidades del usuario final.

. Análisis del Plan de Negocios Estratégico. Si la compañía tiene un plan de negocios estratégico, tiene sentido reflejar cómo el plan de negocios estratégico se relaciona con los requerimientos del sistema que se está diseñando.

. Los sistemas existentes le dan forma a los requerimientos para un nuevo sistema profundamente. El resultado de esta actividad es una descripción del impacto e influencia de los sistemas existentes en los requerimientos del sistema que se está desarrollando.

M2 –Medición, ajuste de fases.

La información de salida de este paso es la separación de los requerimientos generales en fases más manejables.

M3 –Formalización de Requerimientos.

El resultado de este paso es una definición de requerimientos que está lista para pasar a un diseño detallado.

PREQ1 -Definición de un Ambiente Técnico.
Aqui se define todo el ambiente técnico relacionado con el sistema a desarrollar.

D1 -ERD (Diagrama de Relación de Entidades).
La salida de este paso es la identificación de los temas importantes que compondrán al sistema, así como la relación de los temas entre si.

D2 -DIS (Conjuntos de Elementos de Datos).
Cada tema es más ampliamente descompuesta, en términos de nivel de detalle, en un dis.

D3 -Análisis de Performance.
La salida refleja un diseño mucho más perfilado.

D4 -Diseño de la Base de Datos Física.
La salida de este paso es la especificación real de la base de datos al DBMS (Manejador del Sistema de la Base de Datos).

P1 -Descomposición Funcional.
El resultado de este proceso es una gran descomposición funcional describiendo las diferentes actividades a ser realizadas de un nivel alto a uno bajo.

P2 -Nivel de Contexto 0.
Describe las actividades prioritarias a ser realizadas.

P3 -Nivel de Contexto 1-n.
Los niveles restantes de la descomposición funcional describen las actividades más detalladas que ocurren.

P4 -Diagrama de Flujo de Datos (dfd).
El dfd indica la entrada a un proceso, la salida de un proceso, el almacenamiento de datos necesitado para establecer el proceso, y una descripción breve del proceso.

P5 -Especificación del Algoritmo; Análisis de "performance".
El procesamiento actual es descrito.

P6 -Pseudocódigo.
La salida de esta actividad es la codificación de las especificaciones listas para el código real.

P7 -Codificación.
El resultado de este paso es código fuente.

P8 –Realización Superficial.

La salida de este paso es código que ha sido publicamente divulgado y que está tan libre de errores como ha sido posible.

P9 –Compilación.

El resultado de este paso es código compilado, listo a ser probado.

P10 –Prueba.

Hay varios niveles de prueba. La salida de este paso es código probado, listo para ejecución.

P11 –Implementación.

La salida de este paso (si en verdad hay un fin para este paso) es un sistema corriendo satisfactoriamente.

JA1 –Definición del Almacenamiento de Datos.

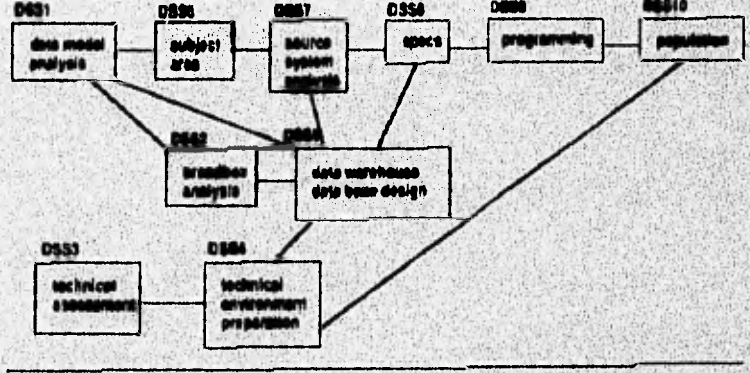
El resultado de este paso es una definición del almacenamiento de datos que satisface al diseño del proceso y de las datos.

GA1 –Revisión de Alto–Nivel.

La salida de este paso es un diseño de alto–nivel listo a proceder con los datos y con las rutas del proceso.

GA2 –Revisión del Diseño.

La salida de este paso es una evaluación del manejo de las fortalezas y debilidades del diseño, y recomendaciones para mejorar la calidad del sistema.



Desarrollo del "Warehouse".

DESARROLLO DEL SISTEMA DE REGISTRO DSS.

Este es un proceso totalmente dirigido por los datos. Se asume que el ambiente tecnológico ha sido establecido.

DSS1 –Análisis del Modelo de Datos.
La salida de este paso es una confirmación de que la organización ha construido un modelo de datos sólido.

DSS2 –Análisis de "Cajón".
Este análisis simplemente proyecta en terminos "rudas" cuánta información el sistema de registro DSS soportará.

DSS3 –Evaluación Técnica.
La salida de este paso es una definición de cómo el sistema de registro DSS se posicionará en el ambiente cliente/servidor.

DSS4 –Preparación del Ambiente Técnico.
Aquí se identifica técnicamente cómo la configuración puede ser acomodada.

DSS5 –Análisis del Area del Tema.
La salida de este paso es un alcance de esfuerzo en términos de un tema.

DSS6 –Diseño del Sistema de Registro DSS.
La salida es el diseño de una base de datos física del almacén de datos.

DSS7 –Análisis del Sistema Fuente.
La salida de este paso es el mapeo de datos del ambiente operacional al ambiente DSS.

DSS8 –Especificaciones.
La salida de este paso son las especificaciones de programa reales que serán usadas para traer datos del ambiente operacional al sistema de registro DSS.

DSS9 –Programación.
Este paso incluye todas las actividades estándares de programación.

DSS10 –Población.
La salida de este paso es un sistema de registro DSS funcional y "poblado".

DEPT1 –Repetición del Desarrollo Estándar.

IND1 –Determinar los Datos Necesitados.

La salida de esta actividad son datos seleccionados para un análisis más profundo.

IND2 -Programa para Extraer Datos.
Este siguiente paso consiste en escribir un programa para acceder y esparcir los datos

IND3 -Combinar, Mezclar, Analizar.
La salida de esta actividad son datos completamente utilizables para análisis.

IND4 -Análisis de Datos.

IND5 -Respuesta a Preguntas.

IND6 -Institucionalización.

1.3.8 Elementos en el Diseño de la Base de Datos en el Ambiente Cliente/Servidor.

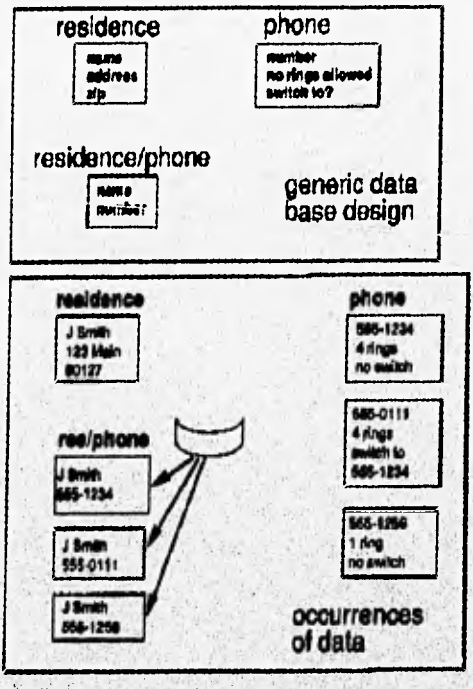
1.3.8.1 Manejando Datos Primitivos y Derivados.

Los datos primitivos son datos cuya existencia depende de una sola ocurrencia de un tema principal. Los datos derivados son datos cuya existencia depende de múltiples ocurrencias de un tema principal.

Los datos derivados pueden ser creados más rapidamente de lo que pueden ser modelados y diseñados.

1.3.8.2 Relaciones en el Ambiente Cliente/Servidor.

Uno de los aspectos más útiles de una base de datos es la capacidad para relacionar una unidad de datos con otra unidad de datos. La primera relación que se debe tomar aquí es cómo la relación será soportada. La siguiente decisión es cómo implementar exactamente la relación.

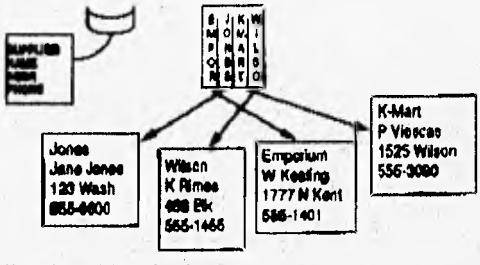


Diseño genérico y ocurrencias de datos.

La estructura llave para cada tabla en el ambiente cliente/servidor debe incluir una referencia al campo en el que se señale la residencia de nodo.

1.3.8.3 Indexación.

El método normal para acceder datos en un nodo es a través de un indice. Si los datos son frecuentemente accedidos pero rara vez actualizados, entonces el diseñador probablemente especifique muchos indices. Si los datos son frecuentemente actualizados pero rara vez accedidos, entonces el diseñador debe especificar muy pocos indices.



Un índice y datos primarios.

Por otra parte el resecuenciar datos a través de un índice puede presentar un problema si está acurriendo en el nodo demasiado procesamiento secuencial masivo.

Otra consideración acerca de los índices es que algunos campos nunca deben ser indexados, por ejemplo campos que sólo pueden tomar pocos valores.

1.3.8.4 Particionamiento de Datos.

El importante tema de particionamiento de datos se refiere a la habilidad de dividir grandes bases de datos en unidades físicas separadas y distintas. De esta forma los datos pueden ser esparcidos entre diferentes nodos y la residencia de nodo puede ser definida.

Cuando el particionamiento ha sido hecho propiamente, no debe de haber rotura al código existente cuando la división es hecha.

1.3.8.5 Codificación/Decodificación de Datos.

Una práctica en el diseño de una base de datos que merece atención es la de codificación/decodificación de datos.

Usada propiamente, la codificación puede ahorrar espacio y E/S (Entrada/Salida); usada impropiamente la codificación puede causar complicaciones y puede costar E/S's.

1.3.8.6 Datos de Longitud Variable.

La mayoría de los DBMS que corren en el ambiente pc/workstation permiten que los datos

sean definidos en términos de longitudes variables. El permitir que un campo sea variable en longitud puede ahorrar mucho espacio, en la circunstancia correcta. Sin embargo, si se permite la actualización de los datos de longitud variable, se puede producir una pesadilla de "performance". Debido a ésto, como una regla se deben de colocar los campos de longitud variable al final del registro, con los campos de longitud fija al frente del registro.

1.3.8.7 Recursión.

La recursión ocurre cuando un objeto tiene una relación con él mismo.

1.3.8.8 Micro/Macro Visión del Sistema.

Para que el sistema cliente/servidor funcione de una manera óptima, es necesario ver al sistema desde una "micro" asi como desde una "macro" perspectiva. Deben de existir ambas formas de entender al sistema cliente/servidor para que el diseño del sistema sea exitoso.

1.3.9 Diseño de Programas en el Ambiente Cliente/Servidor.

En muchas formas el diseño de programas en el ambiente cliente/servidor es muy similar al diseño de programas en otros ambientes.

1.3.9.1 Separación de Programas por Ambiente.

Es muy importante para el programador identificar el ambiente particular en el que se está construyendo un programa dentro del ambiente cliente/servidor porque la estructura, uso, orientación, código, y otras cosas cambian de un ambiente a otro.

Una categoría importante es la de código común a través de todos los nodos contra el procesamiento autónomo de nodo. La segunda importante categoría es la de procesamiento DSS contra procesamiento operacional.

Es importante que el programador entienda para qué categoría su programa ha sido escrito.

1.3.9.2 Respeto para la Residencia de Nodo.

Cualquier información cuya sistema de registro cae fuera de un nodo no puede ser cambiada. Es decir desde una perspectiva de código, ántes de que cualquier actualización de datos pueda ser hecha, los datos deben de ser residentes de nodo.

1.3.9.3 Sensibilidad de Nodo/Insensibilidad.

El código es sensible del nodo cuando "corre" de una forma en un nodo y de otra en otro nodo. El código es insensible del nodo cuando "corre" de la misma forma en todos los nodos.

Un fuerte argumento en favor de la insensibilidad para código común es que los nodos futuros puedan ser agregados a la red con poco o sin ningún impacto en el código.

1.3.9.4 Performance.

El performance a través de la red es efectuado como un resultado de muchos factores de diseño, no sólo del diseño del programa. El diseño de la Base de Datos, la definición adecuada de la residencia de nodo, la expandibilidad de la red, y otros aspectos, contribuyen al performance así como al diseño del programa.

1.3.9.5 Estandarización.

La necesidad (y la oportunidad) de la estandarización es común a través del ambiente cliente/servidor.

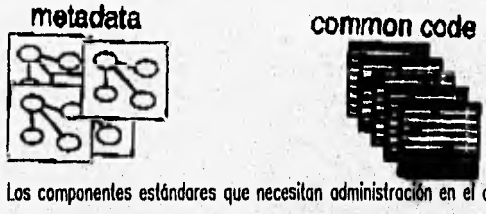
Algunos de los consideraciones para la estandarización son:

- Registro de tiempo(fecha de instalación) del código común.
- No repetibilidad de código.
- Cada módulo representa una función única y separada.

1.3.10 Administración del Ambiente Cliente/Servidor.

Hay dos aspectos del ambiente cliente/servidor que hacen al trabajo de la administración más importante que en ambientes más grandes –el deseo de autonomía en cada nodo y el tamaño de la red, el cual es considerablemente más pequeño que el del ambiente *mainframe*.

La administración entonces, es un componente muy importante en el éxito del ambiente cliente/servidor. Hay dos aspectos principales que deben ser administrados: la red misma y los metadatos y código común que soportan los aspectos "corporativos" del ambiente cliente/servidor.



Los componentes estándares que necesitan administración en el ambiente cliente/servidor.

El probar los programas en producción es otra tarea de la administración. El monitoreo es así mismo otra tarea importante del administrador.

1.4 Estrategia Cliente/Servidor segun Vaskevitch.

Hoy en día los programadores diseñan y construyen grandes aplicaciones orientadas a negocios usando herramientas y metadologías desarrolladas en los últimos 30 años. Aunque muchas de esas herramientas y metodologías son aún apropiadas, claramente nuevos elementos se requieren. Los tecnólogos deben basar los sistemas del mañana en uno estructura conceptual diferente de aquellas de los sistemas del pasado.

1.4.1 Una Estructura Conceptual para el Futuro.

La estructura que ha evolucionado para construir las aplicaciones que comúnmente están corriendo hoy en día es fundamentalmente inadecuada. Sólo cuando la estructura existente sea clara se podrá saber como puede ser aumentada y cambiada para tratar con el futuro.

Un diseño físico se refiere a computadoras, discos, Bases de Datos, líneas de comunicación y otros elementos concretos del mundo real. Un diseño lógico se refiere a la estructura de la aplicación independiente del tipo o lugar del hardware, software y datos. Un buen diseño lógico hace posible el cliente/servidor; sin ese diseño lógico, los sistemas distribuidos son casi imposibles no importando cómo se lleguen a acomodar las "cajas".

Una buena arquitectura de aplicación proporciona una estructura para pensar en, diseñar, construir y dividir aplicaciones que encajan juntas y trabajan bien.

Al diseñar una aplicación, el primer y más importante paso es asegurar que la aplicación realmente satisface las necesidades del usuario. La primera tarea es entender qué es lo que se supone que la aplicación debe hacer.

Las aplicaciones complejas requieren dos capas: una para la aplicación del equipo "desktop" y una para la Base de Datos. De esta arquitectura, algunas personas pueden concluir que el *mainframe* es una Base de Datos, lo cual es entendible. Muchos servidores son ya sea servidores de archivos o servidores de Bases de Datos. Los servidores se conectan a los *mainframes* primariamente para acceder Bases de Datos. Las organizaciones mantienen las aplicaciones en el *mainframe* para un mejor desempeño de la Base de Datos, integridad de la Base de Datos y funcionalidad de la misma. Sin embargo un *mainframe* es más que sólo una Base de datos. El *mainframe* "corre" reglas de negocios, siendo así un lugar donde programas en batch pueden vivir felizmente.

En la mayoría de los grandes *mainframes* aproximadamente la mitad del procesamiento se dedica a la Base de Datos, la otra mitad del tiempo de la computadora se reserva a las aplicaciónes, considerando éstas como programas de software que tienen que ver estrechamente con cómo una organización emplea su tiempo. Las organizaciones de todos los tamaños gastan su tiempo en complejos procesos del negocio que consisten de muchos pasos en un periodo largo. Tomadas individualmente, las aplicaciones del *mainframe* completan tareas específicas. Tomadas conjuntamente, las aplicaciones del *mainframe* forman la máquina que hace que los procedimientos del negocio trabajen. Por lo tanto, el papel principal y verdadero de un *mainframe* es correr los procesos del negocio.

Idealmente los procesos de negocio deben de ser completamente automáticos, sin pausas y sin intervención humana. No obstante el tener a las organizaciones corriendo completamente de una manera automática, sin intervención humana, no es aún deseable. El punto es no efectuar automatización total, sino en su lugar eliminar colas no necesarias,

aprobaciones no necesarias e intervención humana no necesaria. Aquí las computadoras hacen su parte, dejando a la gente las decisiones que realmente se benefician de la intervención humana.

Cuando los procesos se vuelven más automáticos, más crítica es la necesidad de que una máquina coordine, secuencie y se asegure de la correcta operación de todos esos procesos multipaso automáticos. El *mainframe* o su reemplazo se convierte en algo más críticamente importante en el futuro con reingeniería de lo que fue en el enfoque a tarea.

Debida a que los *mainframes* pueden manejar procesos complejos en períodos prolongados de tiempo, la automatización de los procesos del negocio es posible. El *mainframe* es la transmisión automática de la automatización de los procesos del negocio.

El papel principal de las aplicaciones que corren en un *mainframe* es implementar reglas del negocio. Los reglas del negocio son todas las miles de reglas que definen cómo el negocio corre. Los procesos del negocio, los mismos de los que la Reingeniería de Procesos de Negocio trata, son reglas del negocio; al final un proceso es sólo una colección de estas reglas del negocio.

Las reglas pueden ser aplicadas todas de una vez, o pueden ser aplicadas en varias secuencias sobre períodos largos de tiempo. Los procesos que ocurren en períodos largos de tiempo se dice que ocurren en tiempo extendido.

La arquitectura de las aplicaciones de hoy en día no proporciona algún lugar "hogar" explícito para las reglas del negocio. Las nuevas reglas del negocio caen entre las capas superior e inferior, las dos capas que se han manejado hasta hace no mucho.

En la parte superior está la capa de aplicación desktop. La responsabilidad de la aplicación desktop es entendimiento y eficiencia. La función de esta capa es proporcionar la interfase de usuario a todo el sistema. En general la capa de aplicación desktop es responsable de la navegación, presentación, manipulación y análisis. Esta capa transforma a la computadora personal en un escritorio electrónico. El software en la capa de aplicación desktop se construye con herramientas gráficas.

La capa de reglas del negocio es responsable de implementar las políticas de la organización. El software en esta capa está basado en heurística. La heurística es una guía que está frecuentemente orientada bajo términos probabilísticos. De esta forma esta segunda capa es responsable de las reglas y de la heurística. Las reglas y la heurística son implementadas en la forma de decisiones.

La capa del manejo de la Base de Datos es responsable de mantener información consistente y segura. Así el principal propósito de esta capa es asegurar la seguridad de los datos. Así mismo, debe de hacer los datos disponibles sólo a aquellos que tienen derecho a

verlos. Esta capa también asegura el significado de los datos que almacena. El diseño de esta capa debe asegurar respuestas consistentes a preguntas en el momento y en el futuro.

Esta arquitectura de tres capas es una arquitectura lógica, y no física. Esta misma arquitectura de tres capas puede ser usada para construir sistemas distribuidos –pero también sistemas centralizados. Esta arquitectura es una mejor forma de construir aplicaciones –ya sea que sean distribuidas o no. Sin embargo, esta arquitectura hace más fácil distribuir componentes de aplicaciones, si es necesario por razones de negocios.

Una arquitectura física habla acerca del diseño físico de un sistema. Una arquietctura lógica es una forma de pensar acerca de la amplia estructura de la aplicación dejando muchas opciones acerca de cómo será construida físicamente. El punto principal de una arquitectura lógica es capturar los grandes elementos de todo el diseño dejando muchas de las opciones particulares abiertas a propósito.

La arquitectura que se está desarrollando es una mejor forma de diseñar aplicaciones.

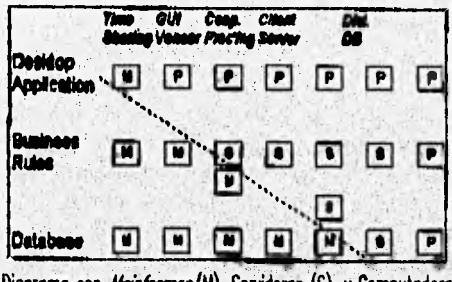


Diagrama con *Mainframes* (M), Servidores (S), y Computadoras Personales (P).

De esta forma la arquitectura de tres capas es mucho más que una forma de hablar acerca de sistemas distribuidos. La belleza del modelo de tres capas es que trabaja sin importar las decisiones que se toman después acerca de qué va dónde, de cuál arquitectura física es la mejor, y de cuántas capas de computadoras habrá.

La clave de sistemas distribuidos efectivos es el diseño de aplicacianes efectivo. Las aplicaciones diseñadas efectivamente trabajan bien en configuraciones centralizadas, configuraciones distribuidas, y en todo lo que hay enmedio. El usar una arquitectura de aplicaciones coherente es una de las estrategias centrales para diseñar aplicaciones y sistemas efectivos.

1.4.2 Arquitectura de la Aplicación: Una Mejor Manera de Diseñar Aplicaciones.

La arquitectura de tres capas proporciona los siguientes beneficios en proyectos de aplicaciones: 1. Una estructura para construir aplicaciones muy flexibles que pueden ser cambiadas facilmente para satisfacer las cambiantes necesidades del negocio. 2. Un alto nivel de reuso de saftware. 3. Un desarrallo más rápido de aplicaciones grandes y complejas que pueden sapartar altos niveles de desempeño en ambientes tanto de soporte de decisiones camo transaccionales. 4. Desarrollo más fácil de aplicaciones distribuidas que soportan equipos autamanejados y manejados centralmente.

Debido a que los grandes sistemas son cualitativamente diferentes de las pequeños, se deben de usar técnicas completamente diferentes para efectivamente diseñar, construir y mantener grandes sistemas.

La primera cuestión que queda clara cuando se piensa acerca de grandes aplicaciones contra pequeñas tiene que ver con qué tan difíciles son las aplicaciones de construir. Las aplicaciones que se vuelven difíciles de construir con las herramientas disponibles son las que se consideran grandes; el diseñarlas y construirlas requieren un gran esfuerzo.

La diferencia entre una aplicación grande y otra pequeña es sarprendentemente difícil de definir. La linea que divide a los sistemas grandes de los pequeños es borrosa, en el mejor de los casos. No obstante, dos cosas están relativamente claras. Primero, los sistemas grandes son necesarios en organizacianes de tamaño grande y mediano. Segundo, los sistemas distribuidos son intrinsicamente más complejos que las sistemas centralizados. Pronto el movimienta de la industria hacia los sistemas cliente servidor forzará a tener que ver con sistemas más grandes que tengan interfases de usuario gráficas.

Generalmente, las sistemas grandes son camplejas. Por lo tanto, las sistemas grandes son también difíciles de desarrollar y difíciles de saportar. Consecuentemente, arquitectura de aplicacianes de tres-capas ya expuesta está basada en la siguiente filosofia: lo grande es difícil -y para ser manejable, lo grande debe hacerse pequeño. Divide y canquista. Esa es la premisa atrás de la arquitectura de tres capas la cual también se basa en el hecho de que ninguna persona sola puede comprender, construir o mantener un sistema muy grande.

Cuando se desarrollan nuevos sistemas, se debe de encontrar la manera de dividir un proyecto de una aplicación grande en una serie de pequeños proyectos, cada uno entendible por sí mismo. Para ésto hay dos mecanismos para dividir y conquistar grandes aplicaciones: abstracción y encapsulamiento. La abstracción es una técnica analítica para dividir un sistema en varias capas de detalle. El encapsulamiento es el praceso de combinar información y camportamiento en una nueva entidad llamada un objeto.

Una abstracción es una descripción que identifica las propiedades esenciales de un objeto mientras esconde los detalles concretos. Un principio central de un buen diseño de un sistema es desarrollar una arquitectura que proporcione varios niveles de abstracción. Usando abstracción, se pueden esconder muchos detalles que no se necesitan en un nivel particular en la aplicación. El reducir la cantidad de detalles visibles en un momento dado es una parte crítica de describir y diseñar sistemas complejos. La técnica primaria para esconder detalle, es definir los niveles de abstracción en una aplicación.

Una arquitectura de capas basada en abstracción tiene tres características principales: 1. Capas definidas claramente: Las capas se deben de construir una sobre la otra, proporcionando cada capa una abstracción más compleja y sofisticada que las capas de abajo. 2. Interfases formales y explícitas entre las capas: Cada capa debe de poder llamar los servicios de las capas inferiores. Las interfases definen las operaciones de alto nivel que activan el comportamiento más detallado implementado por la capa de abajo. Al definir las operaciones que están disponibles a la capa de arriba, la interfase especifica exactamente qué aspectos de las funciones dentro de la capa son visibles y disponibles al mundo exterior. 3. Detalles escondidos y protegidos dentro de cada capa: El esconder detalles es una parte importante de tanto la abstracción como del encapsulamiento.

Niveles de Abstracción en la arquitectura de aplicación de 3-Capas.

La arquitectura de aplicaciones de tres-capas protege deliberadamente cada capa de los detalles contenidos en cada capa inferior. Hay dos beneficios principales de usar abstracción para proteger cada capa de los detalles de las otras capas: - Desarrollo de aplicaciones simplificado: Un desarrollador de una capa tiene un trabajo más simple porque no necesita conocer o pensar acerca de los detalles de la capa inferior. - Protección y seguridad superior: El desarrollador en una capa no puede físicamente controlar a la capa inferior en ningún nivel de detalle.

En la arquitectura de tres capas el centro de la aplicación es la capa del manejo de la base de datos. Esta capa es la parte del sistema que puede literalmente existir por si misma. Los procesos de reglas del negocio dependen fuertemente en la capa del manejo de la base de datos en esta arquitectura. Esta capa hace lo siguiente: - Realiza las tareas requeridas. - Hace decisiones. -Lanza otras tareas en esta capa así como en otras.

La capa de aplicaciones "desktop" puede ser comparada a los gerentes. Para realizar esta tarea los gerentes usan formas predefinidas (aplicaciones) y piden la ejecución de procesos de reglas del negocio. Cada petición de un gerente ejecuta acciones (procesos de reglas del negocio) y archiva elementos (transacciones de la base de datos).

La interfase define la apariencia externa de un componente de la arquitectura. Una interfase realiza las siguientes funciones: – Le dice a la computadora que hacer. .Cuestiana acerca del estado actual de los componentes. – Recibe los resultados de las operaciones pedidas. En un mundo orientado a objetos, la interfase se vuelve muy poderosa. El punto central de la orientación a objetos es el encapsular datos y programas que operan en esos datos en un objeto discreta. Consecuentemente la ùnica forma de trabajar con esos datos es a través de la interfase al objeto correspondiente.

Interfases en la arquitectura de aplicación de 3-Capas.

Uno de los elementos clave que se repite par toda la arquitectura es la importancia de mantener las tres capas independientes una de la otra. Para hacer los procesos de las aplicaciones desktop y de las reglas de negocios independientes unos de los otros se debe evitar que los procesos de las reglas del negocio hablen a los usuarios. De esta manera la capa del manejo de la base de datos debe trabajar sin importar la que son las reglas del negocio. Aùn las transacciones en la interfase de la base de datos no debe de contener ninguna regla del negocio.

La capa del manejo de la base de datos es responsable de proporcionar datos seguros y consistentes permitiendo acceso a esos datos sólo a través de un conjunto de transacciones y consultas definidas.

Las transacciones juegan tres papeles criticos: 1. Actualizaciones consistentes: Las transacciones aseguran que cada petición es completada totalmente o es tratada de otra forma como si no hubiera sucedido. El diseñar sistemas de forma que todas las actualizaciones sean implementadas sòlo a travès de transacciones asegura la consistencia de la base de datos. 2. Cumplir las reglas principales del negocio: Las transacciones de la base de datos cuidadosamente checan y refuerzan las reglas del negocio principales. 3.

Prevenir cambios no autorizados o inválidos a la base de datos: En un sistema bien diseñado, las transacciones son la única forma de hacer cambios a la base de datos.

La parte transaccional de la interfase de la capa del manejo de la base de datos asegura que todas las actualizaciones son llevadas a cabo de una manera protegida, segura y consistente.

La función de la interfase de consultas en la capa de manejo de la base de datos es proporcionar a los usuarios una forma entendible y consistente de tratar con las complejas colecciones de registros que colectivamente definen a la base de datos corporativa. Consiste de un conjunto de consultas diseñado especialmente que extrae datos de la capa de manejo de la base de datos correcta y consistentemente. Las consultas en la interfase de consultas sirven para tres funciones principales: 1. Simplificar uniones complejas: La interfase de consultas proporciona una vista global en la que muchos archivos se reducen a un número muy pequeño, en donde cada uno reúne información de muchos de los archivos más elementales que contienen los datos reales. 2. Asegurar consistencia: El punto principal de la capa de consultas es asegurar que excepciones a la regla se manejan correctamente y de una manera transparente a la aplicación o al usuario que está preguntando. 3. Seguridad: Consultas apropiadamente construidas pueden asegurar qué información sensible esté disponible a sólo aquellos usuarios y aplicaciones a los que se les ha permitido ver esa información.

Mantener una Base de Datos consistente es la función más importante de la capa de manejo de la Base de Datos. Una Base de Datos consistente asegura que todos los usuarios llegarán a las mismas respuestas cuando ataquen cierto problema.

El mezclar reglas del negocio con transacciones de la Base de Datos es un gran problema. Para evitar esto se debe de realizar el diseño de la Base de Datos independiente de las reglas del negocio implementadas en algún otra lugar del sistema.

Las reglas del negocio como ya se ha indicado expresan las políticas organizacionales. Si se incorporan políticas del negocio en el diseño de la Base de Datos, entonces la Base de Datos se desactualizará cuando las políticas cambien. Los registros y los campos en una Base de Datos deben ser independientes de las reglas del negocio y políticas actuales.

Para construir una Base de Datos que sea independiente de las reglas del negocio se debe de: 1. Diseñar cuidadosamente la Base de Datos usando un modelo de datos bien planeado. 2. Desarrollar consultas y transacciones que proporcionen un buen acceso a la Base de Datos: Se deben de usar transacciones que permitan acceso a la Base de Datos pero que a su vez refuercen reglas básicas de consistencia. Se puede desarrollar un manejador de transacciones y consultas que presente una vista más detallada de los datos. 3. Permitir solamente transacciones bien diseñadas para actualizar la Base de Datos: Se

debe de hacer a las transacciones el guardian de la integridad de la Base de Datos 4. Aislar a los usuarios de los detalles y localizaciones de las Bases de Datos fundamentales: Diseñando las transacciones formales y las consultas correctamente, la interfase de la Base de Datos se puede comunicar con una variedad de diferentes Bases de Datos en diferentes computadoras sin que el usuario o la aplicación se enteren de lo que está sucediendo.

El construir Bases de Datos de esta forma lleva a la visión original de la Base de Datos en si: la lógica cambia pero los datos no.

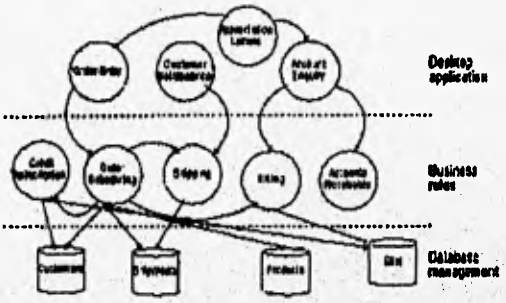
El propósito de mantener una capa de reglas del negocio independiente es la interoperabilidad. La interoperabilidad es la habilidad de compartir trabajo, software y hacer cosas en una forma consistente por toda la compañía.

El equivalente computacional de los procedimientos duplicados es el software duplicado. Comunmente el código no se reusa o comparte entre diferentes aplicacianes. La mayor razón de esto es la falta de una separación observada cuidadosamente entre la interfase del usuario, las reglas del negocio y el manejo de la Base de Datos, todo esto en el mismo programa.

La arquitectura de tres-capas limpiamente separa las reglas del negocio de las aplicaciones desktop y de las Bases de Datos. Sin embargo este no es el caso en la mayoría de los sistemas existentes.

Las reglas del negocio están lejos de ser las únicas funciones en una aplicación de un gran *mainframe*. Típicamente más de la mitad del código de una aplicación de un *mainframe* trata con funciones diferentes de las reglas del negocio. La solución a los problemas que se presentan en este tipo de plataforma es dividir las reglas de los procesos del negocio en unidades de código reusables y modulares.

El separar la capa de reglas del negocio y la capa del manejo de la Base de Datos hace la independencia de la capa de reglas del negocio completo. El pago a esto es una combinación de reusabilidad de código e interoperabilidad, esto es flexibilidad. La interoperabilidad se refiere a tomar una pieza de código que manipula una tarea y hacer que ese código realice la misma función para todas las aplicaciones a lo largo de la organización. Es la interoperabilidad la que hace el reuso de código posible. Así que el beneficio de hacer a la capa de aplicación desktop independiente de las otras capas es mejorar la libertad y flexibilidad de los usuarios –y de la organización. Hay que recordar aquí que los usuarios siempre quieren ver, introducir y manipular información en su propia forma.



Separación de la capa de reglas del negocio de la capa de manejo de la base de datos para lograr interoperabilidad enlas aplicaciones.

La arquitectura de tres-capas proporciona una interfase formal entre la capa de aplicación desktop y los procesos de reglas del negocio fundamentales. Una aplicación desktop manda peticiones de procesos formales que provocan que los procesos de reglas sean ejecutados. La aplicación desktop puede hacer cualquier cosa que el usuario pida - tanto antes como después de mandar una petición de proceso.

Al hacer cada capa de la aplicación independiente de las otras capas, se puede mejorar la calidad de las aplicaciones en cuatro formas: 1. Mejores Bases de Datos: Se mantiene la Base de Datos independiente de los cambios de la política del negocio y se mantienen datos consistentes y con significado en grandes períodos de tiempo. 2. Interoperabilidad: Al separar reglas del negocio de la interfase del usuario y de actualizaciones detalladas de la Base de Datos, se hacen las reglas del negocio reutilizables. De esta forma la interoperabilidad al nivel de reglas del negocio es finalmente posible. 3. Flexibilidad y libertad: Al hacer las aplicaciones desktop verdaderamente independientes de las reglas del negocio más importantes, la aplicación cumplirá mejor con las necesidades de los usuarios. De esta forma se pueden cambiar y mejorar los procesos de reglas del negocio sin afectar la interfase de usuario. 4. División de Labores: Debido a que la arquitectura de la aplicación proporciona un esquema para dividir el trabajo, se puede juntar un equipo de profesionales de la computación con diferentes áreas de especialidad y construir un sistema.

Aún para aplicaciones destinadas a permanecer en el *mainframe*, la arquitectura de tres-capas proporciona un muy buen modelo para interoperabilidad mejorada, consistencia de Bases de Datos, flexibilidad, y división de trabajo en un proyecto.

1.4.3 Diseñando Sistemas Distribuidos: Procesos contra Bases de Datos.

Los elementos centrales en los sistemas distribuidos son los servidores. Estos se encargan de ejecutar esos mismos procesos de reglas del negocio que de otra forma se ejecutarían en el *mainframe*.

El hecho de que las reglas del negocio son difíciles de cambiar, particularmente dentro de las computadoras, hace que las organizaciones sean rígidas y que los procesos de reingeniería sean difíciles de implementar. Una condición fundamental que hace esto difícil es la inflexibilidad real y percibida en los sistemas computacionales. Esa inflexibilidad invariablemente produce implementaciones rígidas de las reglas del negocio. De esta forma un paso importante en el camino a los procesos con reingeniería es poner una infraestructura que permita que las reglas de la organización sean mucho más flexibles.

Un problema en ésto es que un *mainframe* obliga a la organización entera a vivir con una sola implementación centralizada de todas las reglas. La solución a esto es poner esas reglas en servidores locales lo cual permite a cada oficina, departamento, y equipos a madificar y extender las reglas y procesos de la organización para satisfacer las necesidades locales. Debido a que el servidor es físicamente seguro, se puede confiar en él para reforzar la parte no cambiable de las reglas mientras se sigue permitiendo que las otras partes sean adaptadas.

El concepto de tener servidores distribuidos ampliamente, cada uno con su Base de Datos residente y ejecutando procesos de reglas del negocio personalizados a las necesidades del equipo local es atrayente. El problema aquí es que historicamente nadie ha podido diseñar las Bases de Datos que este modelo de realidad implica, considerando que de inicio diseñar una Base de Datos es una meta ya compleja así como lo es diseñar aplicaciones grandes.

La mayoría de los programadores son artesanos profesionales que dependen en técnicas reales y probadas para construir software. Una metodología es un esquema prescrito para diseñar y construir aplicaciones. Describe una serie de pasos, modelos y técnicas que si son seguidas cuidadosamente seguramente conducirán a aplicaciones que trabajan bien. Las metodologías proporcionan la estructura organizacional que permite a los grandes equipos de desarrollo funcionar de una manera coordinada. Aquí la calidad es crítica en términos de tanto evitar problemas como de asegurar que los requerimientos son verdaderamente satisfechos. Las metodologías ayudan a resolver los problemas de escala proporcionando a los que trabajan en un proyecto vocabulario común, técnicas comunes y esquemas de control para medir progreso.

Las metodologías proporcionan predicción y control. Por ejemplo, en Andersen Consulting cada consultar tiene un firme conocimiento en Methad/1, la metodología de Andersen. Esta metodología prescribe la secuencia de pasos, los datos a ser reunidos en

cada etapa, el criterio para completar "milestones", las decisiones a hacer antes de escoger entre técnicas de diseño alternativas, los estándares nombrados para variables, y cualquier otro detalle que se pueda presentar cuando se construye una aplicación. Esto es, control, predicción, y consistencia.

El construir software es sin lugar a dudas una importante parte del negocio, pero de una forma importante, no el centro del mismo. Uno de los más grandes problemas que cualquier organización puede enfrentar es el riesgo asociado con los proyectos de desarrollo mayores. Aqui entra la metodologia. Sin embargo ésta no es un substituto del pensar, no es un reemplazo de un buen diseño ni tampoco algo mágico que simplificará el desarrollo o resultará en trascendentalmente mejores aplicacianes. En el corazón de cualquier metodología descansa primero la arquitectura de la aplicación y después un conjunto de estrategias de diseño.

Según una organización desarrolla una variedad de aplicacianes en el transcurso del tiempo, el hacer a todas convivir es un problema. Esto apunta a la necesidad de integración de aplicaciones.

La solución de la Ingenieria de Información (IE) a este tipo de problemas empieza con el desarrollo de un Plan del Sistema de Información (ISP), el cual cubre la mayoría de las aplicaciones que una organización espera desarrollar en algún periodo de tiempo razonable. El proceso ISP también tiene que ver con elementos de la planeación clásica incluyendo los Factores de Exito Criticos, las estructuras organizacionales, y otros. El ISP proporciona una vista global de toda la estructura juntando todas las aplicaciones de la organización. Basado en el ISP, el portafolio de aplicaciones potenciales de la compañía es dividido en una serie de Areas de Negocio.

Después de una priorización, las áreas de negocio son analizadas a mayor detalle, una por una, en un proceso llamado en el mundo de la IE, Análisis de las Areas del Negocio (BAA).

Posteriormente, aplicaciones en particular son seleccionadas para diseño (Diseño de Sistemas de Negocios, o BSD) y construcción subsecuente (Construcción de Sistemas de Negocios, o BSC). La IE entonces pasa a describir técnicas que incluyen pruebas, versiones, y mantenimiento de aplicaciones.

De esta forma el primer elemento de la técnica de la IE es la definición de una técnica paso a paso para desarrollar aplicaciones integradas basadas en una planeación arriba-abajo (top-down). La contribución primaria de la IE descansa en aplicar estas técnicas probadas a la planeación, diseño, y desarrollo de software computacional. La IE también describe un conjunto de modelos y diseña estrategias para pensar y modelar aplicaciones. Los modelos y el modelado son realmente importantes en el elemento del diseño distribuido.

En la planeación de la IE y en el proceso de diseño, en cada etapa del proceso, planeación, diseño y construcción se procede considerando dos aspectos ligados de la aplicación: el modelo de datos y el modelo funcional.

En los primeros años de los '80s según el desarrollo manejado por modelo, las metodologías y las herramientas CASE empezaron a ser populares, los modelos de datos y el diseño de Bases de Datos hicieron su aparición. Desde entonces por dos decadas el modelado de datos ha sido colocado en un pedestal en el centro del proceso de planeación estratégica. Con el descubrimiento del modelado de datos, las Bases de Datos se convirtieron en no sólo una forma de pensar acerca de aplicaciones, sino en una manera de diseñarlas también.

Una de las técnicas primarias usada para considerar el lado funcional del esquema de la aplicación es llamada descomposición funcional. La descomposición funcional es el equivalente exacto de dividir procesos complejos en tareas muy pequeñas.

Después de 20 años de progreso incremental parece ser que la Base de Datos no es el enfoque más importante en este mundo basado en computadoras orientado a la información. La Reingeniería es el verdadero centro ahora del nuevo mundo de la información. La Base de Datos continua ahí, así como con el proceso, pero es el proceso de hecho el que viene primero y en muchos sentidos maneja todo el diseño. De la misma forma el modelo de datos continua siendo importante y aún juega un enorme papel influenciando el modelo del proceso de desarrollo. El modelo del proceso proporciona un nuevo conjunto de diagramas y una valiosa herramienta para pensar y trabajar en el plan del sistema de información a largo plazo y en el diseño de aplicaciones de alto-nivel. Esta es la misma herramienta y modelo que maneja a la Reingeniería de Procesos del Negocio (BPR).

En un mundo centrado en los datos el primer paso hacia un modelo distribuido es asegurarse que el modelo de datos es correcto. Aquí el problema es que el modelo de datos por definición no es correcto ya que después de todo no es distribuido.

En un diagrama modelo de datos clásico en ningún lugar normalmente se representa el lugar físico de los datos. Después de todo con sólo un modelo de datos, no hay base para escoger una estrategia de distribución efectiva. Esto es la razón por la que por mucho tiempo, mucha gente ha considerado a las aplicaciones distribuidas imposibles de diseñar.

Al distribuir el procesamiento, el de los datos sigue. Los datos son requeridos precisamente por los procesos distribuidos que a su vez han sido distribuidos a los servidores donde los procesos se ejecutan.

1.4.4 Una Metodología Cliente/Servidor.

Después de más de una decada de duro trabajo, diversas metodologías han surgido las cuales brindan métodos controlados para diseñar y construir aplicaciones. La madurez y preparación que han alcanzado se basa en la estabilidad y madurez de la Ingeniería de Información. Las metodologías actuales necesitan cuando menos ser modificadas, y en el peor de los casos ser reemplazadas.

Muchas desarrolladores que se basan en computadoras personales, en especial los que no han conocido otro tipo de computadora en toda su vida, creen que las metodologías de hoy en día no están solamente mal, sino que son realmente innecesarias.

Las aplicaciones pequeñas, es decir aquellas que corren completamente en un solo equipo de escritorio, pueden ser construidas en minutos o en horas. La metodología solo se quedaría en el camino. Las metodologías de tamaño mediano, las cuales giran alrededor de un sola servidor y de docenas de estacianes de trabajo, se pueden construir casi tan rápidamente como las pequeñas.

Cada doce meses, el tamaño y complejidad de las aplicaciones que pueden ser construidas con herramientas modernas, da un paso hacia adelante. Como resultado, muchas aplicaciones serias pueden ser construidas rápida y fácilmente en un ambiente libre de metodalogías. De hecho esta es la forma en que las aplicaciones pequeñas deben de ser construidas.

Pero, ¿Qué hay de las aplicaciones más grandes? Los sistemas grandes son cualitativamente diferentes de los pequeños. Hay una barrera de complejidad. Cuondo esa barrera se cruza, las métadas que trabajaran antes, no solo trabajan más pobremente, sino que dejan de trabajar totalmente.

Lo acabado de exponer dice tres cosas: 1. Las metodalagías son y van a continuar siendo requeridas para la construcción de aplicaciones grandes. De hecho, las metadologías y las estrategias de diseño que contienen, hacen posible en primer lugar el desarrollo de esos grandes sistemas. 2. Los sistemas distribuidos en particular, necesitan más que otros estas metodologías. 3. Las metadalagías de hoy en día tienen que cambiar para cumplir las necesidades de una nueva generación de desarrolladores y usuarios, para acomadar el diseño de las sistemas distribuidas, y para producir de una manera amistosa sistemas fáciles de mantener.

Históricamente, la organización de desarrollo central construye los aplicaciones completas. Los usuarios se confrontan con la proposición de "tómalo o déjalo". En el futuro, la organización central construirá bases de datos basadas en modelos de datos sólidos, módulos de procesamiento de reglas del negocio básicos, y herramientas fundamentales para construir aplicaciones de tipo "desktop". Todo ésto tomará la forma de

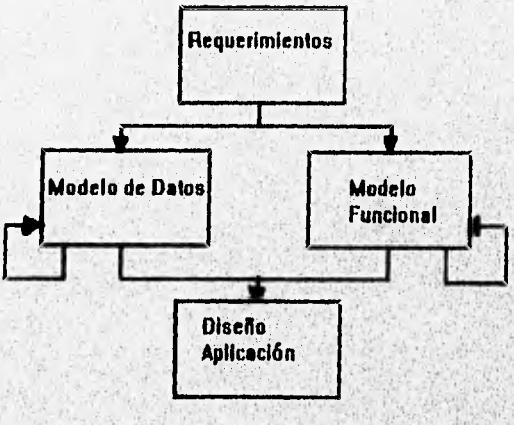
infraestructura.

Eventualmente, las aplicaciones ya no se construirán por un grupo central y tampoco se construirán más desde el inicio. En lugar de ésta existirá un rico ambiente de construcción y diseño.

Se tiene que cambiar substancialmente la forma actual de trabajar. Se tiene que facilitar la transición a nuevas formas de trabajo.

La metodologías clásicas sufren de varias problemas, los cuales tienden a producir aplicaciones monolíticas (difíciles de dividir en pedazos), centralizadas, y problemáticas de usar.

El siguiente modelo de proceso captura la esencia del proceso de diseño clásico. Girando alrededor de los dos solitarios modelos de datos y función, el proceso de diseño tiene problemas importantes:

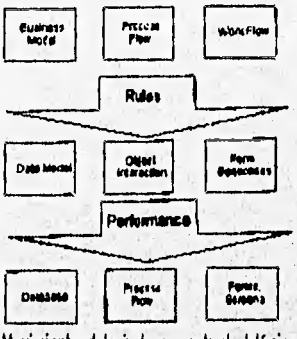


Para desarrollar un nuevo armazón para diseñar y construir aplicaciones, se necesita un modelo basado en la arquitectura de la aplicación.

Planeación, diseño, y desarrollo de aplicaciones.

El modelo conceptual nos muestra como se verá la nueva aplicación, como cambiará el negocio y si requerirá nuevos procesos de negocio. El modelo lógico prueba que la nueva aplicación puede realmente trabajar. El nivel físico es donde la aplicación realmente se escribe.

Casi todas las herramientas de hoy en dia del mundo de las computadoras personales operan casi enteramente al nivel físico. La diferencia fundamental entre el modelo conceptual y el lógica es la necesidad de empezar a poner atención a las reglas del negocio detalladas.



Movimiento del nivel conceptual al lógico y al físico en el diseño de una aplicación.

Un problema de la actualidad es que menos del 5% de la programación en la mayoría de los sistemas de software grandes, consiste de código compartido común. En este sentido, se puede comentar, que aproximadamente cada 10 años algún nuevo avance de consideración en tecnología de empaquetado llega y promete solucionar el problema del reuso.

Un avance fue la encapsulación. Su importancia consiste en que el usuario está en la posición de modificar implementaciones, sin tener que cambiar el programa que las llama en absoluto.

Después surgieron los procesos. Un proceso es un programa separado, independiente, que se mantiene por separado por el sistema operativo, el cual se comunica con otros procesos/programas, a través de mensajes. El concepto de proceso es esencialmente el fundamento, en software, para toda la arquitectura cliente/servidor.

Tal y como las subrutinas proporcionan las bases para los lenguajes de programación modernos, los procesos proporcionan los bloques de construcción fundamentales con los que virtualmente todos los sistemas operativos, redes, y ambientes distribuidos, se construyen.

Los procesos brindan un modo adecuado de encapsular sistemas ejecutándose completos. El único detalle es que los procesos, por ellos mismos, no proporcionan ningún mecanismo para describir los mecanismos disponibles para interactuar con ellos.

Posteriormente nacieron los paquetes. Estos en el contexto de grandes sistemas orientados a la defensa, ayudaron a dar otro gran paso hacia adelante en el desarrollo de software reutilizable. De hecho, los paquetes proporcionaron el trabajo clave para los objetos de hoy en día.

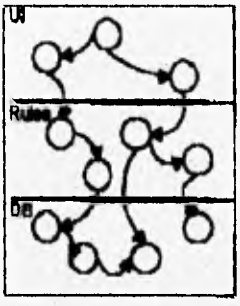
Los objetos son el último avance de una serie, en un periodo de diez años, en que se han producido cada vez mejores técnicas de empaquetamiento. Un objeto es fundamentalmente un paquete más generalizado. Un proceso es un objeto porque hace todo lo que se espera que haga un objeto. Los procesos por sí mismos dan la infraestructura fundamental para implementar sistemas de tipo objeto pero hay algunos pedazos críticos que faltan para hacer del empaquetamiento de verdad tan conveniente como los sistemas de objetos dictan.

Los procesos juegan un papel clave en el mundo de la reingeniería. Conjuntos de reglas de negocio definen a un proceso. Los procesos en el mundo técnico encapsulan compartamiento y datos. Sin embargo, los procesos también carecen de algunos elementos críticos que son importantes. Por ejemplo, no hay estructura formal para describir las interfases que un proceso soporta. Algo más importante es que los procesos son una construcción sin mucho diseño. En especial no hay esquemas comunes que hablen de como los procesos trabajan entre sí. Aquí es donde los objetos se presentan.

El análisis orientado a objetos mapea requerimientos en conjuntos de objetos, con interfases definidas formalmente. Este diseño toma el modelo conceptual que viene de la fase de análisis, toma en cuenta reglas del negocio en detalle, y lleva un conjunto de objetos más detallados que cooperan entre sí a través de interfases definidas formalmente (métodos).

Para propósitos de diseño de sistemas cliente/servidor distribuidos, la mayoría de los beneficios de la orientación a objetos pueden ser conseguidos sin un programación totalmente orientada a objetos.

Los procesos son objetos desde una perspectiva de implementación. Los objetos son procesos desde una perspectiva de diseño. Si se ponen a los dos juntos, una solución completa surge. Esto quiere decir que se usen los objetos para conceptualizar los procesos del negocio así como sus interacciones. Después se deben de usar los procesos para implementar esos objetos cuando es tiempo de construir el sistema.



Descomposición de una aplicación sencilla.

Dos simples mecanismos, la encapsulación de objetos y el dividir una aplicación en capas, llevan a un método completo para cortar aún sistemas muy grandes a un tamaño razonable.

Para maximizar la oportunidad de éxito, un proyecto de desarrollo grande debe de coordinarse con principios básicos como el que la organización entera debe de construirse alrededor de equipos pequeños, ésto es equipos con los que se tiene mayor control, que premian y promueven la interoperabilidad y la flexibilidad.

Adicionalmente a los equipos de desarrollo normal, dos equipos especiales son muy importantes en organizaciones de desarrollo grandes. El equipo de arquitectura que especifica la estructura general de desarrollo y el equipo de herramientas el cual se encarga de evaluar las herramientas y de desarrollar estándares apropiados para su uso.

1.4.5 Herramientas para Construir Sistemas.

Hay dos reglas cardinales que ayudan a evaluar herramientas: 1. Las herramientas de desarrollo son para humanos, no para computadoras. 2. Ninguna herramienta, sin importar la buena que sea, puede hacer todo el trabajo para todas las aplicaciones. El entender y ganar experiencia con una variedad de herramientas es la mejor forma de asegurar el que se estén construyendo aplicaciones de la manera más eficiente.

Todo el software se basa en el mismo pequeño conjunto de elementos. Algunos desarrolladores producen grandiosas aplicaciones y algunos no. La diferencia está en el diseño.

El gran diseño viene de grandes diseñadores. Pero aún los grandes diseñadores necesitan herramientas y procesos para hacer su trabajo posible.

El proposito de un modelo es permitirle al diseñador experimentar con muchos posibles diseños sin tener que construir un sistema final completo en cada ocasión. El papel central de un modelo es funcionar como una representación a escala que representa otro objeto más grande.

El prototipo le permite al diseñador, en cuestión de minutos u horas, desarrollar un modelo de la interfase de usuario que se ve, para fines prácticos, exactamente igual que el producto final.

Sin embargo a diferencia de la capa de interfase gráfica, los modelos usados para describir las reglas del negocio y las capa de la base de datos, son muy diferentes del producto final. No hay representación visual, ni imagen concreta en la que se pueda basar el modelo. En lugar de ésto, el modelo se basa en diagramas algunas veces complementados con descripciones basadas en palabras.

Un modelo de datos no es solo una útil representación para un usuario, es además una fuerte plataforma en la que se puede basar el desarrollo posterior.

Una combinación de diagramas puede ser usada para modelar la aplicación y estos a su vez pueden ser modelados mediante prototipos.

Los diagramas tienen un valor intrínseco alto aún cuando éstas no llevan directamente a código. No son muy rigurosos y están diseñados para proparcionar perspectivas de visualización del diseño de una aplicación. No obstante, el modelo mediante diagramas es un método poderosa, probado por muchos años, que ayuda a producir y refinar mejores diseños. Así mismo el que los diagramas no sean parte de algún esquema científico matemático para la construcción de programas, no significa que no sean por ésto valiosos.

Las herramientas CASE son hoy en día demasiado importantes para el diseño efectivo de sistemas grandes. Estas herramientas juegan un papel fundamental para la mayor parte de la planeación también. Actualmente las herramientas CASE son o complejas o caras o difíciles de usar o lo contrario, limitadas, baratas y de cualquier forma difíciles de usar.

Todos los proyectos tienen tres componentes: el diseño, la programación y la documentación. Estas tres componentes tienen una estrecha relación y cuentan con diferentes y apropiadas herramientas para cada uno de ellos.

Los diseñadores de software necesitan herramientas de diseño. Actualmente los desarrolladores luchan cuando tratan de compartir, editar y preservar el diseño expresado en diagramas. Debido a ésto urgen nuevas herramientas de diseño. Cuando éstas se desarrollen, muy probablemente serán herramientas de diseño construidas para simplemente

facilitar el desarrollo de una variedad de modelos usados para describir y analizar sistemas antes de que sean construidos.

No obstante a lo poco prometedor que se ve el futuro en relación a lo que se acaba de citar, se debe de continuar invirtiendo en software de diseño. Aquí lo importante es estar comprometido con el diseño, los modelos, y los diagramas. De la misma manera en que las herramientas de diseño evolucionan, no es necesario esperar por la última y más sofisticada herramienta. Hay que recordar que aún en el ambiente de herramientas perfecto del futuro, ninguna herramienta hará siempre todo el trabajo. El diseño es crítico pero posible, y para ello se cuenta con ayuda basada en computadora para hacerlo realidad.

Por el lado de la programación, cabe aquí resaltar que las aplicaciones pequeñas que se ejecutan completamente en equipo "desktop" generalmente deben de ser construidas con un solo lenguaje de programación. Las aplicaciones de tamaño mediano, que consisten de un interfase gráfica con una base de datos, pueden ser escritas en dos lenguajes: uno para la interfase gráfica y otro para que manipule las consultas y transacciones de la base de datos. Las aplicaciones grandes, es decir las que consisten de tres capas, llegan a utilizar dos o tres lenguajes de programación, junto con quizas un mayor soporte de un lenguaje más técnico. Se debe de recordar que la metodología de capas hace fácil que los diversos componentes escritos en diferentes lenguajes se comuniquen entre si, sin tener que preocuparse por el lenguaje de implementación.

Las herramientas de diseño y análisis son críticas y ayudan a juntar grandes aplicaciones. En este proceso el almacenar toda la información en una base de datos central donde pueda ser compartida por herramientas de diseño o construcción de todos tipos es vital. Esto se puede lograr mediante una repositorio. Un repositorio es una base de datos que contiene el modelo que describe todas las aplicaciones en un sistema así como las descripciones de las partes que se encuentran en las aplicaciones. Adicionalmente el repositorio mantiene el control de las relaciones entre todas esas partes. Es un repositorio el que hace posible que una variedad de diferentes herramientas puedan trabajar juntas. Es importante destacar que este repositorio debe de ser abierto y extendible.

1.5 Análisis comparativo y Evaluación de las Estrategias Cliente/Servidor según Donovan, Inmon y Vaskevitch.

Se definieran una serie de parámetros o elementos de comparación para llevar a cabo esta tarea. Estos son los siguientes: diseño o modelado, implementación, depuración,

mantenimiento, tiempo de desarrollo, software relacionado, tipo de sistemas a los que se aplica la metodología, diagramas que utiliza o en que se apoya y sustentación o bases teóricas. Estos fueron seleccionados por considerarse claves para la entrega exitosa de soluciones en negocios de integración de sistemas así como por ser parte sin lugar a dudas de lo que debe de ser un ambiente de negocios común donde el compartir recursos se facilita notablemente. Algunos de estos parámetros no son contemplados por todas las metodologías, no obstante aún con esto se incluyen por considerarse importantes en la sustentación de la metodología correspondiente.

Para empezar, en lo referente al diseño Donovan plantea que las funciones de la aplicación se separan en 3 capas: presentación, funcionalidad/conectividad/servidores de bases de datos y los datos en sí. En esta etapa del desarrollo de una aplicación se involucran de una manera importante a los usuarios de la misma asegurando así máximo uso y eficacia de la aplicación tanto aquí en el diseño como en la implementación. El diseño en esta metodología es modular. Inmon menciona que en esta fase de la metodología que propone la atención debe de estar en las aplicaciones que se están construyendo y no en la tecnología relacionada. Aquí se recomienda pensar en realizar tan pocas operaciones de entrada/salida así como en normalizar los datos, todo esto en pro del performance de la aplicación. Un punto negativo para esta metodología es el hacer diferencia de un programa cuando este cambia de un ambiente particular a otro. Vaskevitch nos dice que el diseño lógico es fundamental. Se hace especial énfasis en que la aplicación debe satisfacer cabalmente las necesidades del usuario. El diseño efectivo es clave en los sistemas distribuidos. En los sistemas grandes a diferencia de los pequeños se deben de usar técnicas totalmente diferentes tanto en el diseño como en la construcción y mantenimiento de la aplicación. La ingeniería de información mediante el desarrollo de un Plan del Sistema de Información da una solución a la necesidad de integración de aplicaciones y describe técnicas que soportan el análisis, diseño/modelado, construcción, pruebas, versiones y mantenimiento de una aplicación.

En la siguiente etapa que es la implementación Donovan comenta que con la metodología que él delinea se ponen juntos a todos los sistemas, existentes y adquiridos, en sólo algunos meses mientras que las adquisiciones futuras se integran en sólo días. Esta integración es barata y no requiere reconstruir las Bases de Datos de los sistemas viejos. Aquí cualquier aplicación puede acceder cualquier dato siempre y cuando se cuente con el servidor apropiado. La instalación de un sistema de éstos se realiza en piezas permitiendo así que no haya interferencia con las funciones normales de los sistemas existentes. Así mismo como esta construcción se realiza por paquetes de piezas cada una de éstos se puede ver corriendo individualmente sin la necesidad de que todo el sistema final tenga que ser completado. Inmon nos dice en esta fase que el "performance" es un elemento crítico en el ambiente cliente/servidor que el desarrollador enfrenta. De la misma manera elementos como datos de valor actual contra datos archivados y la residencia en nodo tienen un profundo efecto en la manera en que las aplicaciones son desarrolladas.

Vaskevitch simplemente recuerda aquí que las tres capas de la aplicación en desarrollo, deben de ser independientes una de la otra.

La depuración sólo Donovan hace cierta mención de élla en el sentido de que nos hace ver que ésta es prácticamente nula ya que se va presentando interactivamente durante la implementación.

En el mantenimiento de la aplicación desarrollada según lo expuesto por Donovan se permiten realizar modificaciones de una manera muy fácil según las cambiantes circunstancias que cualquier negocio dicta, esto gracias entre otros factores a la modularidad empleada. Inmon aquí nos habla de que los costos de mantenimiento son menores ya que las aplicaciones que se manejan son más pequeñas y más simples. En beneficio del performance se recomienda reorganizar los datos en el disco duro. A su vez se debe de llevar a cabo aquí también la administración de la red y de los metadatos así como un monitoreo de toda la aplicación.

Los temas de tiempo de desarrollo, software relacionado, y tipos de sistemas a los que se aplica la metodología, únicamente son abordadas por Donovan. En lo relacionado a tiempo de desarrollo se dice que una aplicación estratégica que se convertirá después en un sistema de producción se llega a construir en no más de 5 meses en lugar de los 1 a 6 años en los que una aplicación se desarrolla con una metodología tradicional. En software relacionado se nos hace notar que esta metodología está basada en sistemas abiertos estándares de industria. Existe un conjunto de herramientas específico para desarrollar aplicaciones bajo esta arquitectura llamado Ambiente Distribuido Abierto (ODE). Los tipos de sistemas a los que se aplica la metodología de Donovan tienen alcance de cualquier organización o empresa y se pueden aplicar a casi cualquier aplicación estratégica.

Los diagramas son solamente considerados de una forma explícita por Inmon. Nos dice que en la metodología de los sistemas operacionales se maneja un diagrama de relación de entidades en el diseño así como un diagrama de flujo de datos.

Finalmente cada una de las tres metodologías tienen sus bases teóricas que las sustentan. La metodología planteada por Donovan, donde él mismo fue pionero en establecer los bases de la misma, está soportada por el Grupo de Tecnología de Cambridge (CTGroup). Se considera a ésta como un medio y no un fin en la toma y procesamiento de la información de dentro y fuera de la organización así como en la presentación de la misma de una forma accesible a la gente correcta. La arquitectura que se maneja es adaptable al cambio del negocio y de la tecnología. A su vez ésta trae consigo velocidad y flexibilidad en la implementación debido entre otros factores a que los cambios que se llegan a necesitar se realizan en unidades discretas. La metodología de Donovan proporciona la integración requerida sin duplicar los esfuerzos encapsulados en los sistemas existentes. En la metodología expuesta por Inmon el procesamiento Cliente/Servidor se clasifica como uso operacional y uso DSS. El establecimiento e implementación del sistema

de registro es una base aquí. Los metadatos son una parte esencial del mecanismo de control. Aquí se plantea una metodología para los sistemas operacionales y otra para los de tipo DSS. No obstante sólo se exponen los pasos que se deben de llevar a cabo y no cómo realizarlos. Las Bases de Datos son vitales en el particionamiento de los datos. Los sistemas se ven aquí desde una micro como desde una macro perspectiva. Vaskevitch nos habla primero, entre otras ideas importantes a resaltar, de la arquitectura Cliente/Servidor de 2 capas haciendo mención que ésta fue requerida por aplicaciones "complejas" que acompañaron el nacimiento de la misma. En ésta el *mainframe* jugó un papel fundamental. Después aborda la arquitectura Cliente/Servidor de 3 capas la cual la expone como una forma mejorada de construir aplicaciones, diciendo de ésta que es una arquitectura lógica. Así mismo nos dice que las aplicaciones grandes se deben de hacer pequeñas con ayuda de la abstracción y del encapsulamiento proporcionando así interoperabilidad, consistencia de la Base de Datos, flexibilidad y división de trabajo. Las Bases de Datos que son consideradas aquí como el centro de la aplicación deben de ser independientes de las reglas y políticas del negocio. Los servidores son los elementos centrales en los sistemas distribuidos. La reingeniería es el centro del nuevo mundo de la información.

De acuerdo a lo que se acaba de mencionar se puede observar que Donovan nos presenta una metodología madura, cuya funcionalidad y adaptabilidad a las exigentes necesidades de los ambientes de aplicación de hoy en día, la hacen una excelente opción para tomarla de modelo, como se hará en el presente trabajo, para el confiable desarrollo de una aplicación de nivel empresarial de carácter crítico. Así mismo esta metodología y el software con el que se desarrollará la aplicación que se presenta como caso práctico de este trabajo, son un complemento magnífico por todas las características prácticas que el producto de software posee las cuales son en su mayoría no más que la implementación de lo que la teoría nos dice. Inmon nos enseña una forma demasiado teórica de ver el Cliente/Servidor y en sólo 2 capas las cuales ya no son suficientes para los cada día más complejos sistemas que se requieren. Por último Vaskevitch nos da una muy completa idea de lo que ha sido y es todo lo que está alrededor y en el desarrollo de una aplicación. Sin embargo su exposición carece de una tendencia profunda que se le pueda identificar con algo bien definido y claro.

de registro es una base aquí. Los metadatos son una parte esencial del mecanismo de control. Aquí se plantea una metodología para los sistemas operacionales y otra para los de tipo DSS. No obstante sólo se exponen los pasos que se deben de llevar a cabo y no cómo realizarlos. Las Bases de Datos son vitales en el particionamiento de los datos. Los sistemas se ven aquí desde una micro como desde una macro perspectiva. Vaskevitch nos habla primero, entre otras ideas importantes a resaltar, de la arquitectura Cliente/Servidor de 2 capas haciendo mención que ésta fue requerida por aplicaciones "complejas" que acompañaron el nacimiento de la misma. En ésta el *mainframe* jugó un papel fundamental. Después aborda la arquitectura Cliente/Servidor de 3 capas la cual la expone como una forma mejorada de construir aplicaciones, diciendo de ésta que es una arquitectura lógica. Así mismo nos dice que las aplicaciones grandes se deben de hacer pequeñas con ayuda de la abstracción y del encapsulamiento proporcionando así interoperabilidad, consistencia de la Base de Datos, flexibilidad y división de trabajo. Las Bases de Datos que son consideradas aquí como el centro de la aplicación deben de ser independientes de las reglas y políticas del negocio. Los servidores son los elementos centrales en los sistemas distribuidos. La reingeniería es el centro del nuevo mundo de la información.

De acuerdo a lo que se acaba de mencionar se puede observar que Donavan nos presenta una metodología madura, cuya funcionalidad y adaptabilidad a las exigentes necesidades de los ambientes de aplicación de hoy en día, la hacen una excelente opción para tomarla de modelo, como se hará en el presente trabajo, para el confiable desarrollo de una aplicación de nivel empresarial de carácter crítico. Así mismo esta metodología y el software con el que se desarrollará la aplicación que se presenta como caso práctico de este trabajo, son un complemento magnífico por todos las características prácticas que el producto de software posee las cuales son en su mayoría no más que la implementación de lo que la teoría nos dice. Inmon nos enseña una forma demasiado teórica de ver el Cliente/Servidor y en sólo 2 capas las cuales ya no son suficientes para los cada día más complejos sistemas que se requieren. Por último Vaskevitch nos da una muy completa idea de lo que ha sido y es todo lo que está alrededor y en el desarrollo de una aplicación. Sin embargo su exposición carece de una tendencia profunda que se le pueda identificar con algo bien definido y claro.

2. Breve Panorama de las Bases de Datos y su relación con el modelo Cliente/Servidor.

Existen tres tipos de productos de base de datos (bd): 1. Los que le permiten al usuario primariamente trabajar en su escritorio con colecciones de registros. En este nivel, las bases de datos le permiten al usuario trabajar con bases de datos privadas. 2. Los que le permiten a muchos usuarios compartir una bd y 3. Los que están construidos para el único propósito de facilitar el compartir de la bd.

El modelo de bd le permite a muchos usuarios trabajar inclusive en el mismo archivo al mismo tiempo; la unidad que se comparte aquí es el registro de la bd. En una bd debido a que la información se organiza en registros, se arregla que cada registro sea cambiado por sólo una persona a la vez. La bd proporciona esta protección bloqueando (locking) el registro. De esta forma la bd introduce una nueva idea en el comportamiento de bloqueado: el registro.

La bd puede ser configurada para actuar como un administrador de documentos. Típicamente, las bds introducen otra opción: una cola. Si se trata de cambiar un registro que alguien más está modificando, la bd hace esperar en línea.

La belleza de una bd es que debido a que se ejecuta en un servidor compartido, la bd nunca está ocupada, nunca ausente en vacaciones, y puede ser accedida desde cualquier computadora conectada a ella a través de la red o de líneas telefónicas.

La idea importante de una bd gira alrededor de dos suposiciones: – Toda la información está disponible todo el tiempo. – Cualquier pieza de información es controlada por la bd de manera que sólo una persona a la vez pueda efectivamente modificarla.

2.1 Base de Datos: El Concepto.

Las bases de datos están basadas casi completamente en software. Las bases de datos son una división entre la tecnología centrada en el hardware y un mundo de mecanismos de software. Las bases de datos son entre tanto una clase de aplicación. La base de datos no es un concepto sencillo sino una familia de conceptos relacionados.

Una base de datos praporciona un mecanismo para interpretar o entender el significado de los datos. Esto es, la base de datos es más que un contendedor de datos, proporciana un contexto para interpretar las datas.

Una base de datos no necesita contener datos para ser una base de datos. Una base de datos esta definida por la organización de sus datos (estructura y esquema) y la imposición de un orden consistente (regularidad).

Una base de datos es una colección de tablas o archivos. Una base de datos es también auto-descriptible. Esta ayuda a los usuarios a encontrar registros particulares, a ordenarlos, a producir resúmenes estadísticos, etc.

Una base de datos es tanto la herramienta como la información con la que la herramienta trabaja. Una vez que se han creado las tablas de la base de datos, la base de datos proporciona poderosos comandos para cambiar y mejorar las tablas en el tiempo.

Una base de datos es una colección de tablas, cada una organizada como un conjunto de registras, juntos por un conjunto definido de relaciones entre las tablas. Las bases de datos san poderosas porque pueden crear conjuntos complejos de relaciones entre tablas. Diseñadas propiamente, una base de datos es el modelo de la información de una organización. Debido a que la base de datos es tan buen modelo de negocio, el diseño de la base de datos fue tan importante para tanto la revolución *mainframe* como para la Cliente/Servidor. El cancepto de la base de datos se canvierte así en una herramienta muy poderosa para entender un aspecto importante de cómo las organizaciones operan. Ahora con casi 20 años de edad, este concepto es uno de los primeros bloques fundamentales de la revolución cliente/servidar.

El hecho de que los discos magnéticas permitieran acceso aleatorio llevó a revolucionarios métodos nuevos de almacenamiento de datos en las computadoras. Así mismo el almacenamienta en-línea provocó un cambio importante en el diseño de saftware, entendiendo en-línea como "disponible en la computadora en todo momento".

En los 60s el munda computacional giró alrededor de las aplicaciones. Los programas eran primero, y todo lo demás seguía sus necesidades. El problema con este método es que alenta la creación de muchos archivos diferentes que contienen variaciones de la misma información.

En los últimos años de los 60s los programadores empezaron a preguntarse por qué estaban escribiendo el mismo software para manipular datos una y otra vez. Junto con este pensamiento, los programadores se dieron cuenta de que a lo mejor las datos eran más importantes que los pragramas con los que trabajaban. De hecho, una de las fuerzas que van atrás de la revolución cliente/servidor es la necesidad de acamadar cambios en reglas del negocio no sólo a nivel compañía, sino también con una base local. Los

programas cambian, los datos no.

Una base de datos puede servir como un modelo poderoso para toda la compañía. Como resultado, para diseñar mejores bases de datos, los profesionales del software crearon el modelo de datos, una herramienta poderosa en todo sentido.

El tener información exacta y actualizada es literalmente lo mismo que dinero. Es conveniente tener esa información así por la ventaja competitiva y ganancia que genera. Una base de datos se refiere al modelo conceptual de cómo una compañía trabaja.

La primera meta idealizada del diseño de una base de datos es la eliminación de datos redundantes. El beneficio más importante es la eliminación de inconsistencias.

Por otro lado, un concepto importante en las bases de datos, es el de relación. Una relación creo una liga entre dos registros para que, para los propósitos de una aplicación en particular, los dos registros funcionen como un gran registro. Esta idea de relacionar tablas entre sí es increiblemente poderosa. De esta forma las bases de datos le permiten a las compañías realizar preguntas útiles por adelantado para resolver problemas que aparezcan.

La información es un recurso corporativo. Todos los registros son de la propiedad de la compañía, de ningún departamento en particular. Los registros de la compañía son mejor almacenados en un solo lugar, donde los cambios pueden ser instantáneamente reflejados entre todas las aplicaciones. Los registros también pueden ser compartidos por todas las aplicaciones. Adicionalmente cualquier información es universalmente accesible a través de la red. Por lo tanto, la base de datos puede ser considerada como una base de datos universal, haciendo así toda la información disponible en todas partes, siempre.

Los principios esenciales de la revolución de la base de datos son los siguientes: 1. Las aplicacianes deben ser diseñadas alrededor de la base de datos, no de otra forma. 2. La información debe ser siempre almacenada en solo un lugar, en la base de datos. De esta forma, los cambios sólo pueden ser hechas uno a la vez, copias conflictivas de la misma información son imposibles, y el espacio es usado en la manera más eficiente posible. 3. La información debe ser introducida a la base de datas ton pronto como es generada.

La computadora es el cerebro, la red es el sistema nervioso, y la base de datos es la memoria de la organización.

Las bases de datos son uno clave para transformar las PCs y las redes de aplicaciones personales en máquinas con poder para equipos y organizaciones.

Eventualmente todas las decisiones hechas por el jefe ejecutivo deben de estar relacionadas a la versión más actualizada posible de cada pieza de datos de la compañía, siendo la única forma de hacer esto con una sola base de datos consistente. Una base de datos transaccional por definición tiene que ver con datos actualizados al minuto y aún al segundo. Los datos transaccionales están constantemente combiando.

Los datos analíticos nunca están actualizados. Los datos analíticos son usados para analizar patrones. Los patrones virtualmente siempre invalucran comparaciones, y esas comparaciones siempre están basadas en períodos de tiempo. Actualizado para una base de datos analítica significa los datos del último período. Las bases de datos analíticas capturan secciones congeladas de los datos de la organización, los que permanecen estáticos una vez capturados.

En una base de datos transaccional, los resultados múltiples deben de ser diferentes; en una base de datos analítica , los resultados múltiples deben de ser los mismos. Los datos operacionales son detallados, específicos, locales, y recientes. En contraste, los datos analíticos están basados en agregados. El punto principal de una base de datos analítica es facilitar las comparaciones y esta a su vez implica mantener información de la mayoría de las partes de la compañía toda en una misma parte. Todo ésto da la posibilidad que las bases de datos analíticas y transaccionales puedan ser entidades separadas. Sin embargo, examinando más profundamente se observa que deben de estar separadas. Reconociendo la diferencia esencial entre bases de datos transaccionales y operacionales encontramos una manera de sintetizar dos objetivos y necesidades aparentemente contradictorios.

Históricamente, las bases de datos centrales han sido necesarias para servir las necesidades de información de toda la organización. Los usuarios con requerimientos analíticos han resentido esas bases de datos centrales porque son lentas, difíciles de usar e inflexibles. La introducción de PCs y de sus modelos de bases de datos personales trajeron con ello libertad y anarquía. Mientras tanto, compañías de todo el mundo están enfrentándose al presente dividiendo sus bases de datos de cualquier forma. En el frente operacional, existen algunas ventajas obligatorias para el procesamiento distribuido; el punto ahora es cómo construir esos sistemas. Casi desde el principio, la mayoría de las organizaciones han encontrado que el separar las bases de datos analíticas y operacionales era la mejor parte del valor. Así que la respuesta era simple: parte los datos en dos primero, después tres capas. Algún día cuando ya sea que las Bases de Datos Relacionales (RDBs) sean suficientemente rápidas, o las bases de datos transaccionales sean suficientemente flexibles, o algo nuevo llegue a escena, se habrá llegado a la tierra prometida; todos los datos estarán en una sola base de datos y todas las decisiones estarán basadas totalmente en datos actualizados al minuto.

Las bases de datos son mucho de lo que son las computadoras. Una base de datos bien diseñada es un modelo de la organización. La información contenida en ella le permite

a un gerente estimar la salud de la compañía, predecir su futuro, y resolver problemas antes de que ocurran. Si la red es el sistema nervioso de una organización, la base de datos es su memoria. Las bases de datos son la clave para tanto salud como confianza. El significado real del término base de datos implica algo que construimos, no compramos. La base de datos es una de las tres componentes centrales que diseñamos para tener un sistema cliente/servidor completo.

La visión de la base de datos de los 90s es entonces la visión de bases de datos federadas. Consolidación, consistencia, coordinación, control, libertad personal y personalización local. Después de 20 años, afortunadamente tenemos un plan para finalmente construir una base de datos que trabaje. Y la razón por la que podemos esperar que trabaje es que no tenemos una idea más grande de una base de datos, sino por primera vez, una idea de cómo construir esa base de datos más grande a base de ideas más pequeñas, cada una de las cuales podemos construir.

2.2 Base de Datos: El Paisaje Técnico.

La base de datos ha sido una de las fuerzas filosóficas centrales que conducen el diseño de grandes (y pequeños) sistemas de información en los últimos 20 años. El concepto de base de datos es realmente tres cosas:

- Una máquina compartida que proporciona un almacen compartido de datos coordinados y controlados.
- Una herramienta para extraer, analizar, y desplegar esos datos.
- Un modelo más amplio para representar el estado de una organización tanto en el término corto como en el largo.

Las computadoras pronto cambiarán la sociedad más de lo que lo han hecho en cualquier otro tiempo, y la base de datos va a jugar un papel importante en ese revolucionario cambio.

Una base de datos se vuelve realmente interesante y poderosa en el momento en que empieza a tratar con muchos archivos relacionados a la vez.

Las primeras bases de datos fueron hechas para soportar las necesidades de grandes aplicaciones de procesamiento de transacciones corriendo en *mainframes*. Estas primeras bases de datos fueron construidas para representar conjuntos de registros jerárquicos. Las bases de datos jerárquicas son muy simples y por lo tanto fáciles de entender. La belleza de una base de datos jerárquica en una computadora es que se puede trabajar con cientos de niveles si se

necesita. En una base de datos de este tipo cada registro tiene cuando menos un padre y puede tener muchos hijos.

El modelado de datos es el proceso de usar modelos, típicamente en la forma de diagramas, para desarrollar y refinar el diseño de una base de datos real. Las entidades son la unidad fundamental de datos que cualquiera familiarizado con una organización reconocería cuando piense sobre la manera en que el negocio trabaja. Las relaciones son las conexiones entre las partes de una base de datos. Las entidades son los datos reales, clasificados por tipo, y las relaciones muestran cómo estos tipos de datos se relacionan entre sí. Cuando se describe una base de datos de esta forma, se habla de un modelo entidad relación de esa base de datos.

Una base de datos de red permite que los registros estén en múltiples folders todos al mismo tiempo. Esta permite colocar los datos en cualquier número de vistas jerárquicas, todas al mismo tiempo. Sin embargo, las bases de datos de red son definitivamente más complejas que las bases de datos jerárquicas. Una base de datos de red esencialmente permite que un registro esté relacionado a un número ilimitado de otros registros. Son llamadas bases de datos de red precisamente porque la forma de las relaciones se ve como una red de líneas. Todos los tipos de consultas pueden ser contestadas con una base de datos de red porque se pueden representar directamente todos los tipos de relaciones inherentes en las datos de la organización

El proceso de cambiorse hacia atrás y adelante entre múltiples archivos para encontrar información que llevará o contestar una pregunta se llama navegación. La navegación puede ser activo o pasiva. La navegación pasiva ocurre cuando un programador realiza todas las decisiones de navegación por avanzado y después construye la ruta de navegación a través de la base de datos en la aplicación. Los usuarios navegan activamente a través de una base de datos cuando están explorando los datos para buscar patrones o para encontrar pequeñas cantidades de información.

En medio del desarrollo de bases de datos de los 70s, Edgar Codd, un investigador de IBM, empezó a trabajar en un nuevo modelo de base de datos llamado el modelo relacional. En el mundo relacional, las archivos son tablas, los registros son renglones, y los campos son columnas. Los usuarios están por mucho más cómodos trabajando con simples colecciones de datos que con complejos conjuntos de archivos y relaciones. Al trabajar con una Base de Datos Relacional (RDB) un programador o usuario puede rápida y fácilmente definir una nueva relación entre cualquiera das tablas. Codd quería que las RDBs le hicieran fácil a los usuarios encontrar datos en ellas.

Desde el inicio cada RDB se volvió completa con un lenguaje de consultas. Ese lenguaje SQL (Structured Query Language) se convirtió en una parte intrínseca de la base de datos. Una base de datos es simplemente una máquina para manejar datos compartidos, y un lenguaje de consultas (SQL) es una herramienta específica para especificar conjuntos de registros a ser extraídos de una base de datos. Idealmente, cada base de datos puede ser accedida a través de

muchas herramientas de búsqueda, y una herramienta de consultas puede trabajar con muchas bases de datos. SQL fue por primera vez desarrollado en un tiempo cuando los buenos lenguajes de programación simplemente no existían. SQL rápidamente se convirtió en un estándar. SQL hace fácil especificar relacianes par adelantado "en el vuelo" entre archivos o tablas. La habilidad de SQL para realizar esta tarea esta basada en la unión (join). Estas operaciones de alto nivel como las uniones son instrucciones que le permiten a una persona decirle a una computadora hacer un montón de trabajo en una petición. Una unión es una operación que combina dos tablas no relacionadas previamente en una sola tabla más grande. Las operaciones de alto nivel le proporcionan al usuario un vocabulario para trabajar con grandes cantidades de información sin pragramación. Por otra parte SQL tiene un objetivo más limitado: la misión de SQL está limitada a expresar consultas. Una búsqueda es una especificación que le permite a una base de datos extraer un conjunto específico de registros. SQL es un lenguaje poderoso. Una amplia variedad de búsquedas sofisticadas pueden ser expresadas efectivamente en un lenguaje parecida al inglés. Sin SQL na habria farma fácil para tadas las diferentes herramientas con las que interactúa directamente el usuario de hablar a las muchas máquinas de bases de datos. Debido a que SQL se ha convertido en el estándar, los usuarios pueden tener por seguro que cuando escojan una herramienta de base de datos personal entonces podrán usar esa herramienta para extraer información de tadas las diferentes máquinas de bases de datos en las que haya datos almacenados. SQL parece ser más útil como lenguaje para que programas hablen a otros programas.

El modelo de red está construido alrededor de la idea de expresar la estructura de los datos de la organización directamente en el diseño de la base de datos por sí misma.

Las RDBs están basadas en lo suposición de que estas relaciones no deben de estar dentro de la base de datos, sino la base de datos y su lenguaje de consultas deben de proporcionar poderosas herramientos que permitan especificar y ejecutar relaciones en la marcha. El modelo relacional pide que lo estructura de tanta la base de datos así como de los registros reflejen lo idea de no construir relaciones dentro de la base de datos misma. La razón para ésto es simple: flexibilidad. Una RDB depende de tres principias: normalización, llaves foráneos, y uniones. Normalización es la técnica de base de datos para eliminar estructuras de registros complejas, y las llaves foráneas es la técnica correspondiente para tratar relaciones entre archivos. Estas dos técnicas se ocupan de deshacerse de todas las relaciones expresadas directamenmte. Las unianes proporcionon un mecanismo para construir relaciones al momento. Porte de la normolización involucra dividir datos y ponerlos en un lugar común. La normalización es el proceso de tomar todos los datos repetitivos que aparecen en más de un lugar y separarlos en archivos diferentes. El efecto neto de la normalización es eliminar todos los registros complejos de uno base de datos. Una RDB construida alrededor de una estructura tabular simple no sólo refuerza sino requiere que esas datos se normalicen. Proporcionando un solo lugar para encontrar tados los registros de un tipo particular, una base de datas normalizada elimina virtualmente la redundancia mientras que permite infinidad de flexibles construcciones de relaciones.

El encontrar registros de una manera rápida basándose en el valor de uno o dos campos "llave" es una operación común y crítica en cada base de datos. Por esta razón, facilidades especiales están construidas para hacer estas recuperaciones esencialmente instantáneas. Una base de datos típicamente construye y mantiene un índice en los campos que van a ser usados para la recuperación de información. Los índices realizan la difícil tarea de encontrar información casi trivial, siendo éstos así fundamentales para las bases de datos. Un índice proparciona una "llave" a los datos que de otra manera estarían encerrados en la base de datos. Una "llave foranea" es una columna agregada a una tabla que permite que una relación se establezca con registras en otra tabla. Los valores en una columna de "llave foranea" están asociados con un índice construido en otra tabla. En una base de datos, cada registro tiene o debe tener cuando menos una "llave" única asociada con él, un valor llave que permita a las usuarios referirse a ese y sólo a ese registro.

Una unión combina dos o más tablas en una más grande basada en la correspondencia de valores en columnas comunes. Una característica de las uniones es que según se recombinan datos normalizados previamente en los que las repeticiones son comunes, naturalmente se ven repeticiones otra vez. Sin embargo, las uniones son operaciones temporales.

En el camino relacional a las bases de datos existen varias reglas:

– Normaliza todos los diseños de bases de datos para que la redundancia y las estructuras de registros complejos sean eliminadas.

– Usa siempre llaves primarias y foraneas explícitas(no "invisibles") y externamente entendibles para hacer así a los registros más entendibles.

– Usa las uniones como el mecanismo para establecer y usar las relaciones.

La belleza del camino relacional descansa en su flexibilidad y falta de redundancia. Las RDBs son supuestamente poderosas porque están respaldadas por todo un modelo matemático. Sin embargo las bases de datos que están realmente basados en matemática son interesantes sólo en teoría, pero no en la realidad. El corazón del problema con las RDBs es que son demasiado simples para muchas aplicaciones del mundo real. Este tipo de bases de datos realmente manejan relaciones matemáticas a un cierto grado, pero esas relaciones están basadas en tablas sencillas y algunas veces conjuntos de tablas, algo que no es suficiente para construir bases de datos reales. Las bases de datos normalizadas son por mucho más complejas de construir y entender que las no normalizadas. Las características precisas que la normolización elimina se convierten en las piedras angulares de entendimiento. Las bases de datos normalizadas son siempre más complejas de lo que eran antes de la normalización. Los registros y archivos individuales, una vez normalizados, se convierten algunas veces en muchas tablas sencillas. Lo que es una sola lectura en un sistema no normalizado se convierte en 35–40 lecturas en la base de datos normalizada. Las bases de datos normalizadas invariablemente tienen que convertir un número determinado de archivos en 3–10 veces más de tablas sencillas.

Las tablas son más simples, pero todo el conjunto no lo es, y cuando todo está normalizado, las operaciones de rutina –reportes, consultas, actualizaciones transaccionales– toman hasta 3–4 veces más de operaciones a disco y 3–4 veces más en tiempo de computadora para procesar. Para aplicaciones de tamaño pequeño y mediano, el tomar 3–4 veces más en estas operaciones es aceptable, pero para aplicaciones grandes no lo es.

Las tablas son una manera sencilla de representar datos: un programador puede explicar una tabla a un usuario en algunas minutos. Para aplicaciones sencillas que involucran sólo algunas tablas (menos de 20), una RDB le permite a las aplicaciones, consultas, y reportes ser construidas rápida y fácilmente.

Las RDBs se volvieron populares en el mismo tiempo en que los minicomputadoras y los sistemas departamentales estaban en su inicio. Desafortunadamente aquí el representar grandes Bases de Datos en una forma relacional se convierte en un problema tanto para humanos como para computadoras. La gente no puede trabajar con 20 o 30 uniones de tablas, y el costo de construir esas grandes uniones continua siendo un factor de importancia aún cuando las computadoras son más rápidas cada día. En esto el tener un catálogo de vistas desnormalizadas es un procedimiento estándar en cualquier ambiente de soporte de decisiones grande.

Después consecuentemente por las limitantes de los modelos anteriores han surgido las Bases de Datos Orientadas a Objetos (OODB). Las OODBs en muchas formas son sólo Bases de Datos grandes. En estilo las OODBs son más modernas que las Bases de Datos de Red, pero en esencia representan un regreso al mundo de red del pasado: las Bases de Datos tratan acerca de relaciones entre colecciones de registros, y estas relaciones deben de ser expresadas directamente en la Base de Datos.

Las Bases de Datos de Red, Jerárquicas, Relacionales y de Objetos compiten entre sí en dominios de aplicaciones algo diferentes entre sí. Las Bases de Datos de Red y Jerárquicas todavía contienen la vasta mayoría de los datos de producción del mundo. Las RDBs son ampliamente usadas para aplicaciones departamentales. Un número en aumento de organizaciones están experimentando con la construcción de sistemas cliente/servidor distribuidos con RDBs corriendo en cada servidor. Las OODBs ganan rápidamente una fuerte presencia soportando una nueva clase de Base de Datos.

El modelo relacional no está muerto, ni está de alguna forma mal. A través del tiempo, los diseñadores de Bases de Datos han aprendido a cómo hacerlo mejor. La normalización de las Bases de Datos es una técnica extremadamente valiosa para el diseño de Bases de Datos siempre y cuando no sea aplicada religiosamente.

Mientras hay un modelo matemático asociado con las RDBs, ese modelo describe una forma de Base de Datos que nadie ha construido alguna vez intencionalmente. Las RDBs

efectivamente reemplazaron las Bases de Datos de Red en popularidad. Las OODBs quiza reemplacen a las RDBs, de la misma forma en que las RDBs reemplazaran a sus predecesaras.

SQL ha emergido camo un estándar muy importante tanto para programadores coma para herramientas de construcción de Interfases Gráficas de Usuario de Bases de Datos. La situación actual sugiere que las sistemas relacianales posiblemente no desaparezcan.

Las Bases de Datos de Red que sopartan estructuras de registros complejos regresarán. Sin embargo quizá regresen como Bases de Datas Postrelacionales. Una Base de datos Postrelacianal es la que cambina las mejores características de las RDBs con las nuevas características encontradas en las OODBs.

Finalmente, hay un giro que garantiza que el futuro será interesante y diferente: el modelo cliente/servidar. La necesidad de saportar la aperación de verdaderas Bases de Datas Distribuidas es un factor que cambiará tado el panarama de las Bases de Datos de una farma todavía no canocida. En el próximo siglo, las Bases de Datos Relacionales basadas en alguna forma de modelo Postrelacional y amarradas a Interfases Gráficas de Usuario serán la norma.

3. Las Redes en el entorno del modelo Cliente/Servidor.

3.1 La Red de Area Local: Un Nuevo Tipo de Computadora.

Una Red de Area Local (LAN) es una red de alta velocidad que interconecta terminales y computadoras en muy pequeñas distancias; el equipo usualmente está contenido en un solo edificio. Debido a la velocidad de la LAN, no se considera realmente como una red.

Una red es un mecanismo para conectar dispositivos computacionales juntos. Adentro de cada computadora hay una red. Cada computadora consiste de una colección de componentes computacionales: discos, impresoras, memoria, la unidad de cómputo, el display de estado, etc. Una red de alta velocidad especializada conecta todos estos componentes. Algunas veces esta red es llamada plano posterior, otras canal, y algunas otras veces bus. En muchas formas, la velocidad de un PC está definida por la velocidad a la que la información puede fluir a través de las ranuras opcionales en donde se conectan la mayoría de los componentes internos de la misma.

El propósito de una red es conectar múltiples computadoras y terminales juntas, generalmente sobre grandes distancias. El propósito de la estructura de bus adentro de una computadora es enlazar computadoras discretas para crear un sistema computacional funcional.

La red funciona como una extensión de la estructura de bus interna de la PC. Los usuarios pueden acceder los componentes ligados a la LAN tan rápido o más rápido de la que pueden acceder los componentes ligados a la estructura de bus que está adentro de la PC misma. En un sentido, la LAN es una extensión de la PC.

Los *mainframes* y las minicomputadoras son muy caras. En general las computadoras tienen una característica no usual: es muy difícil decir con precisión por cuánto tiempo serán útiles. La utilidad de una computadora está basada en tres factores: - cuánto tiempo toma el que se deteriore. - La velocidad a la cual se vuelve obsoleta. - cuándo las usuarios acaban con su capacidad.

Pero la combinación del caro *mainframe* y un huésped (host) de minicomputadoras más pequeñas alrededor del mismo ofrece un conjunto de pasos más atractivo en el futuro de muchas corporaciones.

Por otra parte los servidores basados en PCs son muy baratos. El agregar servidores uno a la vez según la demanda crece lleva a una curva de costo que es prácticamente plana. La arquitectura basada en una LAN ofrece el potencial de poder computacional casi sin límite, teniendo solamente suficientes PCs y servidores.

En un principio las LANs fueran vistas como un complemento de las *mainframes* y de las minicomputadoras del tiempo. En las últimos años de los 70s, Datapaint introdujo la Computadora de Recursos Atadas (Attached Resource Computer) construida alrededor de la ARCNET, una tecnología propietaria de LAN. Los sistemas ARCNET fueron construidos alrededor de dos tipos de computadoras, las cuales serían llamadas clientes y servidores hoy en día. Los sistemas ARCNET proporcionaron dos nuevas ideas: – Tanto las clientes como los servidores son relativamente baratos. – Debida a que ARCNET es tan rápida, el sistema completo se puede considerar como una gran computadora.

En un sistema Datapaint, la red local es tan rápida que no se le debe de considerar como una red. El sistema Datapaint permite que el edificio de oficinas completo en donde todos trabajan sea la caja que contiene el sistema computacional. De esta manera, la computadora y el edificio de oficinas están ahora entrelazados.

La belleza de ARCNET es que cada vez que un servidor o una estación de trabajo (workstation) es instalada, la misma computadora central es expandida. Con sistemas computacionales basados en terminales, el agregar una terminal suma carga de trabajo adicional a la computadora central sin agregar ningún tipo de capacidad computacional.

En el ambiente de LAN, no hay computadora central. En lugar de esto, existe un sistema computacional distribuido. Aquí los clientes son una parte integral del sistema computacional completo. En un sistema cliente/servidor real, las estaciones de trabajo son más que sólo terminales inteligentes. Este estilo de computación es a veces llamado procesamiento cooperativo. La red es una computadora. La computadora es la combinación de todos los servidores, todas las estaciones de trabajo y todos los cables de alta velocidad que los conectan.

El primer impacto conceptual de la revolución cliente/servidor basada en una LAN fue la introducción de un sistema computacional con poder de adaptación y que podía crecer suavemente. Empezando con una estación de trabajo, una organización podía hacer crecer un sistema computacional integrado, agregando servidores y estaciones de trabajo a cualquier escala que tuviera sentido. Así la configuración completa se hace más grande y más rápida.

Aunque ARCNET por sí misma no es muy usada hoy en día, demostró no solamente que la red podría ser la computadora, sino también que el edificio podría ser la caja. En el ambiente LAN, la caja más grande es el edificio completo. El segundo impacto de importancia del sistema ARC fue el hacer a la gente entender que las computadoras de negocios, como las PCs, podían responder a las necesidades sencillas de usuarios. Pero al desarrollo de la visión de Datapoint le faltaron algunos elementos que son requeridos para permitir una verdadera revolución en la manera en que las organizaciones se desenvuelven. No obstante que los recursos computacionales de un sistema ARC estaban distribuidos alrededor de edificios completos, el estilo de las aplicaciones que se construían sobre esos

recursos, era prácticamente idéntico al estilo de aplicaciones de *mainframe*. En teoría, las estaciones de trabajo eron capaces de cambiar la monera en que la gente interoctuaba con las computadoras, pero en la práctica, todavía se veían como terminales.

El viaje hacia el cambio de la sociedad, o cuando menos de las organizaciones, comienza mostrando que sucede cuando la PC finalmente aprende cómo cohabitar con el *mainframe*.

Los mundos virtuales de todos tipos son construidos sobre enormes cantidades de datos. Una de las toreas clave de una PC es dar visuolizaciones de esos mundos virtuales. El punto principal de la GUI gira alrededor de la creación de algún tipo de mundo virtual. El poder desplegar mejores imágenes implica mejores mundos virtuales, lo cual significa mejores aplicaciones.

Con el advenimiento de los CD-ROMs, los cuales pueden olmacenar y transportar grandes cantidades de información en muy pequeños y baratos discos, un simulador de vuelo podría tener escenario para docenas de aeropuertos y ciudades todo en un solo CD.

El Cliente/Servidor combino los mundos virtuales hechos posibles por las PCs con la información real y en tiempo real contenida en las computadoras de la organización. La necesidad de trobajar con información actualizada en tiempo real implica que los usuarios de la computadora deben tener acceso directo a los datos en la computadora central de la organización. Uno PC trae control directo sin-preguntas sobre la información de una oplicación personal. Uno PC es un sirviente inconsable que hace lo que se le diga. Aquí cabe recordar que los individuos no controlan al *mainframe* – el *mainframe* los controla a éllos.

Los mundos virtuales sólo son posibles cuando la información que conduce a la aplicación está en la misma computadora que la aplicación.

Millones de gente tienen PCs que son usadas para hablarle a las aplicaciones basadas en *mainframe* – y minicomputadora. La mayoría del tiempo, la PC es una terminal mientras habla a la computadora más grande. El hacer las aplicaciones de *mainframe* fáciles de usar requiere más que una simple transformación de las pantallas que el usuario ve. El reemplazar a la terminal con una PC permite usar una clase de herramientas llamada raspador de pantalla (screen scraper) la cual atrapa la forma que es mandada a ser desplegada desde el *mainframe* y en lugar de desplegarla la viste para que se vea más presentable. Sin embargo esto no convierte al código residente en el *mainframe* en una aplicación gráfica. No obstante hay una manera de tener tanto al mundo gráfico como al que comparte la información trabajando conjuntamente. Un nuevo tipo de computadora realiza esta tarea, una computadora que combina las características de la computadoras personales y de las *mainframes*: la LAN.

La LAN se debe de ver como un nuevo tipo de computadora, no como una red. Ahora la computadora tiene tanto elementos dedicados personalmente como compartidos organizacionalmente. Los datos necesitan estar en la computadora, y la LAN es la computadora. Una parte de la LAN – la PC – está dedicada a uno; otra parte está dedicada al compartir. A través de la LAN son uno gran computadora – un sistema integrado sencillo. El resultado combina las virtudes de las computación personal, las gráficas y los mundos virtuales, con la computación compartida, los bases de datos y el control coordinado. Esto origina mundos virtualmente compartidos. Si la PC es el escritorio electrónico, la LAN es la oficina electrónica.

Los PCs juegan un importante papel, proporcionando la máquina para generar los mundos gráficos. Los servidores son las máquinas que comparten y coordinan el acceso a una variedad de recursos compartidos que van desde impresoras hasta bases de datos. La LAN liga las PCs y los servidores, formando así un nuevo tipo de sistema computacional en el cual los datos personales y los datos compartidos son los mismos. La belleza de la LAN es que facilita este mundo compartido, dejándole a cada usuario el poder y la autonomía ofrecida por la PC.

3.2 Redes de Area Amplia: Conectando el Mundo.

Las LANs no se les considera redes. El problema con éstas es que simplemente no hacen los casos que una red debería hacer. Cuando se requiere acceder una computadora de un lugar diferente que el mismo edificio donde se encuentra, se necesita otra solución diferente a los LANs. Esta solución es la Red de Area Amplia (WAN).

El concepto central detrás de una red computacional es la conectividad. El realizar una conexión libre de preocupaciones, libre de pensamiento, de cualquier computadora a otra, es crítico para el eventual éxito a largo plazo de los sistemas cliente/servidor.

Las primeras terminales, ya sean locales o remotas, estaban conectadas a sus computadoras a través de una conexión dedicada. Con la invención del modem, las computadoras y las terminales pudieron hablar entre ellas sobre líneas de teléfono normales, y el sistema telefónico no pudo detectar que eran datos los que se movían de un lado a otro, y no voces. Muchos de las redes más simples de hoy en día están todavía basadas en esta simple tecnología.

Según las terminales se volvían realmente populares, las fuerzas armadas se dieron cuenta de las grandes ventajas de usar terminales pegadas a computadoras para coordinar y controlar unidades militares extendidas alrededor del mundo. Desafortunadamente, cuando

se introduce un sistema tal, la organización rápido alcanza el punto de dependencia total. Por esta razón, en 1975, la Agencia de Proyectos de Investigación Avanzados de Defensa (DARPA) fundó la investigación en la construcción de redes altamente confiables.

La red que se ha descrito hasta el momento consiste de tres componentes: terminales, computadoras, y nodos de red. Los nodos son servidores especializados que convierten mensajes a paquetes, manipulan la detección de errores y la retransmisión, y otras cosas. Con la llegada de la conmutación de paquetes, las conexianes directas regresaron a la jugada. Pero ahora, se habla acerca de conexiones directas entre nodos de red, en lugar de entre terminales y computadoras o entre computadoras y computadoras. La capacidad de intercambio es lo que hace a la red telefónica tan flexible. Las redes de computadoras realizan también esto en el intercambio de paquetes.

Las esquemas de direccianamiento no son ideas nuevas; éstas resuelven un gran problema en las comunicaciones. Si nuevas tipos de redes van a existir, nuevos tipos de direccionamiento tendrán que existir junto con ellas. Así que una parte importante de una red intercambiadora de paquetes es el esquema de direccionamiento. Las direcciones pueden ser nombres, números, o cualquier otra construcción que pueda ser programada en una computadora; el nodo computacional, después de todo, es sólo una computadora. Los nodos de red introducen una nueva posibilidad: el almacenar y el reenviar.

Lo mejor acerca del intercambio de paquetes es que es adaptivo y dinámico. Esto significa que la red mantiene la mejor ruta para información cada vez que ésta tiene que ser mandada. Una red de intercambio de paquetes define prácticamente disponibilidad y silencio. Las rutas de información son cambiadas en el momento, aún a la mitad de los mensajes, adaptándose no sólo a errores, sino aún a tráfico pesado y ruido en la línea.

Las redes de conmutación (intercambio) de paquetes son tan importantes estratégicamente en el mundo cliente/servidor debido al grado de infraestructura compartida que facilitan. Debido a que mucha gente puede comportir los nodos en una red de intercambio, es económico interconectar estos nodos con ligas de muy alta velocidad. El comportamiento de intercambio de las redes de paquetes le permite a un pequeño número de líneas de transporte largas de super-alta-velocidad dar servicio a muchos más nodos de red esparcidos a través del país. Usando el intercambio en un ambiente de almacenar y remandar, las usuarios en los nodos en aún lugares muy pequeñas pueden todavía acceder ligas de muy alta velocidad alrededar del mundo. Estas redes publicas tienen un impacto cultural fundamental así como técnico basado en dos aspectos: precio independiente de la distancia y el desarrollo de correo electrónico.

El correo electrónico (E-mail) es instantánea porque una vez que una pieza de correo es mandada, ésta está inmediatamente disponible en todo el mundo. No obstante que el E-mail también se "encola", una vez mandado, el E-mail espera pacientemente por siempre con ninguna acción extra requerida. El E-mail da rápido acceso a las

comunicaciones, mientras que también proporciona control sobre las interrupciones. La medida que más nos dice acerca de la importancia del correo electrónico es su popularidad. Para muchos usuarios el E-mail es la aplicación que más corren cada día. El correo electrónico es la aplicación que virtualmente define de lo que se tratan las redes. El E-mail es la aplicación que más ha propagado el crecimiento de grandes redes.

Según las LANs crecieron tanto en tamaño como en popularidad, una nueva clase de sistemas E-mail apareció en estas redes locales. Ahora los grupos de trabajo dependen cada día en dos tipos de servidores para coordinación: correo electrónico y bases de datos.

La tendencia de nominar al *mainframe* como el servidor de elección está muy vigente cuando se habla de las aplicaciones cliente/servidor más sofisticadas. Sin embargo, la mayoría de las organizaciones finalizaron con dos tipos de servidores: grandes y pequeños. Los computadoras corporativas de gran-escala se convirtieron en servidores centrales salvaguardando los datos de la organización más críticos (y valiosos) y los servidores basados en PC del grupo de trabajo, le dieron a las grupos locales los medios para compartir y coordinar el acceso a datos.

Para pequeños grupos de trabajo y pequeñas compañías, sin embargo, aún el correo electrónico basado en minicomputadoras era por mucho muy caro. El desarrollo de LANs y de servidores basados en PCs en la mitad de los 80s le dieron aún a la más pequeña compañía o grupo de trabajo los beneficios del E-mail.

La función central de todas las redes es el de dar acceso global a los datos. Una de las razones primarias de tener una computadora coordinando el acceso a datos y el compartir de datos es hacer a esos datos disponibles de cualquier parte. La función de la WAN es convertir a las bases de datos del mundo en una biblioteca global dándole a cada usuario de una computadora acceso a un amplio almacenamiento de todos tipos de datos a todos horas. El impacto de la red es no convertir a las computadoras en una biblioteca global, sino convertir a la red misma en una aldea global.

DARPA fundó la investigación original que resultó en redes de intercambio de paquetes, y consecuentemente en las WANs de hoy. Según el intercambio de paquetes se volvió disponible en los laboratorios de investigación, DARPA estuvo de acuerdo en fundar la instalación de una red de intercambio de paquetes de nivel nacional, llamada ARPAnet, para probar la tecnología en desarrollo y dar una base para desarrollo posterior.

Casi al mismo tiempo que ARPAnet apareció, los sistemas E-mail ganaban popularidad en muchos de los mismos lugares de investigación que fueron seleccionados para estar en la red. El E-mail empezó a hacer colaboraciones en el continente de una forma tan conveniente como las colaboraciones en el edificio.

En los últimos años de los 70s cada país grande tenía su propia red de intercambio de paquetes pública. Sin embargo, tan pronto como un usuario en una red quería comunicarse con otro usuario de una red diferente, todo el sistema se iba para abajo. A principias de los 80s una organización llamada CCITT (Consultative Committee for International Telephony and Telegraphy) vino al rescate, facilitando el desarrollo de protocolos comunes. Estos protocolos llamados estándares, hacen del mundo un lugar conectado. De esta forma la CCITT desarrolló un protocolo para redes de intercambio de paquetes llamado X.25, el cual permitió a todas estas redes una interconexión transparente. El estándar X.25 describe los mecanismos con los que las computadoras le hablan a las redes, las terminales a las redes, y las redes entre si mismas.

A partir de 1982 la CCITT asignó un grupo de trabajo para desarrollar un estándar para interconectar sistemas de correo electrónico. El resultado, llamado X.400, prescribe mecanismos donde dos o más sistemas de e-mail pueden intercambiar correo, así como otros mecanismos para permitir que las PCs hablen a los proveedores de E-mail de una forma que tome ventaja de la inteligencia de esas PCs.

Los protocolos más importantes habían sido implementados de una forma tan amplia que ARPAnet estuvo lista para convertirse en la red Internet. La idea principal de la Internet representa un paso hacia adelante en lo que se refiere a las WANs. La Internet introdujo una nueva idea, la cual es el propósito primario de la WAN: no conectar terminales o computadoras, sino computadoras a computadoras. La WAN ahora sirve a otro propósito: mueve correo electrónico y otra formas de datos flexiblemente entre computadoras. Las computadoras interconectadas pueden ser también los "clusters" computacionales cliente/servidor creados por una LAN. Por esto, las WANs pueden conectar computadoras con redes locales de computadoras así como redes locales con otras redes locales. De esta forma una WAN es frecuentemente una red entre las redes, o una interred. La Internet es la abuela de redes grandes. Hoy en día, no obstante que las barreras culturales todavía existen el bloque principal es la interconexión E-mail. A este respecto, la CCITT creo el estándar X.500 para definir un estándar para directorios de servicio para intercambiar información así como para que otras computadoras pregunten acerca de esos directorios de servicio. Todo parece apuntar a que para el final de esta década, el E-mail se convertirá en un mecanismo estándar para la comunicación orgonizacional y personal en distancia y tiempo.

Juntas las PCs, los servidores, las LANs y las WANs hacen posible el construir sistemas distribuidos que realmente abastecen las necesidades de equipos automanejados.

4. Caso de Estudio.

4.1 Análisis de Requerimientos.

4.1.1 Introducción.

4.1.1.1 Propósito.

Se pretende realizar un proyecto tipo con el fin de examinar un producto de software con el cual se puedan desarrollar eficientemente nuevas aplicaciones críticas para la empresa Grupo ICA, así como substituir las ya existentes. El grupo de desarrollo del proyecto estará a cargo de gente externa e interna de Digital Equipment de México S.A. de C.V. del área de Integración de Sistemas (S.I.) así como de Productos de Software de Digital/Forté.

4.1.1.2 Organización del Cliente.

El cliente como ya se mencionó es el Grupo ICA, cuya primera empresa se fundó el 4 de julio de 1947. La primera meta de los fundadores de ICA consistió en ejercer a plenitud su capacidad profesional y técnica para ser competitivos y participar en las oportunidades de trabajo del país. El Grupo ICA tiene y ha tenido una presencia constante en todos los campos de la construcción al ritmo del crecimiento del país realizando las más variadas obras de infraestructura y equipamiento urbano. El prestigio del Grupo que perdura hasta hoy, es el de ser una de las empresas constructoras latinoamericanas de mayor capacidad realizadora.

El grupo ha participado en negocios inmobiliarios y ha emprendido actividades en las áreas de autopartes, electrónica, minería y petroquímica. Sin embargo paralelamente a estas actividades, ICA es y sigue siendo ante todo un grupo constructor ya que nació y creció en la construcción, la cual es su actividad primordial.

La organización del Grupo ICA es la siguiente:

- Presidencia.
- Comite Ejecutivo.- Vicepresidencia de Finanzas, Vicepresidencia de Administración y Dirección de Recursos Humanos y Servicios Corporativos.
- Unidades de Negocio.- Construcción Pesada, Construcción Urbana, ICA Fluor-Daniel, Operación Internacional, ICA Bienes de Capital, ICA Asociadas, ICA Concesionaria y Servicios Municipales, ICA Maquinaria e ICA Servicios.
- Empresas, Gerencias.
- Obras, Proyectos, Operación, Concesionarias, Taller, Almacen.

Durante el desarrollo de este proyecto se trabajará estrechamente con la Dirección de Telecomunicaciones e Información del Grupo ICA la cual es parte de la Dirección de Recursos Humanos y Servicios Corporativos.

4.1.1.3 Historia.

Para la Dirección de Recursos Humanos y Servicos Corporativos del Grupo ICA se realizó en GUPTA SQL Windows un sistema con el cual se lleva todo el directorio de la misma, el cual consiste de aproximadamente 5500 empleados técnicos y administrativos. Sin embargo, debido a las limitaciones particulares que tenía este sistema, GUPTA fue solamente un producto intermedio con el cual se solucionó momentaneamente la necesidad que se tenía. Nunca se perdió de vista el substituirlo por otro con mejores características.

Por otra parte en ICA Construcción Urbana también existe desde bastante tiempo atrás, una serie de programas desarrollados en Clipper los cuales sirven de apoyo para la declaración anual del Impuesto Sobre el Producto del Trabajo de los trabajadores de la empresa. Estos programas entre otros limitaciones tienen que el tipo de datos que éstos únicamente acceden es el formato .dbf.

4.1.1.4 Proceso de Análisis.

Para llevar a cabo esta etapa en el desarrollo del proyecto se efectuarán entrevistas con personas de la Dirección de Recursos Humanos y Servicios Corporativos, de ICA Construcción Urbana así como de la Dirección de Telecomunicaciones e Información del Grupo ICA.

4.1.2 Objetivos del Negocio.

4.1.2.1 Necesidades del Grupo ICA.

La Dirección de Recursos Humanos y Servicios Corporativos del Grupo ICA requiere de un sistema con el cual en especial altos ejecutivos del Grupo puedan obtener al momento información sobre el o los empleados que se necesite incluyendo inclusive fotos de los mismos. Para esto se requiere contar con una fuente de información confiable así como hacer uso de un ambiente distribuido para ello.

Por el lado de ICA Construcción Urbana y los programas desarrollados para el pago de impuestos, se ve claramente la necesidad de migrar de un producto como Clipper a otro

que permita sobre todo el manejo de información proveniente de distintas fuentes como archivos de tipo texto, Oracle y claro archivos tipo .dbf.

El manipular multimedia es así mismo una preocupación del Grupo ICA. El que ésta se pueda manejar desde cualquier aplicación es un aspecto vital en el desarrollo de las mismas.

De manera general se busca una alternativa mucho más robusta para el desarrollo definitivo de las necesidades en materia de producción de software de la empresa. Se debe efectuar aquí una agrupación del software a crear en un ambiente de desarrollo así como de ejecución comun, poderoso y confiable entre otras características.

4.1.2.2 Metas del Negocio y Medidas.

Entre las metas del negocio para este caso se encuentra el adoptar un software de desarrollo como un estándar para el desarrollo de todas las aplicaciones futuras de la empresa así como para la migración gradual de todas las ya existentes, sin que esto interfiera en obsoluto en el desarrollo normal de las mismas. Para cumplir con los intereses del Grupo, este software debe de ser escalable, poderoso y flexible.

El que estas metas se puedan alcanzar redituará en beneficios incalculables en el desarrollo de los procesos involucrados.

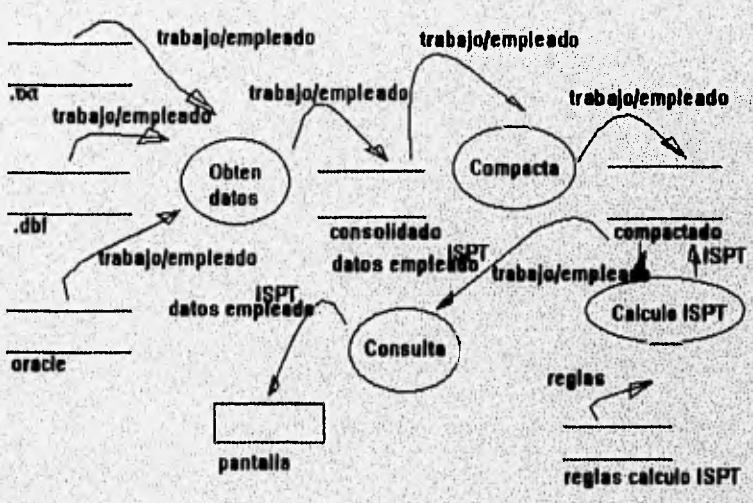
4.1.3 Procesos del Negocio y Flujos de Información.

Los flujos de información de los procesos en estudio son los siguientes:

Proceso de la Dirección de Recursos Humanos y Servicios Corporativos del Grupo ICA.

Obten datos empleado. Proceso que extrae de una tabla de Oracle toda la información de los empleados especificados para desplegarla después en pantalla.

Proceso de ICA Construcción Urbana.



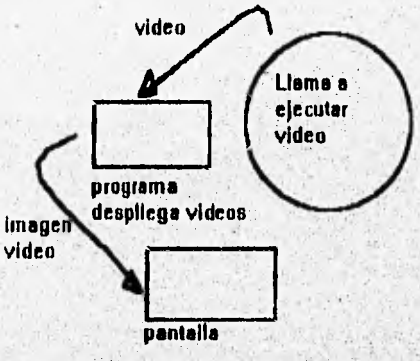
Obten datos. Proceso que lee información de trabajadores desde distintas fuentes (archivos tipo .txt, .dbf y tablas Oracle) y la reune en una tabla de Oracle.

Compacta. Proceso que elimina la redundancia de información de los distintos trabajadores dejando un solo registro por cada uno de ellos en otra tabla de Oracle.

Calculo ISPT. Proceso que efectua el cálculo del ISPT para los diversos trabajadores del Grupo ICA y que deja los resultados de dichos cálculos en la misma tabla de donde leyó la información.

Consulta. Proceso que arroja en forma de tabla a pantalla la información generada por el proceso Calculo ISPT

Proceso de manipulación de multimedia.



Llama a ejecutar video. Proceso que manda a ejecutar la aplicación que ejecuta el video especificado.

4.1.4 Requerimientos de la solución.

4.1.4.1 Requerimientos del negocio.

La empresa tiene que llevar a cabo un gran desempeño cooperando a la realización de este proyecto, de manera que se puedan cubrir de forma satisfactoria las necesidades planteadas. Es probable que para que se cumpla con ésto no se requiera que se cambie la forma de realizar ningun proceso del negocio. Sin embargo, en caso de que ésto fuera necesario se solicitaría sin lugar a dudas inmediatamente.

4.1.4.2 Requerimientos de la organización.

Se requiere que se proporcione el apoyo necesario y oportuno en lo referente a cualquier tipo de duda que se llegue a presentar relacionada con la forma de llevar los procesos del negocio actualmente. Debe de existir un compromiso para ello.

4.1.4.3 Requerimientos tecnológicos.

4.1.4.3.1 Ambiente.

Se requiere tener acceso a los sistemas con los que actualmente se llevan a cabo los procesos involucrados. Así mismo es vital el poder tener acceso al equipo y a los Bases de Datos del grupo para que de esta forma el proyecto en desarrollo sea lo más cercano al final, esto es al real.

4.1.4.3.2 Requerimientos de calidad.

Los fallos que se pueden llegar a presentar con relación a los requerimientos de calidad de la solución a implantar son mínimas por no afirmar que nulas. Si se llegan a presentar serán más que fallas, operaciones de mantenimiento sin mayor repercusión.

4.1.4.3.3 Requerimientos de la solución general.

Se requiere que el sistema a desarrollar cubra con todos los requerimientos futuros en materia de desarrollo de nuevas aplicaciones para el grupo ICA. El desarrollo involucrado debe de ser lo más sencillo posible. Las aplicaciones a desarrollar deberán acceder en línea diferentes fuentes de información del grupo de una manera transparente y eficiente generando si así se requiere diversos reportes que se indiquen. La convivencia con software "multimedia" es un aspecto que en esas nuevas aplicaciones se debe de cubrir.

4.1.4.3.4 Limitaciones.

Se pueden presentar por limitaciones del software que se va a utilizar para realizar la implementación de la solución. Se confía en que éstas sean mínimas y en caso de presentarse se superen sin mayor dificultad.

4.1.4.4 Implementación de la propuesta.

Afortunadamente se puede afirmar que es menor la probabilidad de que alguna consideración adicional, de las planteadas hasta este momento, que se llegue a presentar pueda afectar la solución a desarrollar.

4.1.4.5 Beneficios pronosticados.

Los beneficios que a simple vista se detectan son los siguientes:

- Estandarización en el desarrollo de aplicaciones que la empresa necesite.
- Explotación más completa de las fuentes de información de la empresa.
- Substancial mejora en tiempo y manejo de los distintos procesos informáticos de la empresa.
- Interacción de aplicaciones con software de "multimedia".

4.1.5 Estrategia.

El presente trabajo se desarrollará siguiendo todas las indicaciones en cuanto a tiempo y a características del prototipo en la medida de lo posible atendiendo a su vez cualquier modificación o requerimiento adicional a los originales que se presente.

Así mismo se seguirá la metodologia DPM (Digital´s Program Methodology) en el desarrollo del sistema.

4.2 Diseño del Sistema.

4.2.1 Introducción.

4.2.1.1 Diseño.

El diseño es el primer paso en la fase de desarrollo de cualquier producto o sistema de ingeniería. Puede ser definido como : "... El proceso de aplicar distintas técnicas y

principios con el propósito de definir un dispositivo, proceso o sistema con los suficientes detalles como para permitir su realización física".

El objetivo del diseñodor es producir un modelo o representación de una entidad que será construida más tarde. El diseño de software para computadoras, como los métodos de diseño de ingeniería de otras disciplinas, cambia cantinuamente, conforme aparecen nuevos métodos, mejores análisis y un más amplio conocimiento. El diseño de software está en una etapa relativamente temprana de su evalución. Existen técnicas para el diseño de software, están disponibles criterios para cualificar un diseño y pueden aplicarse notaciones de diseño.

Una vez que se han establecido los requerimientos del software, la fase de desarrollo comprende tres pasos distintos: diseña, generación del código y prueba.

El diseño es el lugar en donde se asienta la calidad del desarrollo del programa, es la única forma mediante la que se puede traducir con precisión los requerimientos del cliente en un producto o sistema acabado. El diseño de programas sirve como base para todos los pasos de desarrollo y fase de mantenimiento que siguen.

El diseño de programas es un proceso mediante el que se traducen los requerimientos en una representación del software. El diseño del software se realiza en dos pasos. El diseño preliminar se refiere a la transformación de los requerimientos en datos y arquitecturo del software. El diseño en detalle se enfoca hacia los refinamientos de la representación arquitectónica que conduce a una estructura de datos detallada y a representaciones algorítmicas del software.

4.2.1.2 Fundamentos del Diseño.

En las últimas tres decadas se ha establecido un conjunto de conceptos fundamentales en el diseño de software. Todos han resistido la prueba del tiempo. Todos dan al diseñador de software una base desde la que pueden aplicarse metodologías de diseño más sofisticadas.

Estos fundamentos del diseño sirven todos para motivar los diseños modulares. La modularidad se ha convertido en un enfoque aceptado en todas las disciplinas de la ingeniería. Un diseño modular reduce la complejidad, facilita los cambios y da como resultado una más fácil implementación, posibilitando el desarrollo paralelo de diferentes partes de un sistema.

Uno de estos fundamentos es la cohesión. Un módulo coherente ejecuta una tarea sencilla en un procedimiento de software y requiere poca interacción con procedimientos que se ejecutan en otras partes de un programa. Un módulo coherente sólo debe hacer una

cosa. Un módulo que ejecuta tareas que están relacionadas lógicamente es coherente lógicamente.

En una aplicación no es necesario determinar el nivel preciso de cohesión. Es importante buscar una cohesión alto y reconocer la cohesión baja de forma que el diseño del software pueda modificarse, de manera que contenga una mayor independencia funcional.

Otro fundamento del diseño es el acoplamiento. El acoplamiento es una medida de la interconexión entre módulos en una estructura de programa. El acoplamiento depende de la complejidad de la interfase entre módulos, el punto en el que se hace una entrada o referencia a un módulo y los datos que pasan a través de la interfase.

En el diseño de software, se busca el más bajo acoplamiento posible. La conectividad sencilla entre módulos da como resultado un software que es más fácil de comprender y menos propenso a "efecto onda" causado cuando los errores ocurren en una posición y se propagan a lo largo del sistema.

4.2.1.3 Diseño Orientado al Flujo de Datos.

El diseño de datos es la primera de las tres actividades de diseño realizados durante la ingeniería del software: diseño de datos, diseño arquitectónico y diseño procedimental. El diseño de datos tiene una profunda influencia en la calidad del software. Existen varias metodologías de diseño los cuales hacen algunos intentos de enfocar los aspectos del diseño de datos. Los datos bien diseñados pueden conducir a una mejor estructura del programa, modularidad y reducción de la complejidad procedimental. El análisis de requerimientos y el diseño se solapan frecuentemente.

El diseño es un proceso multipaso en el que los representaciones de la estructura de datos, estructura de programa y procedimiento, se sintetizan a partir de los requerimientos de la información. El diseño está conducido por la información.

Un método de diseño es el orientado al flujo de datos. El objetivo del mismo es dar un enfoque sistemático para la derivación de estructuras de programa. El diseño orientado al flujo de datos define varias representaciones que transforman el flujo de información en estructura del programa.

Un método de diseño orientado al flujo de datos es particularmente útil cuando la información se procesa secuencialmente y no existe una estructura de datos jerárquica. Las técnicas de diseño orientadas al flujo de datos se aplican también a aplicaciones de procesamiento de datos.

El diseño orientado al flujo de datos permite una cómoda transición de las representaciones de la información o una descripción de diseño de la estructura del programa. La transición desde el flujo de información a la estructura, se realiza como parte de un proceso de cinco pasos: 1) se establece el tipo del flujo de información; 2) se indican los límites del flujo; 3) el Diagrama de Flujo de Datos se convierte en la estructura del programa; 4) se define la jerarquía de control mediante factorización, y 5) se refina la estructura resultante usando medidas y heurísticos (criterios) de diseño.

Un Diagrama de Flujo de Datos se convierte en una estructura de programa usando una o dos técnicas de análisis de diseño –análisis de transformación o análisis de transacción.

4.2.1.4 Diseño Orientado a las Estructuras de Datos.

La estructura de la información, llamado estructura de datos, se ha demostrado que tiene un importante impacto en la complejidad y eficiencia de los algoritmos diseñados para procesar la información. La estructura de la información predice muy bien la estructura del programa.

El diseño orientado a la estructura de datos transforma una representación de la estructura de datos en una representación del software. Los creadores del diseño orientado a la estructura de datos definieron un conjunto de procedimientos de "transformación" que utilizan la estructura de la información (datos) como guía.

En este método deben de realizarse siempre las siguientes tareas de diseño: 1) evaluar las características de la estructura de datos; 2) representar los datos en términos de formas elementales, tales como secuencia, selección y repetición; 3) transformar la representación de la estructura de datos en una jerarquía de control para el software; 4) refinar la jerarquía del software usando los criterios definidos como parte de un método, y 5) desarrollar finalmente una descripción procedimental del software.

El diseño orientado a la estructura de datos, se enfoca hacia el dominio de la información. Los métodos orientados a la estructura de datos, utilizan la estructura de la información como el conductor de la derivación del diseño.

4.2.1.5 Diseño Orientado al Objeto.

El diseño orientado al objeto da como resultado un diseño que interconexiona los objetos de datos y las operaciones de procesamiento, de forma que modulariza la información y el procesamiento en vez de sólo el procesamiento.

La naturaleza única del diseño orientado al objeto está ligada a su habilidad para construir, basándose en tres conceptos importantes de diseño del software: abstracción, ocultación de la información y modularidad. Sólo este método de diseño da un mecanismo que facilita al diseñador adquirir los tres sin complejidad o compromiso.

Este método llamado diseño orientado al objeto ha emergido durante los últimos 15 años. Hoy en día el diseño orientado al objeto se está usando en aplicaciones de diseño de software que van desde animación de gráficos por computadoras a las telecomunicaciones.

El diseño orientado al objeto crea un modelo del mundo real que puede ser realizado en software. Los objetos suministran un mecanismo para representar el dominio de la información, mientras que las operaciones describen el procesamiento asociado con el dominio de la información. Los mensajes dan la forma en la que se llama a las operaciones. La característica distintiva del diseño orientado al objeto es que los objetos "conocen" qué operaciones pueden aplicarse a éllos. Este conocimiento se adquiere combinando las abstracciones de datos y procedimentales en una única componente de programa, llamada objeto o paquete.

La metodología del diseño orientado al objeto consta de un método en tres pasos que requiere que el diseñador establezca el problema, defina una estrategia informal de solución y formalice la estrategia identificando los objetos y operaciones, especificando las interfases y dando los detalles de implantación para las abstracciones de datos y procedimientos.

El diseño orientado al objeto nos da una forma de suprimir las "subdivisiones" entre datos y procesos. Al hacerlo se puede mejorar la calidad del software.

4.2.1.6 Diseño Orientado a la Tecnología Cliente-Servidor.

La tecnología Cliente-Servidor proporciona el cimiento para construir las aplicaciones de información estratégicas que pueden ayudar de manera substancial al negocio. La arquitectura de 3-capas permite construir aplicaciones rápidamente, modificarlas fácilmente según las circunstancias del negocio dictan, e involucrar a los usuarios directamente en el diseño y desarrollo para asegurar así máximo uso y efectividad.

Una aplicación estratégica basada en arquitectura cliente-servidor puede ser diseñada, construida, e instalada en meses en lugar de años.

La arquitectura cliente-servidor involucra separar las siguientes funciones de la aplicación en componentes intercambiables o "capas":

. Presentación (interfase de usuario).

. Funcionalidad, conectividad y servidores de bases de datos.

. Datos. Esta capa puede incluir sistemas existentes y aplicaciones que han sido encapsulados para tomar ventaja de esta arquitectura.

Los usuarios interactúan sólo con la capa de presentación. Los "clientes" de presentación se comunican con los servidores intermedios, los cuales manipulan la interacción con las bases de datos o sistemas "heredados" existentes para manipular la información que el usuario requiere.

La capa de presentación proporciona los clientes de la interfase de usuario que soportan interacción computadora-humano a través de dispositivos como el teclado, el ratón y el monitor.

La capa de servicios es el corazón de la aplicación, donde el cómputo crítico se lleva a cabo. Los servidores realizan muchas funciones. El agregar una conección de datos adicional a la aplicación es tan simple como introducir un servidor más que se pueda comunicar con los datos nuevos.

La capa de datos manipula toda la información que la aplicación busca finalmente alcanzar y usar. Los clientes de la interfase de usuario en la capa de presentación sólo se comunican con los datos a través de los servidores intermedios apropiados. Los tipos de datos que pueden poblar esta capa no tienen restricciones. Cualquier aplicación puede ganar acceso a cualquier tipo de datos, en cualquier sistema, en cualquier lugar, siempre y cuando el servidor apropiado esté en su lugar.

4.2.2 La arquitectura Cliente-Servidor de 3-Capas en el diseño de la aplicación.

La aplicación se compone de 3 partes ya descritos en la sección 5.1. de este trabajo: programa para la Dirección de Recursos Humanos y Servicios Corporativos del Grupo ICA, programa para ICA Construcción Urbana, y programa para manipulación de multimedia. A

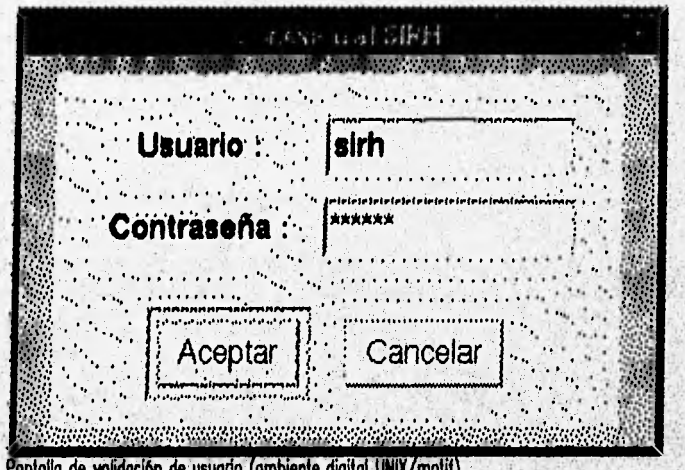
continuación se expone cada una de las 3 capas en las que se divide cada una de las 3 partes de la aplicación.

4.2.2.1 Capa de Presentación.

Los clientes correspondientes soportan interacción con el usuario a través del teclado, del "ratón", y del monitor.

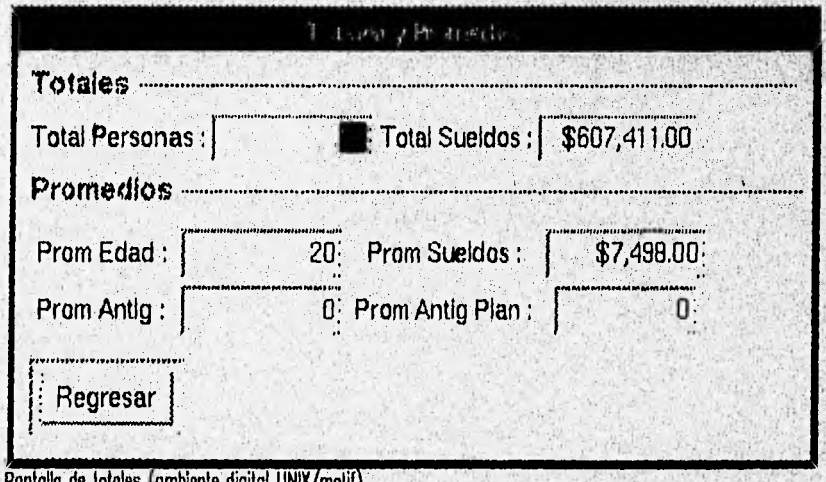
El "look and feel" de las tres interfases de usuario es el que se presenta a continuación:

Programa para la Dirección de Recursos Humanos y Servicios Corporativos del Grupo ICA.

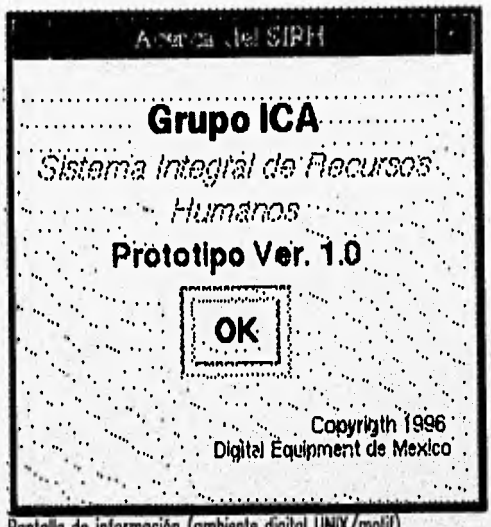


Pantalla de validación de usuario (ambiente digital UNIX/motif).

Pantalla principal (ambiente digital UNIX/motif).

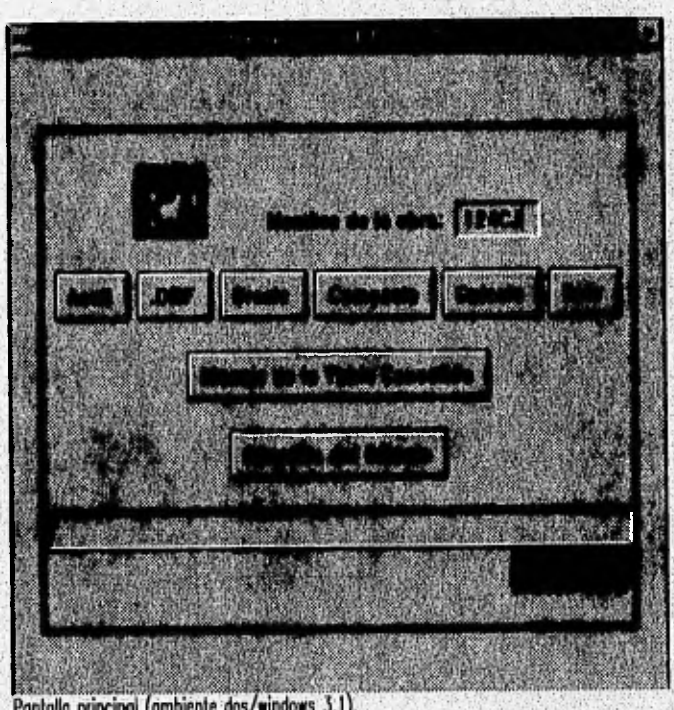


Pantalla de totales (ambiente digital UNIX/motif).

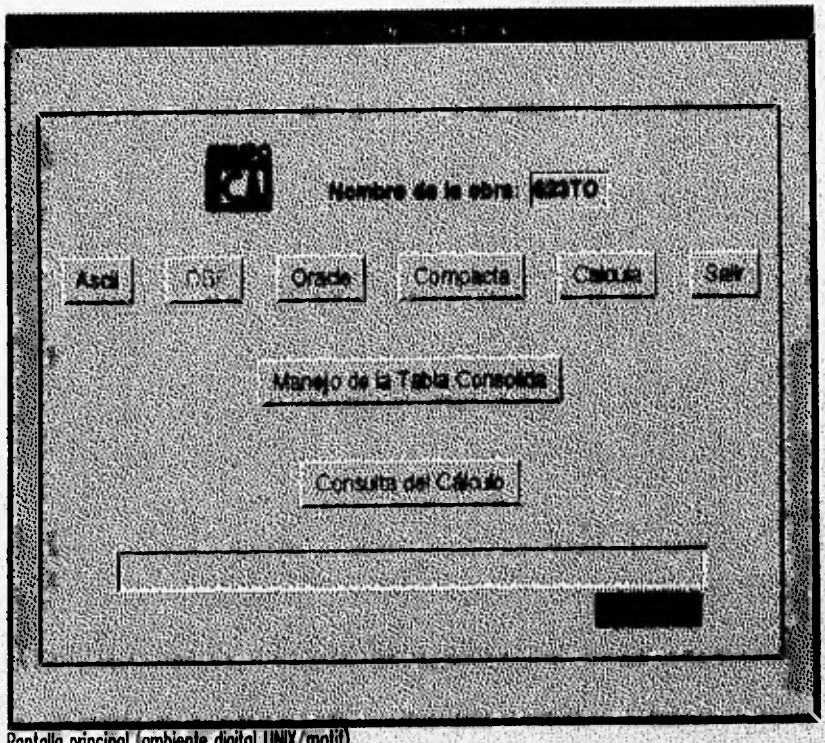


Pantalla de información (ambiente digital UNIX/motif).

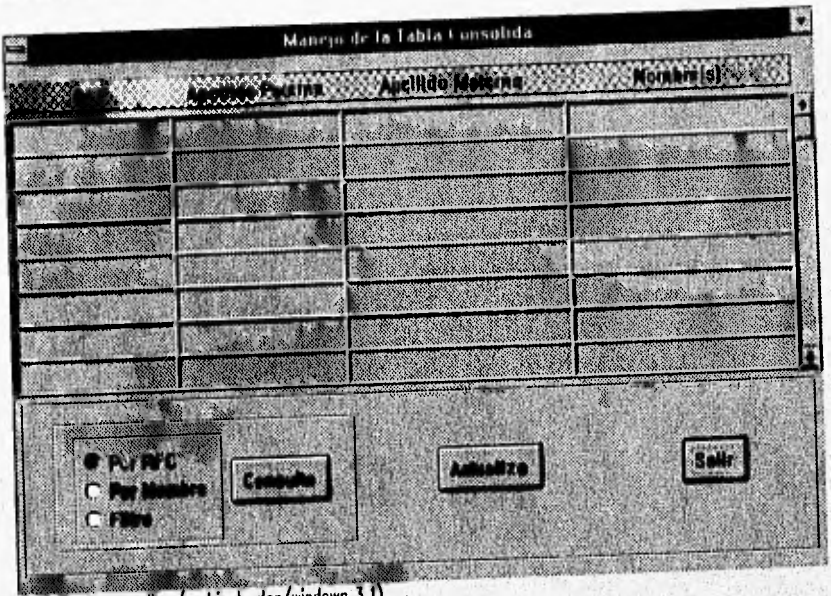
Programa para ICA Construcción Urbana.



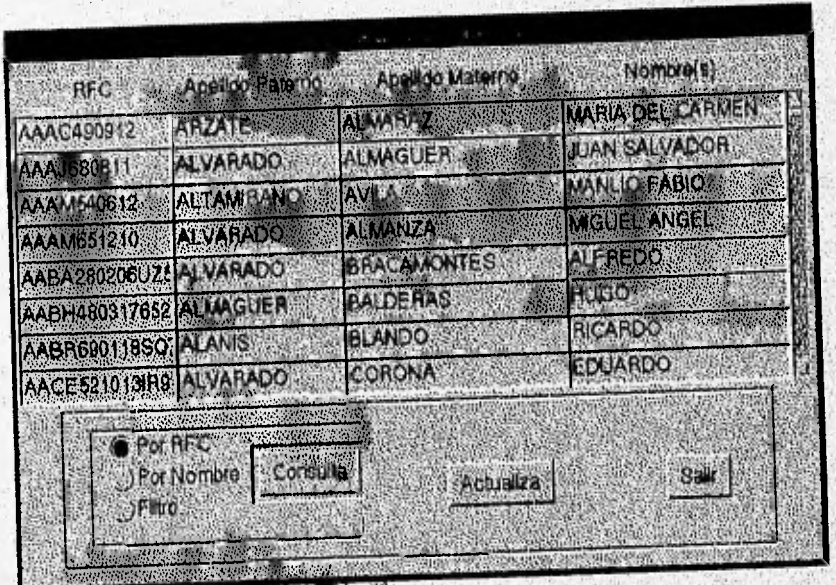
Pantalla principal (ambiente dos/windows 3.1).



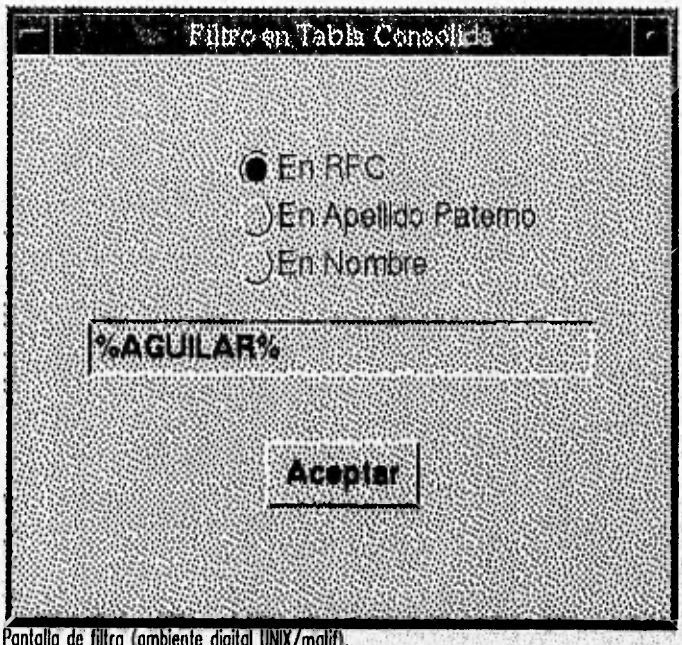
Pantalla principal (ambiente digital UNIX/motif).



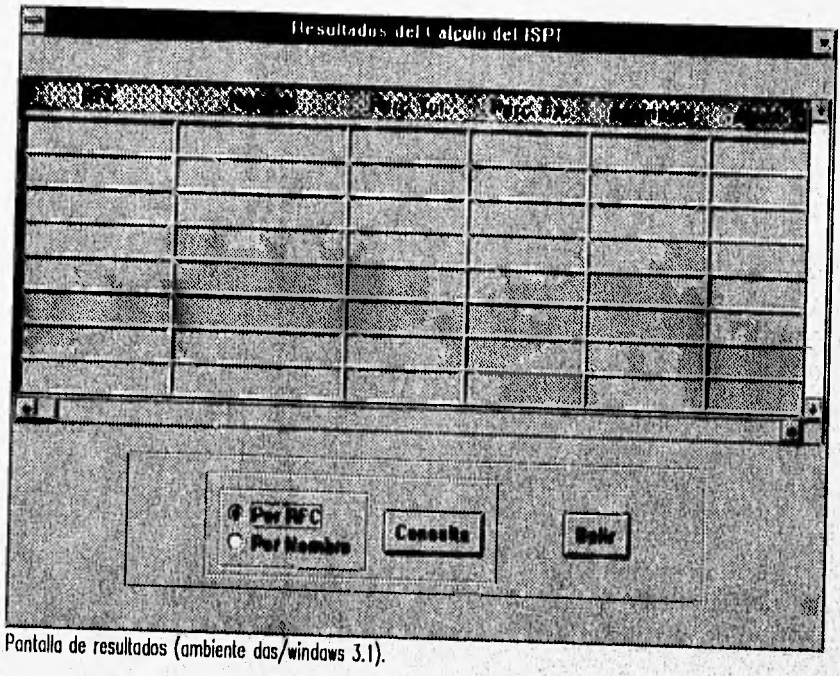
Pantalla de consultas (ambiente dos/windows 3.1).



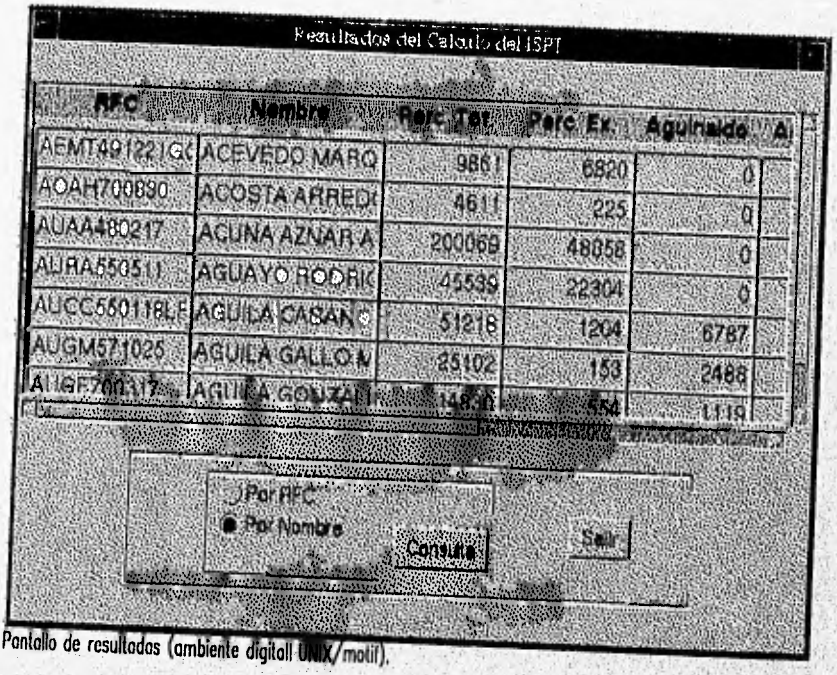
Pantalla de consultas (ambiente digital UNIX/motif).



Pantalla de filtra (ambiente digital UNIX/motif).



Pantalla de resultados (ambiente dos/windows 3.1).



Pantalla de resultados (ambiente digital UNIX/motif).

Programa para manipulación de multimedia.

Pantalla principal (ambiente dos/windows 3.1).

Pantalla de video (ambiente dos/windows 3.1).

4.2.2.2 Capa de Servidores.

Programa para la Dirección de Recursos Humanos y Servicos Corporativos del Grupo ICA.

Existe un servidor de bases de datos. La función que realiza es la siguiente:

. Proporciona acceso a la base de datos de Recursos Humanos.

Programa para ICA Construcción Urbana.

Contempla dos servidores, uno de bases de datos y otro de cálculos. En particular, la tarea que llevan a cabo es:

. Proparciana acceso a la base de datos de ICA Canstrucción Urbana.

. Efectua el cálculo del ISPT para cada uno de los trabajadores de la empresa.

Programa para manipulación de multimedia.

Contiene un servidor de imágenes. El trabajo que efectúa es relativamente sencillo y es el que se cita a continuación:

. Llama a ejecutar una aplicación que se encarga de desplegar un video.

4.2.2.3 Capa de Datos.

Programa para la Dirección de Recursos Humanos y Servicios Corporativos del Grupo ICA.

Está compuesto por la siguiente fuente:

. Una tabla de Oracle.

La descripción de la tabla es la siguiente:

SQL> describe todoica

Name	Null?	Type
UN		VARCHAR2(2)
EMPRESA		VARCHAR2(8)
OBRA		VARCHAR2(5)
RFC		VARCHAR2(13)
NOMBRE		VARCHAR2(40)
EDAD		NUMBER(3)
F_SUELDO		DATE
SUELDO		NUMBER(8)
CVE_SUELDO		VARCHAR2(1)
POR_SUELDO		NUMBER(2)
EVAL1		VARCHAR2(2)
EVAL2		VARCHAR2(2)
EVAL3		VARCHAR2(2)
ESCOLAR		VARCHAR2(30)
CVE_ESC		VARCHAR2(5)
CVE_INST		VARCHAR2(30)
ESTATUS		VARCHAR2(1)
F_PUESTO		DATE
CVE_PTO		VARCHAR2(8)
PUESTO		VARCHAR2(20)
F_ING		DATE
ANTIG		NUMBER(3)
PLAN		VARCHAR2(1)
SOCIO		VARCHAR2(5)
F_PLAN		DATE
POS_TAB		NUMBER(1)
IMSS		VARCHAR2(13)
SEXO		VARCHAR2(1)
EDO_CIVIL		VARCHAR2(1)
NO_DEPEND		NUMBER(2)
F_NACI		DATE
NO_EMP		VARCHAR2(5)
F_ING_EMP		DATE
ANT_PLAN		NUMBER(3)

Programa para ICA Construcción Urbana.

Incluye los tipos de datos que se listan en seguida:

. Una tabla de Oracle.
. Archivos .dbf.
. Un archivo ASCII.

La descripción de la tabla de Oracle, a la cual se sujetan los archivos .dbf así como los archivos ASCII, es la que se da a continuación:

SQL> describe obra

Name	Null?	Type
OBRA	NOT NULL	VARCHAR2(5)
PATERNO		VARCHAR2(40)
MATERNO		VARCHAR2(40)
NOMBRE11	NOT NULL	VARCHAR2(40)
CLAVE	NOT NULL	VARCHAR2(2)
ALTA		NUMBER(8)
BAJA		NUMBER(8)
IMSS		VARCHAR2(10)
PUESTO		VARCHAR2(30)
SUELDO	NOT NULL	NUMBER(9,2)
ZONA	NOT NULL	VARCHAR2(1)
FACTOR1	NOT NULL	NUMBER(8,5)
FACTOR2	NOT NULL	NUMBER(8,5)
STAT	NOT NULL	VARCHAR2(1)
D01		NUMBER(16,2)
D02		NUMBER(16,2)
D03		NUMBER(16,2)
D04		NUMBER(16,2)
D05		NUMBER(16,2)
D06		NUMBER(16,2)
D07		NUMBER(16,2)
D08		NUMBER(16,2)
D09		NUMBER(16,2)
D10		NUMBER(16,2)
D11		NUMBER(16,2)
D12		NUMBER(16,2)
D13		NUMBER(16,2)
D14		NUMBER(16,2)
D15		NUMBER(16,2)
D16		NUMBER(16,2)
D17		NUMBER(16,2)
D18		NUMBER(16,2)
D19		NUMBER(16,2)
D20		NUMBER(16,2)
D21		NUMBER(16,2)
D22		NUMBER(16,2)

D23		NUMBER(16,2)
D24		NUMBER(16,2)
D25		NUMBER(16,2)
D26		NUMBER(16,2)
D27		NUMBER(16,2)
D28		NUMBER(16,2)
D29		NUMBER(16,2)
D30		NUMBER(16,2)
D31		NUMBER(16,2)
D32		NUMBER(16,2)
D33		NUMBER(16,2)
D34		NUMBER(16,2)
D35		NUMBER(16,2)
D36		NUMBER(16,2)
D37		NUMBER(16,2)
D38		NUMBER(16,2)
D39		NUMBER(16,2)
D40		NUMBER(16,2)
D41		NUMBER(16,2)
D42		NUMBER(16,2)
D43		NUMBER(16,2)
D44		NUMBER(16,2)
D45		NUMBER(16,2)
D46		NUMBER(16,2)
D47		NUMBER(16,2)
D48		NUMBER(16,2)
D49		NUMBER(16,2)
D50		NUMBER(16,2)
D51		NUMBER(16,2)
D52		NUMBER(16,2)
D53		NUMBER(16,2)
D54		NUMBER(16,2)
D55		NUMBER(16,2)
D56		NUMBER(16,2)
D57		NUMBER(16,2)
D58		NUMBER(16,2)
D59		NUMBER(16,2)
D60		NUMBER(16,2)
D61		NUMBER(16,2)
D62		NUMBER(16,2)
D63		NUMBER(16,2)
D64		NUMBER(16,2)
D65		NUMBER(16,2)
D66		NUMBER(16,2)
D67		NUMBER(16,2)
D68		NUMBER(16,2)
D69		NUMBER(16,2)
D70		NUMBER(16,2)
D71		NUMBER(16,2)
D72		NUMBER(16,2)
FECHA11		NUMBER(8)
RFC11	NOT NULL	VARCHAR2(13)
CODIGO		VARCHAR2(10)

ISR_CAU		NUMBER(16,2)
SAC_CAU		NUMBER(16,2)
SNA_CAU		NUMBER(16,2)
GRA_EXE		NUMBER(16,2)
CRED		NUMBER(16,2)
O_PT		NUMBER(16,2)
O_PE		NUMBER(16,2)
O_AG		NUMBER(16,2)
O_ISR		NUMBER(16,2)
O_SAC		NUMBER(16,2)
O_SNA		NUMBER(16,2)
OP_DIAS		NUMBER(8)
FAC1	NOT NULL	NUMBER(8,5)
FAC2	NOT NULL	NUMBER(8,5)
RFC12	NOT NULL	VARCHAR2(13)
PER_GRA		NUMBER(16,2)
TOT_ING		NUMBER(16,2)

Programa para manipulación de multimedia.

Consiste de la siguiente información:

. Archivos .mov.

4.3 Implementación del Sistema.

4.3.1 Forté y sus Características Generales.

Forté es un ambiente de desarrollo de aplicaciones avanzado. El ambiente incluye facilidades para desarrollo, particionamiento, prueba, y ejecución de aplicaciones distribuidas.

Forté es una solución para construir aplicaciones cliente/servidor de alcance empresarial, queriendo decir con ésto aplicaciones que se ejecutan en ambientes heteragéneos complejos, con funcionalidad compleja de aplicaciones distribuidas, y que soportan grandes números de usuarios con altas exigencias de desempeño, confiabilidad y manejo.

Para cumplir en los 90s con los requerimientos de las aplicaciones cliente/servidor, Forté tiene características sobresalientes en cada una de las siguientes tres áreas:

. independencia de ambiente- incluyendo particionamiento y portabilidad de aplicaciones.

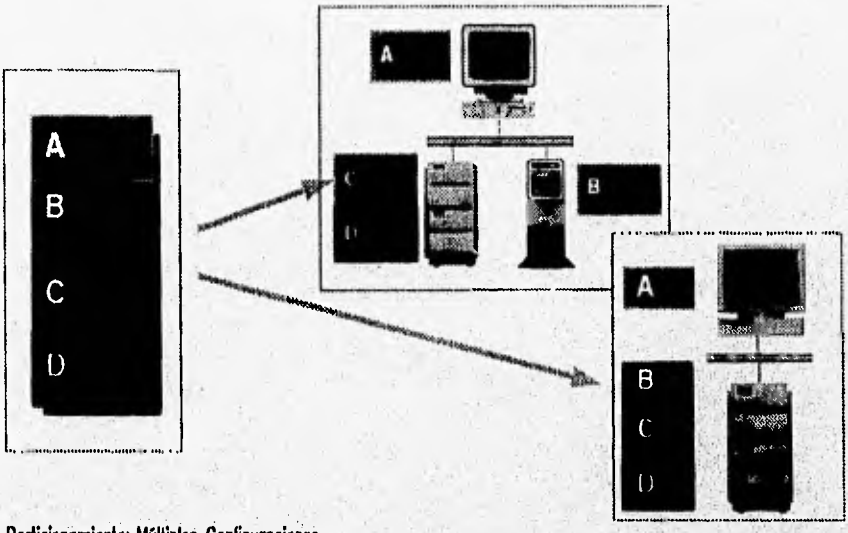
. capacidad de 3-capas de alcance empresarial- incluyendo servicios de aplicaciones compartidas, integración abierta y procesos de negocios manejados por eventos.

. soporte a sistemas de producción críticos- incluyendo confiabilidad, desempeño y manejo.

4.3.1.1 Independencia de Ambiente.

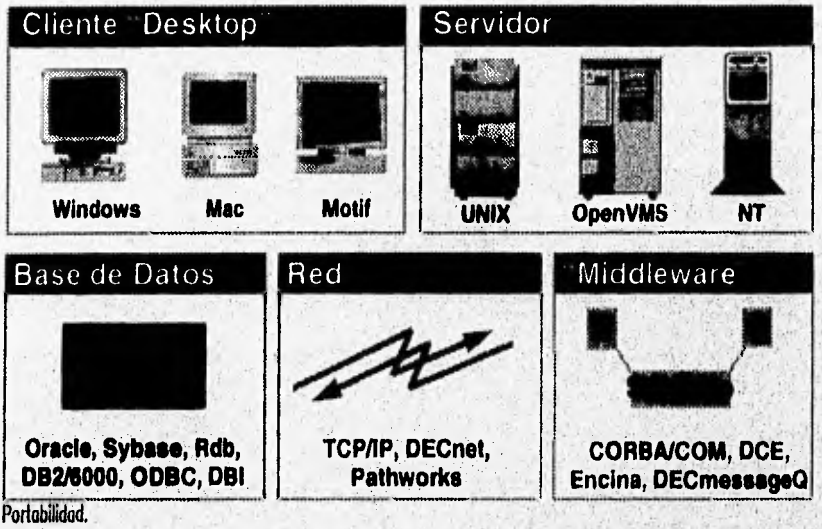
El desarrollar aplicaciones distribuidas que puedan correr en ambientes heterogéneos complejos puede costar un gran trabajo. Sin embargo Forté permite construir una aplicación concentrándose en la funcionalidad (la lógica) de la aplicación sin tener que decidir por adelantado cómo será desplegada esa aplicación a través de un conjunto específico de plataformas. Con Forté primero se construye una aplicación lógica. Después en un paso separado, Forté mapea esa aplicación lógica en cualquier ambiente heterogéneo especificado. No hay código específico para una plataforma que se requiera. Forté permite esta independencia de ambiente mediante dos capacidades poderosas: el particionamiento y la portabilidad.

Forté permite construir la aplicación lógico como una colección de servicios de aplicación. Forté después divide automáticamente a la aplicación lógica en componentes separados, llamados particiones, y mapea esas particiones en un ambiente de despliegue específico. La capacidad de particionamiento de Forté proporciona un mapeo por omisión de las aplicaciones lógicas en un ambiente de despliegue específico. Pero Forté también soporta múltiples configuraciones de particionamiento para una sola aplicación lógica: - Se puede particionar una aplicación para cualquier número de diferentes ambientes de despliegue. - Se puede cambiar la configuración de particionamiento por omisión para cualquier ambiente.



Particionamiento: Múltiples Configuraciones.

La capacidad de particionamiento de Forté se hace mucho más poderosa por el hecho de que las particiones de Forté pueden "correr" en un número muy grande de plataformas. Con forté se puede migrar el ambiente de soporte de la aplicación, o cambiar la mezcla de clientes "desktop", servidores, bases de datos, y redes, sin ninguna reprogramación de la aplicación. El soporte de plataformas de Forté proporciona una flexibilidad real y protege la inversión hecha. Un ejemplo particularmente interesante de portabilidad es la Interfase de Usuario Gráfica (GUI). Con Forté se puede desarrollar código de una GUI en una plataforma "desktop" MAC o Windows o Motif y la misma puede ser desplegada en cualquiera de estas plataformas sin ningún cambio. En ésto Forté soporta las herramientas nativas de cada plataforma en lugar de simplemente emular la vista. Para acomodar las diferencias entre los diferentes sistemas de desplegado, Forté emplea un esquema de manejo de geometría que preserva las relaciones entre los "widgets" (objetos) en la pantalla, mientras no deja de desplegarlos en su "look and feel" nativo. De esta manera, los usuarios pueden libremente cambiar de plataforma a plataforma. Con el enfoque de Forté, se tiene un equipo de programadores que desarrollan la aplicación lógica. Forté los protege de la complicada infraestructura computacional. Una vez desarrollada, la aplicación puede ser particionada y después desplegada en cualquier número de ambientes.



Portabilidad.

4.3.1.2 Capacidad de 3-Capas de Alcance Empresarial.

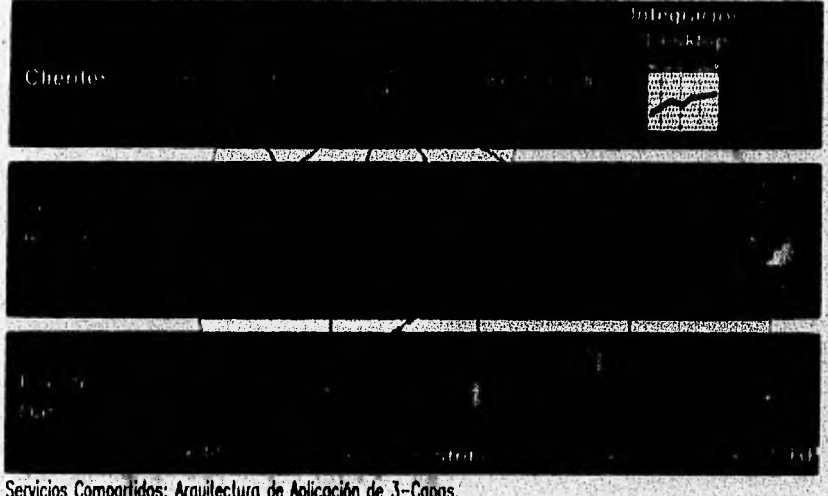
La independencia de ambiente de Forté satisface el reto de cada vez más ambientes complejos. Similarmente, la capacidad de 3-Capas de Alcance Empresarial de forté cumple con el reto de cada vez más aplicaciones complejas. Tres capacidades diseñadas para los complejas sistemas de tecnología de información de hoy en dia son: servicios de aplicación compartidos, integración abierta, y procesamiento dirigido por eventos

Un servicio de aplicación compartido es un servicio que puede ser compartido por múltiples usuarios y por múltiples aplicacianes. Los servicios de aplicación de Forté tienen capacidad multi-usuario incorporada. Forté tiene una capacidad de bloqueado que aplica a sus servicios de aplicación. Igualmente Forté soporta "multi-threading" de manera que varios usuarios puedan presentar peticiones a un servicio y Farté se pueda ocupar de calendarizarlas y ejecutarlas en la cantidad tiempo más pequeña. Forté también soporta transacciones en sus servicios de aplicación.

Los servicios de aplicación compartidos no sólo son compartidos por múltiples usuarios, sino también por múltiples aplicaciones. El escenario en el que múltiples usuarios y múltiples aplicaciones cliente comparten conjuntamente los mismos servicios, es lo que se llama una aplicación de alcance empresarial. Hay un número de ventajas al emplear una arquitectura de servicios compartidos para una aplicación; Reusabilidad.- Se puede construir un servicio y usarlo para diferentes aplicaciones. Mantenimiento.- Los combios se realizan en un solo lugar. No se tiene que ir a cada aplicación; en lugar de eso, las reglas

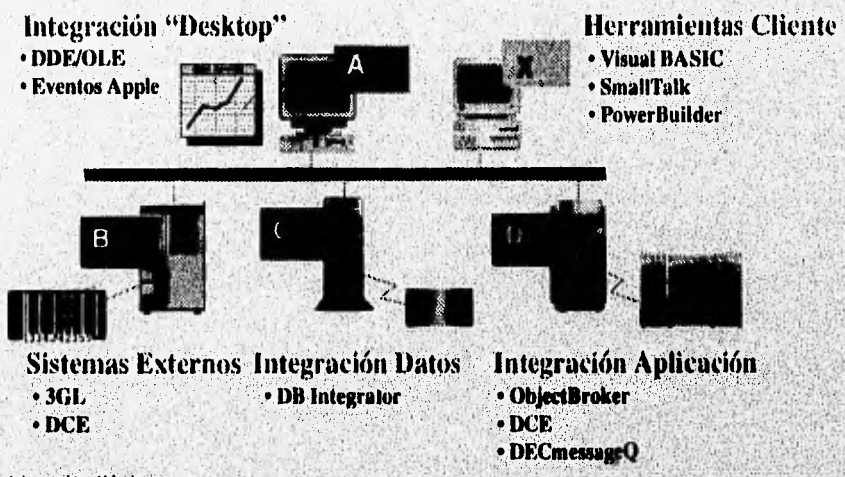
del negocio se encuentran en el servicio, el cual es compartido y reusado por todas las aplicaciones. Control.- Los componentes del cliente se basan en servicios compartidos que son mejor manejados ya que se ejecutan en servidores que están baja un cuidado más estrecho. Consistencia.- Se puede proporcionar una funcionalidad similar en todas las aplicaciones sin preocupación de reglas de negocio inconsistentes deslizándose en el sistema. Portabilidad.- Con Forté, los servicios son portables y re-localizables a través de todos los servidores en el ambiente.

Una arquitectura Cliente/Servidor de 3-Capas de segunda generación separa una aplicación en un conjunto de componentes lógicos que modelan e implementan el proceso del negocio. Componentes del Cliente.- Aquí es donde la colección y desplegado de datos se lleva a cabo. Servicios de Aplicación Compartidos.- En la capa de enmedio están los servicios del negocio reutilizables y compartidos que se pueden comunicar entre sí. Fuentes de Datos.- Puede haber una variedad de fuentes de datos. Forté es muy flexible y permite mezclar diferentes tipos de bases de datos. Éstas pueden ahora ser más efectivamente reusadas con esta arquitectura con una variedad de aplicaciones, eliminando la necesidad de tener copias duplicadas de la base de datos para cada sistema departamental. El particionamiento es la base para construir sistemas distribuidos de 3-Capas. Se diseña el sistema de la aplicación como un conjunto de servicios del negocio que exactamente modelan la forma en que el negocio marcha. Después se construye una aplicación lógica utilizando esos servicios. El particionamiento permite que estos servicios sean distribuidos a través de diferentes plataformas dentro de un ambiente de despliegue. Forté permite construir toda la funcionalidad de la aplicación primero con una herramienta y después simplemente particionarla. Forté proporciona verdaderos servicios de aplicación portables y compartibles.



Servicios Compartidos: Arquitectura de Aplicación de 3-Capas.

Adicionalmente a proporcionar servicios de aplicación compartidos, Forté soporta la integración de estos servicios con componentes existentes en un sistema cliente/servidor de alcance empresarial. Integración "desktop".- En el nivel de integración "desktop", Forté soporta DDE y OLE así como "Apple events". Herramientas del Cliente.- También en el nivel "desktop", Forté soporta integración de componentes de aplicación que son construidos afuera de Forté. Estos componentes pueden ser construidos en Visual-Basic, SmallTalk, o aún con PowerBuilder. Sistemas Externos.- Forté también soporta integración con muchas tipos de sistemas externos usando un enfoque 3GL de Forté. Con este enfoque se integra Forté con cualquier rutina externa del lenguaje C de programación. Integración de Datos.- Forté también tiene un número de opciones de integración con recursos de datos. Integración de Aplicaciones.- Forté también soporta integración con aplicaciones existentes, algunas veces llamados aplicaciones heredadas. Forté se puede integrar con cualquier aplicación heredada encapsulada con ObjectBroker.

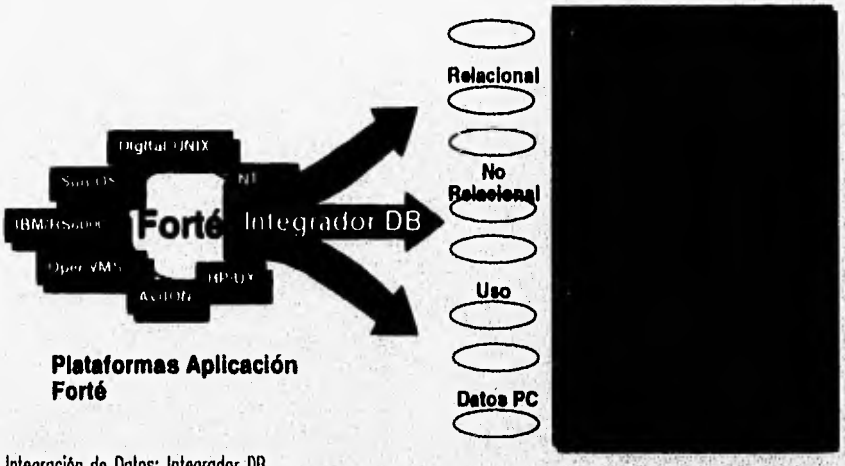


Integración Abierta.

Se pueden integrar aplicaciones de Forté con cualquier rutina de C externa a través de una técnica llamada "wrappering". Un "wrapper" 3GL es una pieza de código que se crea (usando herramientas proporcionadas por Forté) para acceder rutinas de C como si fueran cualquier otra pieza de código de Forté. Una vez que el "wrapper" está construido, se registra en el repositorio de desarrollo compartido de Forté. Una vez que está registrado como parte del repositorio, otros desarrolladores pueden usar ese "wrapper" transparentemente.

Además de proporcionar generación SQL para la mayoría de los sistemas de bases de datos, Forté también soporta acceso a otros recursos de datos a través del Integrados de Bases de Datos de Digital (DBI). Al usar el DBI con las aplicaciones de Forté, se puede acceder una amplia variedad de sistemas de manejo de datos existentes y sistemas de

archivos especializados, y hacer ésto con facilidad. Las operaciones de maneja de datos de tipo no SQL pueden también ser realizadas en aplicacianes de Forté. Llamadas de servicio normales pueden ser hechas a archivos planos y dispositivos especiales como lectores de código de barras y lineas de servicio de información.



Integración de Datos: Integrador DB.

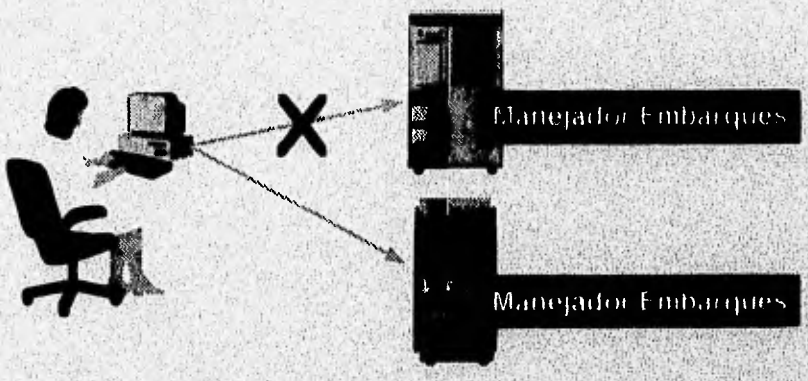
Forté proporciona integración con aplicaciones existentes a través de ObjectBroker, una implementación de Digital de la interfase CORBA basada en estándares. Las aplicacianes heredadas las cuales están encapsuladas en un "wrapper" de ObjectBroker pueden ser directamente accedidas por aplicaciones de Forté usando una interfase construida en el producto Forté. De esta forma, las aplicaciones de Forté pueden utilizar operaciones de la aplicación ObjectBroker como servicios distribuidos estándares de Forté.

Otro componente clave de la implementación de aplicaciones de alcance empresarial, es el uso de Forté del procesamiento manejado por eventos el cual permite llevar un negocio en una manera de tiempo real. Para esto Forté utiliza un modelo universal muy eficiente. Cualquier componente del sistema puede notificar a otro componente en tiempo real que algo de interés -un evento- ha sucedido, y en respuesta, el otro componente puede hacer algo aprapiado. Forté implementa el procesamiento manejado por eventos a través de un paradigma de subscripción y publicación. Si el usuario A está interesado en eventos publicados en la partición B, hay una sola sentencia escrita en el código del usuario A que le permite subscribirse a los eventos en la partición B. Igualmente, hay una sola sentencia que le permite a la partición B publicar eventos a sus subscriptores, como el usuario A. Forté se ocupa de todos los detalles que están enmedio.

4.3.1.3 Soporte a Sistemas de Producción Críticos.

Forté proporciona la confiabilidad, desempeño y manejo de aplicación necesarios para las operaciones del negocio fundamentales, especialmente según los sistemas de información continuan en expansión y/o responden a cambios.

Forté permite replicar un servicio y ejecutarlo como un servicio de respaldo en un servidor diferente. La réplica puede ser colocada en un sistema operativo completamente diferente en hardware, reenrutándose así automaticomente las peticiones de servicio del servicio caido al servicio de respaldo, no importondo dónde se encuentre. Todo esto es transparente para el usuario. De esto manero se brindo a lo aplicoción tolerancia a fallas y se praporciona la flexibilidad para decidir qué parciones de la aplicación son absolutamente críticas para el sistema, para de esta formo replicarlas para que estén protegidas en contra de cualquier falla del sistemo.



Replicación de Servicio para Tolerancia a Fallas

Confiabilidad: Tolerancia a Fallas.

Otra capacidad de confiabilidad clave es el soporte de Forté para las transacciones. Una transacción es una unidad de trabajo que puede involucrar el cambio de muchas piezas de datos. Si la transacción falla, no se puede permitir que cualquier dato cambie. Con Forté se puede reiniciolizar el estada de las datos en caso de que una transacción falle. Dentro del código de Forté se puede fijar el comienzo y el final de las transacciones, y dentro de estos límites, indicar qué datos se quieren incluir en la transacción.

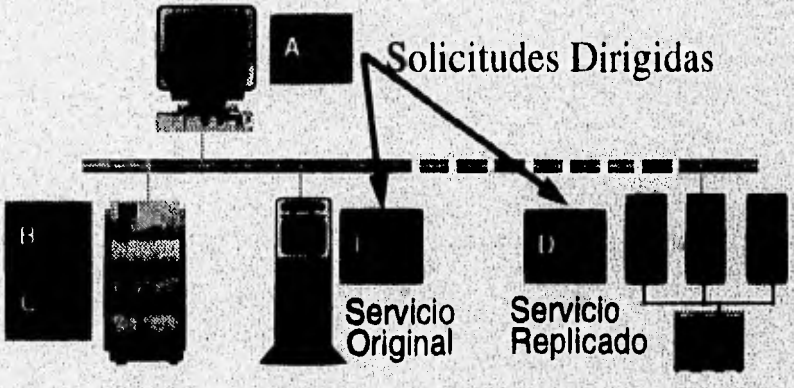
Otra dimensión de confiabilidad importante, es la habilidad de Forté de integración con otros servicios de la capa de enmedio.

La capacidad de particionamiento de Forté permite fácilmente redistribuir la aplicación, lo cual en ocasiones genera una tremenda mejora en el desempeño.

Farté por otra parte puede generar código fuente en C++ de su lenguaje de desarrollo de aplicaciones 4GL. El código en C++ es después compilado en código de máquina que se ejecuta a tiempo de "corrida", praparcionando con esta una ganancia en desempeño substancial.

El procesamiento en paralelo es otra importante fortaleza de desempeño de Farté. Una aplicación de Forté se puede ejecutar en múltiples computadoras, tanto cliente como servidor. En cada computadora, la aplicación se puede estar ejecutando en múltiples procesos; y cada proceso puede estar ejecutando múltiples "threads" así mismo. El procesamiento en paralelo es especialmente importante en aplicaciones distribuidas donde un cliente puede estar interactuando con múltiples servicios, distribuidos a través de multiples servidores.

Otra fortaleza del desempeño de Forté es su fácil escalabilidad. Si se tienen varios servicios en un servidor, una opción es colocar algunas de estos servicios en otros servidores. Así se puede replicar un servicio sobrecargado y balancear la carga entre el servicio original y el replicado. Farté se encarga de enrutar algunas peticiones al sevicio original y algunas otras al replicado. Se pueden colocar servicios replicados en un mismo o diferente servidor.



Replicación de Servicios para Balanceo de Carga

Desempeño: Escalabilidad.

Una tercera capacidad bajo el título general de soporte de un sistema de producción de misión crítica de Farté es el manejo de la aplicación. Forté ha incorporado tecnología muy poderasa que hace del manejo de aplicaciones algo muy sencillo. Con ésto se puede obtener una idea del tráfico de mensajes y sobre qué genera saturación en la red. Las herramientas de manejo del sistema de Farté ayudan a diagnasticar y corregir problemas de performance en una aplicación que se está ejecutando.

Otro aspecto importante del manejo de aplicaciones es el asegurarse de que cuando se actualiza un componente de una aplicación, esas versiones nuevas y viejas no se mezclen equivocadamente. Forté automaticamente verifica que todas las particianes de una aplicación sean consistentes entre sí, para que no se corra el riesgo, por ejemplo, de ejecutar una transacción contra reglas del negocio no actualizadas. También se pueden tener varias versiones instaladas y ejecutándose al mismo tiempo. Así que en Forté se pueden tener dos versianes instaladas simultaneamente y se pueden tener ambas trabajando en paralelo mientras se migra transparentemente la actual a una nueva versión.

4.3.2 Forté en la elaboración de la Aplicación.

4.3.2.1 Forté en la Capa de Presentación.

Forté cuenta esencialmente con dos talleres ("workshops") para el desarrollo de esta capa en la implementación de una aplicación, el Taller de Ventana y el Taller de Menú.

El Taller de Ventana proporciona un editor visual para crear las ventanas para ésta, la interfase de usuario. En el taller, se pueden construir y probar ventanas, aunque se necesita usar otro taller, el Taller de Menú que se describe a continuación, para crear la barra de menú para la ventana. El taller por sí mismo consiste de dos ventanas separadas: la ventana de paleta de herramientas y la ventana de usuario

La paleta de herramientas en el Taller de Ventana brinda un amplio rango de campos que se pueden usar para diseñar la forma. Los campos en esta paleta de herramientas incluyen controles estándares, tales como diferentes tipos de botones, listas, campos de texto, e imágenes, así como campos compuestas especiales tales como campos de tipo arreglo (los cuales despliegan información tabular) y campos de cuadrícula (los cuales permiten crear ventanas portables).

Para crear una ventana, simplemente se selecciona un artículo ("item") de la paleta y se coloca directamente en la forma de la ventana. Tal como un programa de dibujo , el taller permite colocar los campos en la forma moviéndolas y cambiándolos de tamaño según se necesita. Comandos especiales permiten alinear, formatear, y colorear los artículos que están en la forma.

El Taller de Menú proporciona un editor visual para crear la barra de menú para una ventana. Para crear o modificar una barra de menú, se construye una jerarquía que representa los menúes "pull-down" en la barra de menú. Cada menú puede incluir cualquiera de lo siguiente: -botones de menú, los cuales son controles simples que despliegan comandos. -switches de menú, que son controles simples que despliegan un

switch el cual puede ser prendido o apagado por el usuario. -listas de menú, las cuales son controles simples que despliegan un conjunto de opciones del cual el usuario final puede hacer una selección. -menúes deslizables.

Una ventana consiste de tres componentes básicos: -armazón de la ventana. -forma. -barra de menú. El armazón de la ventana es el borde de una ventana, el cual contiene controles (menúes y botones) para manipular la ventana, tales como para minimizar o cerrar la ventana. El tipo de la ventana determina si ésta tiene o no un armazón. La forma es el área de despliegue dentro del armazón de la ventana donde el usuario puede interactuar con la aplicación examinando e introduciendo datos, y haciendo selecciones y dando comandos. Todas las ventanas tienen formas. Una barra de menú de una ventana es aquella que contiene menúes "pull-down" para dar comandos de aplicación o cambiar características de la aplicación. Las barras de menú son opcionales.

Como ya se mencionó, las ventanas que se construyen en el Taller de Ventana son completamente portables. Para proporcionar un "look and feel" nativo, Forté usa el formato de ventana del sistema de ventanas en curso. De esta forma, la ventana de Forté toma tanto la apariencia familiar así como los controles de la ventana normalmente proporcionados por el sistema de ventanas. Por ejemplo, una ventana de Forté en un equipo Macintash despliega la barra de menú de la ventana activa en la parte de arriba de la ventana, separadamente de la ventana, mientras otros sistemas de ventanas despliegan la barra de menú para una ventana en la parte de arriba de la misma ventana. Si se está creando una interfase de usuario que se ejecutará en múltiples sistemas de ventanas, existen varios elementos de diseño que se deben de tomar en consideración cuando se planean las ventanas.

En el caso específico del prototipo del presente trabajo, las interfases de usuario realizadas se desplegaron en ambientes Motif y Windows 3.1 y 3.11. Estos ambientes se localizaron en equipos DEC 3000 y DECpc Lpv 466d2/HP Vectra DX respectivamente.

Forté tiene la capacidad de importar (respaldar) y exportar (restaurar) las aplicaciones (proyectos) que se realizan con el producto. Parte del código exportado de las diferentes partes del prototipo, el cual hace referencia a la capa de presentación, es el que se muestra a continuación:

Programa para la Dirección de Recursos Humanos y Servicios Corporativos del Grupo ICA.

```
class About_window is mapped inherits from DisplayProject.UserWindow


has public  method Init;
has public  method Display;

has property
        shared=(allow=off, override=on);
```

```
        transactional=(allow=off, override=on);
        monitored=(allow=on, override=on, default=off);
        distributed=(allow=off, override=off);

has
+5151463100000504000004239fe401010000210102000000000202060094070 0
+1d009fff01010421 0f1011120101010100009fff00000080000 0bb80fa00000
+0000000000000000010160600000000000000009d0bb89d0fa00001050303
+01fffd0000000000000c0405000600000201000100001200000100130300002 1
+0102000000005040000 1c0112124163657263612064656c20534952482e2e2e
+0506007a07004d009fff01010101 0f1011120101010100009fff010000080000
+5dbd5dc400000000035003030002040 4a0a0000000000000000009d5dbd9d5d
+c4000105066b010000005dbd5dc400060006000021020200000000708070600
+92070081029fff2121040121212121212121212100009fff120000089d04469d07
+53011205a1000000001a03030300050101a0a0000000000000000009d01129d
+05a1000103010909010150000000 9e00442024000900001c012c2c436f707972
+6967746820313939362 00a4469676974616c2045717569706d656e7420646520
+4d657869636f0a0806007c07000d009fff01010401 0f1011120101010100009f
+ff0c000008689cd005f7091f0000000003030303000501016060000000000000
+0000009d05f79d091f0001050a1001000000 5f7091f00040501009fff009fff
+00010100000b0000000000000404000000000000020c000d000e000f0010000a
+0000210502000000000c0d0e0f100b060003020100000000000000c0600920700
+81029fff2121040121212121212121212100009fff0d0000089d02620001070450
+000100013e3e030300080101a0a0000000000000000009d01079d04500001 03
+010911010 18c0100009e00442014001100001c010a0a477275706f204943410a
+0d060092070081029fff2121040121212121212121212100009fff0e000008009d
+010700e8091f000200012a0303030006 0101a0a0000000000000000009ce89d
+091f000103010912010178020000 9e00442014001200001c011d1d53697374 65
+6d6120496e74656772616c20646520526563757273 6f730a0e0600920700810 2
+9fff2121040121212121212121212100009fff0f0000089d02f59d01ef00e80334
+000300012a0303030008010 1a0a0000000000000000009ce89d033400010301
+0913010178020000 9e00442094001300001c0108084875 6d616e6f730a0f0600
+92070081029fff2121040121212121212121212100009fff100000069d019a9d02
+d700e805eb000400013c0303030008010 1a0a0000000000000000009ce89d05
+eb000103010914020100009e00442014001400001c011313507 26f746f746970
+6f2056657 22e20312e300a1006007807020102090 94f6b5f627574746f6e0102
+042121212121212121010001 9fff110000089d031f9d03b1023902d600050001
+040303030008010 1a0a0000000000000000009d02399d02d6000109020 24f4b
+000201141a7c879ca09cf19cfd9d01549d01879d01ff9d020e9d021a9d026f9d
+02809d02d49d02f89d034e9d035d9d03b19d03cb9d042342191a375c5d697c86
+678b9ca09cbd9ce29ce39cef9cf19cfb9cfd9d011a9d01439d01449d01469d01
+549d01589d01879d01a49d01ca9d01cb9d01d79d01f49d01ff9d02099d020e9d
+021a9d02379d025e9d025f9d02619d026f9d02739d02809d029d9d02c39d02c4
+9d02c69d02d49d02d89d02f89d03159d033d9d033e9d03409d034e9d03529d03
+5d9d037a9d03a29d03a39d03a59d03b19d03b59d03cb9d03f19d041a9d041b9d
+04230000
-031c291c


end class;
```

Programa para ICA Construcción Urbana.

```
class Filtro is mapped inherits from DisplayProject.UserWindow


has public  method init;
has public  method Display(output OpcFiltro: integer,
        output ResFiltro: Framework.TextData);

has property
        shared=(allow=on, override=on, default=off);
        transactional=(allow=on, override=on, default=off);
        monitored=(allow=on, override=on, default=off);
        distributed=(allow=off, override=off);

has
+5151463100000329000002909fe401010000210102000000000202060094070 0
+1d009fff01010421 0f10111201010117000 9fff0000000800000bb80fa00000
+0000000000000000010160600000000000000000009d0bb89d0fa00001050303
+01fffd0000000000000c0405000600000201000100000e00000100030300002 1
+0102000000000504000 1c01191946696c74726f20656e205461626c6120436f
+6e736f6c6964610506007a07004d009fff01010101 0f10111201010117000 9f
+ff0100000800005dbd5dc400000000035103030002040 4a0a000000000000000
+00009d5dbd9d5dc4000105066b010000005dbd5dc400060006000021030200 00
+0000070809070600f6070089020b0b546578746f46696c74726f01020b212121
+2121010101170001080854657874446174610c0000089d01af9d057901270b6c
+0000000040303030005010280a0000044000000000009d01279d0b6c000103
+020107001e040008ff8100009e0040001403020806 00b2070009020c0c4f7063
+696f6e46696c74726f0102042121212121010101170002070749 6e7465676572
+0d0000089d05039d01da02f606dd0000000004030303000501018080000000 00
+00000000009d02f69d06dd0001030101780003 0a0700020101 0a000021030200
+01a6010b0c0d0b0001a601010e000e00001c010606456e205246430c0001a601
+020f000f00001c011313456e204170656c6c69646f2050617465726e6f0d0001
+a6010310001000001c010909456e204e6f6d6272650906007807020102070741
+6365707461720102042121212121212121211700039fff0e0000089d05429d0788
+0239045b0000000004510303000501 0a0a0000000000000000009d02399d04
+5b000109070741636570746172000201101a7c879ca79cf89d01059d01739d01
+d99d01e69d01ee9d01fb9d02039d021d9d02259d02359d02902d191a375f5d69
+7c86878b9ca79cc49ce99cea9cf69cf89d01029d01059d01359d015e9d015f9d
+01629d01679d01739d01a39d01cc9d01cd9d01d29d01d69d01d99d01e39d01e6
+9d01ee9d01f29d01fb9d02039d02079d021d9d02259d02299d02359d02599d02
+829d02839d02900000
-013ccecb


end class;
```

Programa para manipulación de multimedia.

```
class W1 is mapped inherits from DisplayProject.UserWindow


has public  method Init;
has public  method Display;

has property
        shared=(allow=off, override=on);
        transactional=(allow=off, override=on);
        monitored=(allow=on, override=on, default=off);
        distributed=(allow=off, override=off);

has
+515146310000042c0000034a9fe401010000210102000000000202060094070 0
+1d009fff01010421 0f10111201010101000 09fff0000000800000bb80fa00000
+000000000000000001016060000000000000000009d0bb89d0fa00001050303
+01fffd0000000000000c0405000600000201000100001000000010013030000021
+0102000000000504000 01c0109095669646 56f744943410506007a07004d009f
+ff01010101 0f10111201010101000 09fff0100000800005dbd5dc40000000003
+51030300020404a0a0000000000000000009d5dbd9d5dc4000105066b010000
+005dbd5dc400060006000021050200000000708090a0b070600902070081029f
+ff2121040121212121212121212100009fff0c00000 89d03a89d015c00b302eb00
+0000003c03030300050101a0a0000000000000000009cb39d02eb00010301 09
+0c01056401000 09e00442014000c00001c010b0b706f722044696769746 16c08
+0600920700810 29fff2121040121212121212121212100009fff0d000008693501
+12049a000000003e030303000501 01a0a0000000000000000009d01129d049a
+00010301090d01059ca00100009e00442014000d00001c01090956696465 6f74
+494341090600b4070009020505566964656f01020421212121210101010001
+0808546578744617461 0e0000089d012e9d02cf012704310000000004050303
+0005010180a0000000000000000000009d01279d043100010302010300e03000304
+0e00002105020001a6010f101112130f0001a601011400140000 1c0107076b65
+6e6e65647910000 1a601021500150000 1c01060661706f6c6c6f110001a60103
+16001600001c0105056c656e696e120001a601041700170000 1c0106067772 69
+676874130001a601051800180000 1c0106066 16d656c69610a06007807020102
+0808566572426f746f6e01020421212121212121210100029fff0f000008699d
+0472023902eb0000000004030303000501 01a0a0000000000000000009d0239
+9d02eb00010903035665720002010b060078070201020a0a53616c6972426f74
+6f6e01020421212121212121210100039fff100000089d034a9d047202390334
+0000000004030303000501 01a0a0000000000000000009d02399d0334000109
+050553616c697200020118 1a7c87979ce89cf79d014d9d015f9d01b39d01c39d
+02209d022f9d02379d02459d024d9d025a9d02629d026e9d02769d02839d028b
+9d02989d02ee9d034a3f191a375c5d697c86878b979cb49cd99cda9ce69ce89c
+f29cf79d01149d013c9d013d9d013f9d014d9d01519d015f9d017c9d01a19d01
+a29d01a49d01b39d01b79d01c39d01ed9d02169d02179d021a9d021e9d02209d
+022a9d022f9d02379d023b9d02459d024d9d02519d025a9d02629d02669d026e
+9d02769d027a9d02839d028b9d028f9d02989d02bd9d02e49d02e59d02ee9d03
+159d033e9d033f9d034a0000
-02189f1a


end class;
```

4.3.2.2 Forté en la Capa de Servicios.

Una aplicación de Forté se compone de objetos. Todos los servicios distribuidos con los que la aplicación interactúa son objetos. Una Base de Datos que se puede llegar a necesitar acceder desde la aplicación es un objeto. Igualmente, una aplicación 3GL existente que se desee usar es un objeto. Para interactuar con estos objetos, se invocan métodos sobre ellos, tal y como se haría con otra cualquier objeto de la aplicación.

En una aplicación distribuida, cada objeto tiene un sólo lugar fijo. Cuando un método es invocado en un objeto, éstos se ejecutan en la máquina en la que el objeto se localiza. Normalmente, el objeto se localiza en la máquina en la cual fue creado.

No obstante que la aplicación puede consistir de miles de objetos, sólo algunos necesitan estar en lugares particulares. Los objetos que necesitan estar en un lugar en particular abarcan: –un objeto que representa un recurso externo existente, tal y como un sistema de manejo de Base de Datos o un servicio 3GL que ya está presente en un máquina en particular. –un objeto que define un servicio del negocio compartido. –un objeto que define un servicio que se desea replicar para proporcionar tolerancia a fallas o balanceo de carga de trabajo.

Cuando se está listo para dividir la aplicación en particiones, estos servicios centrales son los únicos objetos con los que se necesita trabajar.

Cuando se crea un proyecto, se debe de especificar qué objetos en la aplicación necesitan ser localizados en particiones específicas. Para hacer esto, simplemente se crean y nombran objetos de servicio. La localización del resto de objetos en la aplicación no importa. Si un objeto de servicio crea otros objetos, éstos se localizarán en la misma partición que la de su creador. Y Forté proporciona acceso completamente transparente a estos objetos.

Un objeto de servicio es simplemente un objeto con nombre que se puede referenciar desde cualquier método en la aplicación. Se puede en un objeto de servicio como una variable global, con la excepción de que se especifica su valor en tiempo de compilación. Cuando se crea un objeto de servicio, se le da un nombre, una clase, y valores para sus atributos.

Después de crear un objeto de servicio, se puede trabajar con él como con cualquier otro objeto. Si se quiere invocar un método en un objeto de servicio que está localizado en una partición remota, se puede invocar el método como se haría con cualquier otro objeto. El método se ejecuta en la máquina donde el objeto se localiza, y Forté devuelve el valor de regreso y los parámetros de salida a la partición donde se invocó el método. No obstante que no se puede reasignar el valor mismo del objeto de servicio, se

pueden cambiar los valores de sus atributos individuales tal y como se fijaría el valor de cualquier atributo de un objeto.

Un tipo especial de objeto de servicio es un manjeador de recursos de un Sistema Manejador de Base de Datos (DBMS). Un servicio de manejo de recursos de un DBMS proporciona acceso al sistema de manejo de la Base de Datos. Definiendo un servicio de este tipo, se puede interactuar con la Base de Datos como con cualquier otro objeto. Simplemente se invocan métodos en el objeto de servicio para empezar una sesión de Base de Datos, extraer datos, actualizar datos, etc.

El objeto de servicio se crea como parte del proyecto. Posteriormente, se pueden asignar los objetos de servicio a particiones específicos.

Una de las ventajas más importantes de usar objetos de servicio es la capacidad de hacer múltiples copias de los mismos (replicación) para balancear la carga de trabajo y/o soportar la tolerancia a fallas.

Haciendo un breve resumen de lo que se ha citado hasta este momento sobre Forté en la capa de servicios, se tiene lo siguiente. Los objetos de servicio brindan las bases para crear una aplicación distribuida. Todos los servicios distribuidos con los que la aplicación interactúa están representados por objetos de servicio. Para acceder una Base de Datos desde una aplicación, se debe de crear un objeto de servicio para representar esa Base de Datos. Un objeto de servicio es un objeto con nombre que representa un recurso externo existente, un servicio del negocio compartido por Forté el cual se comparte por usuarios múltiples, o un servicio que se desea replicar para brindar tolerancia a fallas o balancear la carga de trabajo. El objeto de servicio contiene información necesaria por el servicio así como operaciones que el servicio puede realizar.

Por otra parte, en Forté existen tres tipos de objetos de servicio: Manejador de Recursos de una Base de Datos, Sesión de Base de Datos, Clase TOOL.

Los objetos de servicio del tipo Manejador de Recursos de una Base de Datos representan una instalación de un DBMS. Antes de que se pueda accesar un sistema de manejo de base de datos desde Forté, el manejador del sistema debe definir un manejador de recursos que represente al sistema de base de datos en particular según se instale en un nodo en especial. Después de que el manejador de recursos se define, se puede crear un objeto de servicio del tipo tratado aquí, el cual le permite a la aplicación de Forté interactuar con el manejador de recursos. Una vez que la clase del objeto de servicio del tipo Manejador de Recursos de una Base de Datos está creada, se puede iniciar conecciones múltiples (sesiones) a la Base de Datos que representa.

Un objeto de servicio de Sesión de Base de Datos representa una sesión en particular con un manejador de recursos de una base de datos. Antes de que se pueda

accesar un sistema manejador de base de datos desde Forté, el manejador del sistema debe definir un manejador de recursos que represente el sistema de base de datos en particular según se instale en un nodo en particular. Después de que el manejador de recursos se defina, se puede crear un objeto de servicio del tipo DBSession, el cual le da a la aplicación de Forté una sesión de base de datos para el manejador de recursos en particular. Cuando se crea un objeto de servicio del tipo Sesión de Base de Datos, se debe de especificar el nombre del manejador de recursos para el cual se está arrancando la sesión. Para ésto, el manejador del sistema debe de asignar un nombre de manejador de recursos a cada instalación individual de un DBMS cuando define el ambiente de despliegue. El nombre de base de datos para un objeto de servicio de este tipo es la base de datos específica que se desea accesar. Cuando se crea el objeto de servicio, se tiene la opción de especificar la base de datos en especial para la que se quiere iniciar una sesión. Cuando se crea el objeto de servicio del tipo Sesión de Base de Datos, se especifica el nombre y el "password" del usuario para la sesión. Debido a que se usa un objeto de servicio de este tipo para cualquier número usuarios desconocidos, se debe configurar un nombre de usuario genérico en el DBMS de manera que éste proporcione los privilegios que la aplicación necesite.

TOOL es el lenguaje de programación de cuarta generación orientado a objetos de Forté. Un objeto de servicio del tipo Clase TOOL es cualquier servicio compartido de Forté que se define con una clase que el usuario diseña y desarrolla, o con una clase proporcionada en una librería de Forté o por un proyecto que la ofrezca. Debido a que un objeto de servicio interactua con objetos en particiones remotas, la clase desde la cual se crea un objeto de servicio debe de estar distribuida. Cuando se crea un objeto de servicio del tipo Clase TOOL se pueden especificar valores iniciales para cualquiera de los atributos públicos del objeto que tengan tipos de datos simples. Cualquier atributo público para el cual no se especifique un valor se inicializa a sus valores por default.

Los servicios que se definieron para el prototipo del presente trabajo se encontraron localizados en un equipo HP9000/842 y en otro HP9000/827S.

La parte del código de Forté en donde se definen los servicios que crearon es la que se muestra enseguida:

Programa para la Dirección de Recursos Humanos y Servicios Corporativos del Grupo ICA.

```
-- START SERVICE OBJECT DEFINITIONS
servica ICADBMgr : GenericDBMS.DBResourceMgr = (DialogDuration = SESSION,
        Visibility = environment,
        ExternalManager = 'ICA_Source',
        FailOver = FALSE,
        LoadBalance = FALSE) HAS PROPERTY extended = (UUID = '87C031A2-3654-11CE-
9B0D-9AF951BAAA77');
service DoSql : ICADEMO.ICASqlManager = (DialogDuration = SESSION,
        Visibility = environment,
```

```
        FailOver = FALSE,
        LoadBalance = FALSE) HAS PROPERTY extended = (UUID = '89743160-3654-11CE-
9B0D-9AF951BAAA77');
-- END SERVICE OBJECT DEFINITIONS
```

Programa para ICA Construcción Urbana.

```
-- START SERVICE OBJECT DEFINITIONS
service DefaultDBSession : GenericDBMS.DBSession = (DialogDuration = SESSION,
        Visibility = environment,
        ExternalManager = 'ICA_Source',
        ResourceName = 'rechuman.sirh',
        UserName = 'demo',
        UserPassword = 'demonlaco',
        FailOver = FALSE,
        LoadBalance = FALSE) HAS PROPERTY extended = (UUID = 'F37FF1B5-36F0-11CE-
94D5-53543D0CAA77');
service Calcular : Impuestos.AuxDivide = (DialogDuration = MESSAGE,
        Visibility = environment,
        FailOver = TRUE,
        LoadBalance = FALSE) HAS PROPERTY extended = (UUID = '337AFF87-6C93-11CF-
8682-E11102EBAA77');
-- END SERVICE OBJECT DEFINITIONS
```

4.3.2.3 Forté en la Capa de Datos.

Forté soporta interfases a un número de sistemas de base de datos: -DB2. -Servidor Dinámico En-Línea de Informix. -Oracle. -Rdb (tanto Oracle Rdb como su predecesor, DEC Rdb). -Sybase. -Sistemas de manejo de base de datos accesibles por ODBC.

Las aplicaciones de Forté accesan sistemas de base de datos usando sentencias y clases de SQL TOOL definidas en la librería del DBMS Genérico. La forma más sencilla de accesar una base de datos es usar sentencias de SQL TOOL. Se pueden escribir aplicaciones portables usando construcciones SQL estándares que todos los vendedores soportan. También se puede explotar la funcionalidad específica de algún vendedor, como uniones externas o conjuntos de operaciones complejas, pero perdiendo con ésto la portabilidad de la aplicación a través de diferentes bases de datos.

Para construir aplicaciones que ejecutan instrucciones de SQL dinámico, se usan las clases en la librería del DBMS Genérico. Para usar cualquier sentencia de SQL TOOL, o cualquier referencia a las clases de la base de datos, se debe agregar la librería del DBMS Genérico como una librería suministradora en el Taller de Proyecto.

La interfase ODBC (Open Database Connectivity) de Microsoft es un estándar emergente que le permite a las aplicaciones accesar datos desde una variedad de sistemas

manejadores de bases de datos. Las aplicaciones se desarrollan, compilan, y envían independientemente del DBMS en uso. En tiempo de ejecución, la librerías de liga-dinámica, llamadas "drivers", se ligan según se requiere para accesar una fuente de datos específica mediante un método de comunicación específico.

Uno de los beneficios de ODBC es que proporciona una interfase al Servidor SQL de Microsoft por medio de ODBC en Windows/NT, así como a ciertas bases de datos (tales como Access de Microsoft) en Windows/DOS. Estas interfases permiten hacer prototipos aplicaciones enteras de bases de datos en una máquina cliente "stand-alone".

Algunas características de un DBMS disponibles a través de Forté, tales como procedimientos de base de datos y tipos de datos binarios, no son soportadas por algunos "drivers" ODBC. En este caso cuando características no soportadas del DBMS son referenciadas desde aplicaciones de Forté, el producto genera una excepción.

En el prototipo expuesto en este trabajo, las fuentes de datos se ubicaron en un equipo HP9000/827S.

Conclusiones.

El prototipo implementado cumplió satisfactoriamente con las expectativas planteadas al inicio de este trabajo.

Se creó un sistema fácil de modificar, el cual de hecho se corrigió tantas veces como fue necesario durante su desarrollo, quedando así una aplicación que nos entrega información totalmente actual en el momento que así se requiere.

El sistema se adaptó sin ningún problema al equipo disponible tanto en la fase de desarrollo como en la de ejecución y si en un momento dado se presentará el requerimiento de adaptación del sistema a un equipo adicional o a uno totalmente distinto al usado, esto no impactaría en absoluto al sistema gracias a las características propias del Ambiente de Desarrollo de Aplicaciones Avanzado Forté, software con el cual se desarrolló el mismo.

Uno de los objetivos principales en el prototipo desarrollado era evaluar una herramienta con la cual se permitiera la creación de aplicaciones con las que se uniformizara el manejo de información en las diversas empresas del Grupo ICA. Este objetivo se considera cumplido al haberse consolidado en el prototipo la manipulación de una significativa parte de la información del Grupo poniéndola disponible para cualquier área del mismo que así lo solicitara.

El acoplamiento del sistema con las aplicaciones ya existentes fue total, queriendo decir con esto que no se tuvo que modificar de ninguna manera la forma en que se venían dando o ejecutando los procesos del Grupo ICA involucrados en el prototipo, quedando a su vez abierta la posibilidad de una migración paulatina de los procesos existentes por unos nuevos o de una integración de algunos otros.

Forté nos brinda capacidades de desempeño y confiabilidad de verdad notables. No importando que en el prototipo desarrollado no se pudieran mostrar adecuadamente, éstas están disponibles para ser explotadas en cuanto sean requeridas.

Por otra parte el sistema desarrollado, apoyado en Forté, es sin lugar a dudas un muy buen y claro ejemplo llevado al mundo real de lo expuesto por John J. Donovan en su Arquitectura Cliente/Servidor de 3-Capas.

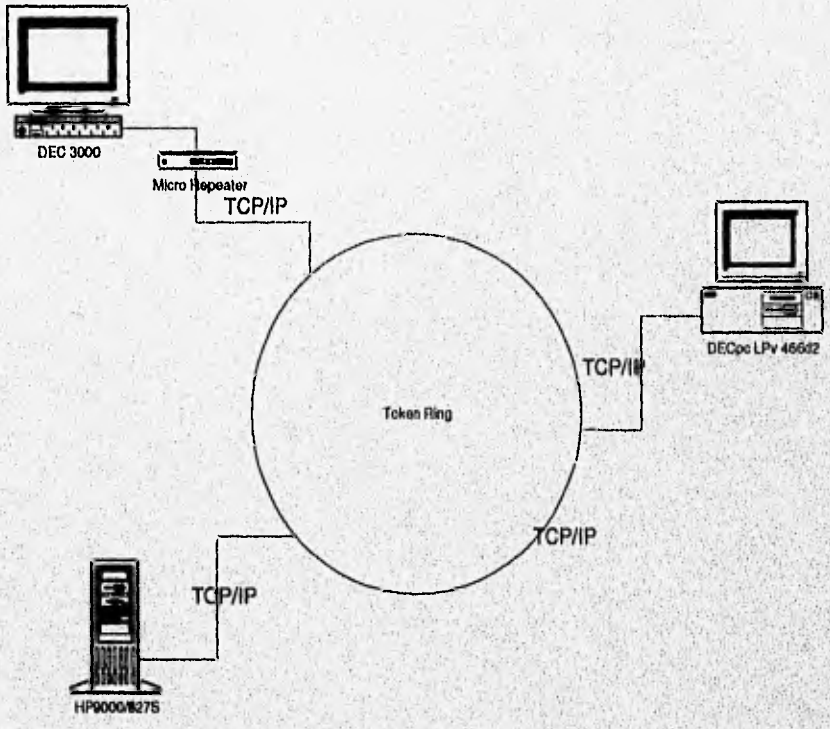
Las posibilidades que se aprecian en el campo de la Reingeniería de Software en estrecha relación con la manera en que hasta este momento se hace uso de la información en la mayoría de las empresas mexicanas son sencillamente enormes. Sin embargo parece ser que desafortunadamente todavía se tendrá que esperar para estar lo suficientemente preparados de manera que esta revolución tecnológica se pueda dar en nuestro país.

No abstante el cloro éxito obtenido con la realización del prorotipo, el cual se acaba de exponer, el sistema creado no llegó a la etapa de puesta en producción. Lo razón principal de este aparente fracaso confirmadamente quedó totalmente fuera del área técnica y se puede resumir con esta sencilla frase: el producto de software Forté quedó grande para las necesidades del Grupo ICA. Parece ser que desde un inicio no se dimensionó bien el amplio alcance de Forté y el haberlo adoptado como la solución a los requerimientos del Grupo hubiera sido, como lo mencionó un integrante de ICA, "querer matar moscas a cañanazos". Sin embargo, se reitera que lo realizado en esta aplicación no tiene precedente convirtiéndose así en una importante referencia en esta materia.

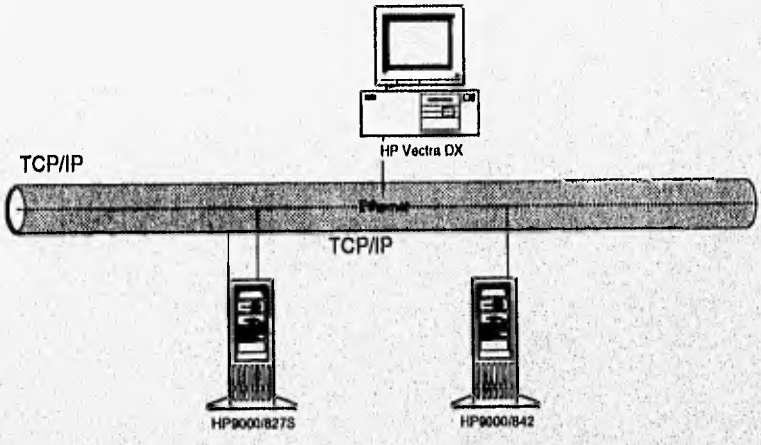
El futuro para este tipo de desarrollos está completamente relacionado con el crecimiento económico del país y par consecuencia con la llegada de empresas medianas y grandes, para las que los sistemas como el presentado en este trabajo, están pensados. Para entonces, más que una oferta, estos sistemas serán una verdadera necesidad para todas estas empresas.

Apéndice A. Diagramas de las Configuraciones de los Ambientes de Desarrollo y de Despliegue (Ejecución) del prototipo elaborado.

Ambiente de Desarrollo.



Ambiente de Despliegue (Ejecución).



Apéndice B. Código de la lógica del prototipo.

Programa para la Dirección de Recursos Humanos y Servicios Corporativos del Grupo ICA.

```
-- START METHOD DEFINITIONS

------------------------------------------------------------
method ICASqlManager.Init
begin
super.Init;
end method;

------------------------------------------------------------
method ICASqlManager.ConnectDb: GenericDBMS.DBSession
begin
--
-- This method connects to the database.
rdbsession : dbsession ;
rdbsession = ICADBMgr.ConnectDB('rechuman.sirh','demo','demoniaco');
return rdbsession;
end method;

------------------------------------------------------------
method ICASqlManager.RetrieveData(input Mysession: GenericDBMS.DBSession,
        input CommandString: Framework.TextData,
        input NumColumns: integer): copy ICADEMO.StoredData
begin
// This method retrieves the column names and data dynamically from the
// database and stores them in their respective arrays. The object which
// contains these arrays is returned.
dynStatement      : DBStatementHandle;// Statement from Prepare
inputDescriptor   : DBDataSet;      // Description of input
outputDescriptor  : DBDataSet;      // Description of output
outputData        : DBDataSet;      // Actual output
rowType : integer;
StatementType : integer;
DataFetched : StoredData = new;

   dynStatement = MySession.Prepare(
        commandString = CommandString,    // command
        inputDataSet = inputDescriptor,   // output of substitutions
        cmdType = statementType);         // output of statement type

   rowType = MySession.OpenCursor(
        statementHandle = dynStatement,   // The statement
        resultDataSet = outputDescriptor); // The description of output

// Load the ColumnsArray with the column names returned in the
// outputdescriptor.
// These column names will be used to create the field labels when
// the data is displayed.
        for j in 1 to NumColumns do
```

```
                ColumnName : TextData;
                ColumnInfo : DBColumnDesc;
                ColumnInfo = OutputDescriptor.GetColumn(position = j);
                ColumnName = ColumnInfo.Name.Clone(TRUE);
                DataFetched.ColumnsArray.AppendRow(ColumnName);
        end for;

    numrows : integer;
    totalRows : integer = 0;
    tmp : textdata = new;

// Fetch each row from the result set
    while TRUE do
        TotalRows = TotalRows + 1;
        DataFetched.ResultSetArray[TotalRows] = new;
        numrows = MySession.FetchCursor(
          statementHandle = dynStatement,
          resultDataSet = outputData);
          if numrows <= 0 then
              exit;
          end if;
// Store the data retrieved in the DataArray.
          for j in 1 to NumColumns do
              ColumnValue: TextNullable = new;
              ColumnValue.SetValue(outputData.GetValue(position = j));
              DataFetched.ResultSetArray[TotalRows].DataArray.AppendRow(ColumnValue);
          end for;
    end while;
    return DataFetched;

exception

// If any type of DB error occurs then raise the exception to the calling(client) method
// so that it can handle it.  This exception block will not handle problems like the
// database going down.
        when e:DBResourceException do
            raise;


end method;

-----------------------------------------------------------
method ICASqlManager.Gettotal(input querystring: Framework.TextData,
        input mysession: GenericDBMS.DBSession): ICADEMO.TotalObject
begin
dynStatement      : DBStatementHandle;// Statement from Prepare
inputDescriptor   : DBDataSet;      // Description of input
outputDescriptor  : DBDataSet;      // Description of output
outputData        : DBDataSet;      // Actual output output
rowType : integer;
StatementType : integer;
Total1 : TotalObject = new;
 dynStatement = MySession.Prepare(
        commandString = QueryString,    // command
        inputDataSet = inputDescriptor,   // output of substitutions
```

```
        cmdType = statementType);
          rowType = MySession.OpenCursor(
        statementHandle = dynStatement,    // The statement
        resultDataSet = outputDescriptor);  // The description of output

  numrows : integer;

        numrows = MySession.FetchCursor(
           statementHandle = dynStatement,
           resultDataSet = outputdata);


// Store the data retrieved in the DataArray.
          for i in 1 to 6 do
           avgvalue : integernullable = new;

           avgvalue.SetValue(outputdata.GetValue(position = i));
           if (avgvalue.IsNull ) then
            avgvalue.SetValue('0');
            else
                 avgvalue.SetValue(outputdata.GetValue(position = i));
           end if;
           total1.totals[i] = avgvalue;
   end for;
   return Total1;
exception

// if any type of DB error occurs then raise the exception to the calling(client) method
// so that it can handle it.  This exception block will not handle problems like the
// database going down.
        when e:DBResourceException do
                raise;

end method;

-----------------------------------------------------------
method ICASqlManager.test1
begin
Mysession:DBSession;
MySession = ConnectDb();
temparr : array of icadataobject = new;
kount : integer;
sql select
ANT_PLAN,ANTIG,CVE_INST,EDAD,EDO_CIVIL,EMPRESA,ESCOLAR,ESTATUS,EVAL1,EV
AL2,EVAL3,F_ING_EMP,F_NACI,F_PLAN,F_PUESTO,NOMBRE,OBRA,PLAN,PUESTO,RFC,
SOCIO,SUELDO,UN
   into :temparr
        from per_todoica
        on session Mysession;
sql select count(*) into :kount
   from per_todoica
   on session Mysession;

task.part.logmgr.putline(kount);
for j in 1 to temparr.items do
```

```
  task.part.logmgr.putline(temparr[j].ANT_PLAN);
  task.part.logmgr.putline(temparr[j].ANTIG);
  task.part.logmgr.putline(temparr[j].CVE_INST);
  task.part.logmgr.putline(temparr[j].EDAD);
  task.part.logmgr.putline(temparr[j].EDO_CIVIL);
  task.part.logmgr.putline(temparr[j].EMPRESA);
  task.part.logmgr.putline(temparr[j].ESCOLAR);
  task.part.logmgr.putline(temparr[j].ESTATUS);
  task.part.logmgr.putline(temparr[j].EVAL1);
  task.part.logmgr.putline(temparr[j].EVAL2);
  task.part.logmgr.putline(temparr[j].EVAL3);
  task.part.logmgr.putline(temparr[j].F_ING_EMP);
  task.part.logmgr.putline(temparr[j].F_NACI);
  task.part.logmgr.putline(temparr[j].F_PLAN);
  task.part.logmgr.putline(temparr[j].F_PUESTO);
  task.part.logmgr.putline(temparr[j].NOMBRE);
  task.part.logmgr.putline(temparr[j].OBRA);
  task.part.logmgr.putline(temparr[j].PLAN);

end for;

end method;

-----------------------------------------------------------
method ICADataObject.Init
begin
super.init;
ANT_PLAN = new;
ANTIG = new;
CVE_INST = new;
EDAD = new;
EDO_CIVIL = new;
EMPRESA = new;
ESCOLAR = new;
ESTATUS = new;
EVAL1 = new;
EVAL2 = new;
EVAL3 = new;
EVAL3 = new;
F_ING_EMP = new;
F_NACI = new;
F_PLAN = new;
F_PUESTO = new;
NOMBRE = new;
OBRA = new;
PLAN = new;
PUESTO = new;
RFC = new;
SEXO = new;
SOCIO = new;
SUELDO = new;
UN = new;

ANT_PLAN.SetValue('');
```

```
ANTIG.SetValue('');
CVE_INST.SetValue('');
EDAD.SetValue('');
EDO_CIVIL.SetValue('');
EMPRESA.SetValue('');
ESCOLAR.SetValue('');
ESTATUS.SetValue('');
EVAL1.SetValue('');
EVAL2.SetValue('');
EVAL3.SetValue('');
EVAL3.SetValue('');
F_ING_EMP.SetValue('');
F_NACI.SetValue('');
F_PLAN.SetValue('');
F_PUESTO.SetValue('');
NOMBRE.SetValue('');
OBRA.SetValue('');
PLAN.SetValue('');
PUESTO.SetValue('');
RFC.SetValue('');
SEXO.SetValue('');
SOCIO.SetValue('');
SUELDO.SetValue('');
UN.SetValue('');

end method;

------------------------------------------------------------
method Criteria.Init
begin
super.Init;
Column = new;
ColValue = new;

end method;

------------------------------------------------------------
method DataRecords.Init
begin
super.Init;
DataArray = new;
end method;

------------------------------------------------------------
method StoredData.Init
begin
super.Init;
ColumnsArray = new;
ResultSetArray = new;
end method;

------------------------------------------------------------
method TotalObject.Init
begin
super.Init;
```

```
totals = new;
end method;

------------------------------------------------------------
method ImageList.Init
begin
super.Init;
end method;

------------------------------------------------------------
method ICA_DataWin.Init
begin
super.Init;
pict1 = new;
ANT_PLAN = new;
ANT_PLAN.TextValue.Clear();
ANTIG = new;
ANTIG.TextValue.Clear();
CVE_INST = new;
CVE_INST.TextValue.Clear();
EDAD = new;
EDAD.TextValue.Clear();
EDO_CIVIL = new;
EDO_CIVIL.TextValue.Clear();
EMPRESA = new;
EMPRESA.TextValue.Clear();
ESCOLAR = new;
ESCOLAR.TextValue.Clear();
ESTATUS = new;
ESTATUS.TextValue.Clear();
EVAL1 = new;
EVAL1.TextValue.Clear();
EVAL2 = new;
EVAL2.TextValue.Clear();
EVAL3 = new;
EVAL3.TextValue.Clear();
F_ING = new;
F_ING.TextValue.Clear();
F_NACI = new;
F_NACI.TextValue.Clear();
F_PLAN = new;
F_PLAN.TextValue.Clear();
CVE_PTO= new;
CVE_PTO.TextValue.Clear();
NOMBRE = new;
NOMBRE.TExtValue.Clear();
OBRA = new;
OBRA.TextValue.Clear();
PLAN = new;
PLAN.TextValue.Clear();
PUESTO = new;
PUESTO.TextValue.Clear();
RFC = new;
RFC.TextValue.Clear();
SEXO = new;
```

```
SEXO.TextValue.Clear();
SOCIO = new;
SOCIO.TextValue.Clear();
SUELDO = new;
SUELDO.TextValue.Clear();
UN = new;
UN.TextValue.Clear();

ANT_PLAN.SetValue('');
ANT_PLAN.IsNull = TRUE;
ANTIG.SetValue('');
ANTIG.IsNull = TRUE;
CVE_INST.SetValue('');
CVE_INST.IsNull = TRUE;
EDAD.SetValue('');
EDAD.IsNull = TRUE;
EDO_CIVIL.SetValue('');
EDO_CIVIL.IsNull = TRUE;
EMPRESA.SetValue('');
EMPRESA.IsNull = TRUE;
ESCOLAR.SetValue('');
ESCOLAR.IsNull = TRUE;
ESTATUS.SetValue('');
ESTATUS.IsNull = TRUE;
EVAL1.SetValue('');
EVAL1.IsNull = TRUE;
EVAL2.SetValue('');
EVAL2.IsNull = TRUE;
EVAL3.SetValue('');
EVAL3.IsNull = TRUE;
F_ING.SetValue('');
F_ING.IsNull = TRUE;
F_NACI.SetValue('');
F_NACI.IsNull = TRUE;
F_PLAN.SetValue('');
F_PLAN.IsNull = TRUE;
CVE_PTO.SetValue('');
CVE_PTO.IsNull = TRUE;
NOMBRE.SetValue('');
NOMBRE.IsNull = TRUE;
OBRA.SetValue('');
OBRA.IsNull = TRUE;
PLAN.SetValue('');
PLAN.IsNull = TRUE;
PUESTO.SetValue('');
PUESTO.IsNull = TRUE;
RFC.SetValue('');
RFC.IsNull = TRUE;
SEXO.SetValue('');
SEXO.IsNull = TRUE;
SOCIO.SetValue('');
SOCIO.IsNull = TRUE;
SUELDO.SetValue('');
SUELDO.IsNull = TRUE;
UN.SetValue('');
```

```
UN.IsNull = TRUE;

//<F_NACI>.DateTemplate = 'dd-mmm-yy';
//<F_PLAN>.DateTemplate = 'dd-mmm-yy';
//<F_PUESTO>.DateTemplate = 'dd-mmm-yy';
//<F_ING_EMP>.DateTemplate = 'dd-mmm-yy';
end method;

------------------------------------------------------------
method ICA_DataWin.Display
begin
// Copyright 1996, Digital Equipment de Mexico S.A. de C.V.

// Se instancia una sesion con la base de datos
Mysession : DBsession;
// Se llama al servicio para realizar la conexion
Mysession = DoSql.ConnectDb();
first_time : boolean = FALSE;
open;
// Deshabilita botones al inicio de la consulta
<Query_button>.state = FS_DISABLED;
<total_button>.state = FS_DISABLED;
<PrevRec_button>.state = FS_DISABLED;
<NextRec_button>.state = FS_DISABLED;
DataFetched : StoredData = new;
got_image_first_time : boolean = FALSE;
ImageFile : String;
ImageArray : Array of ImageList ;
ImageArray = AddToImagesList();
ImageIndex : Integer = 1;
InfoEven : DisplayEvent = new;
TeclaFuncion : Integer;

event loop
        when task.shutdown do
                exit;

        // Si se registra algun cambio en algun campo habilita botones
        when <container>.ChildAfterValueChange do
          if ( first_time = FALSE) then
            <Query_button>.state = FS_ENABLED;
            <PrevRec_button>.state = FS_ENABLED;
                  <NextRec_button>.state = FS_ENABLED;
             first_time = TRUE;
           end if;

        // Limpia los campos de la pantalla
        when <clear_button>.click do
                if ( first_time = TRUE) then
                  <Query_button>.state = FS_DISABLED;
                  first_time = FALSE;
                  end if;
                  clearall();
                  <container>.State = FS_UPDATE;
                  got_image_first_time = FALSE;
```

```
        //      Ejecuta el query
        when <Query_button>.Click do
                <container>.State = FS_VIEWONLY;
                <Query_button>.state = FS_DISABLED;
                CommandString = BuildQuery();          // Se contruye el query
          task.part.logmgr.putline(commandstring);  // Envia el comando construido a la consola
de Forte
             if (CommandString <>nil) then
                        begin
                                NumColumns = 24;

                           // Obtiene el resultado del query
                                DataFetched = DoSql.RetrieveData(Mysession,CommandString,
                                        NumColumns);
                                DisplayedRowCount  = 1;

                                exception
                                        when e:DBResourceException do
                                        task.ErrorMgr.ShowErrors(TRUE);
                                        when e:DBUsageException do
                                        task.ErrorMgr.ShowErrors(TRUE);
                                        when e:DBValueException do
                                        task.ErrorMgr.ShowErrors(TRUE);
                        end;
                     if
(DataFetched.ResultSetArray[DisplayedRowCount].DataArray[DisplayedRowCount] = NIL) then
                                window.MessageDialog('No se encontraron registros para la
condicion dada. ');
                        else
                                // Despliega resultados del query
                  DisplayDataRow(dataFetched.ResultSetArray[DisplayedRowCount].DataArray,
                                        dataFetched.ColumnsArray);
                                                <total_button>.state = FS_ENABLED; // Habilita
la opcion de totales
                           // Obtiene una imagen para el registro desplegado
             if got_image_first_time = FALSE then
               ImageIndex = 1;
               got_image_first_time = TRUE;
               ImageFile = ImageArray[ImageIndex].ImageName;
               if ImageIndex = ImageArray.Items then
                 pict1 = GetImage(Imagefile);
               else
                 ImageIndex = ImageIndex + 1;
                 pict1 = GetImage(Imagefile);
               end if;
             end if;

                     end if;
                    end if;

        // Despliega el registro anterior
        when <Prevrec_button>.click do
        // Despliega los campos del registro si actualmente se esta al menos en
        // el segundo registro
```

```
        if DisplayedRowCount >= 2 then
                DisplayedRowCount = DisplayedRowCount - 1;

DisplayDataRow(dataFetched.ResultSetArray[DisplayedRowCount].DataArray,
                        dataFetched.ColumnsArray);
        // Despliega la imagen
          if got_image_first_time = TRUE then
    ImageFile = ImageArray[ImageIndex].ImageName;
    If ImageIndex = ImageArray.Items then
      ImageIndex = 1;
      pict1 = GetImage(Imagefile);
    else
      ImageIndex = ImageIndex + 1;
      pict1 = GetImage(Imagefile);
    end if;
          end if;
        else
                window.MessageDialog('No existen mas registros, este es el primero...');
        end if;

// Despliega el registro siguiente si es que este existe
when <NextRec_button>.click do
// Despliega el siguiente registro
        if DisplayedRowCount < dataFetched.ResultSetArray.Items - 1 then
                DisplayedRowCount = DisplayedRowCount + 1;

DisplayDataRow(dataFetched.ResultSetArray[DisplayedRowCount].DataArray,
                        dataFetched.ColumnsArray);
// Trae la Imagen
                if got_image_first_time = TRUE then

    ImageFile = ImageArray[ImageIndex].ImageName;

    if ImageIndex = ImageArray.Items then
      ImageIndex = 1;
      pict1 = GetImage(Imagefile);
    else
      ImageIndex = ImageIndex + 1;
      pict1 = GetImage(Imagefile);
    end if;
                end if;
        else
                window.MessageDialog('No existen mas registros...');
        end if;

// Despliega la ventana Acerca de...
when <MenAbout>.activate do
        About:About_window=new;
        About.display;

// Desconecta la sesion establecida con la base de datos y sale a
// la ventana anterior
when <exit_button>.click do
        mysession.disconnect();
        exit;
```

```
        // Lo mismo que la opcion anterior pero desde el menu
        when <MenExit>.activate do
        mysession.disconnect();
                exit;

        // Despliega la ventana de totales
        when <total_button>.click do
          querywin:Totales_screen = new;
          querywin.display(WhereClause,Mysession);

        // Se definen teclas de funcion para realizar tareas que
        // pueden ser invocadas desde un boton
        // Con F10 Totales
        // Con F1  Limpiar pantalla
        // Con F12 Salir
        when Window.FunctionKeyPress(InfoEven,TeclaFuncion) do
        if TeclaFuncion = FN_F10 then
                querywin:Totales_screen = new;
                querywin.display(WhereClause,Mysession);
        elseif TeclaFuncion = FN_F1 then
                if ( first_time = TRUE) then
                  <Query_button>.state = FS_DISABLED;
                  first_time = FALSE;
                end if;
                clearall();
                <container>.State = FS_UPDATE;
                got_image_first_time = FALSE;
        elseif TeclaFuncion = FN_F12 then
                mysession.disconnect();
                exit;
        end if;

end event;
close;
end method;

-----------------------------------------------------------------
method ICA_DataWin.BuildQuery: Framework.TextData
begin
// This method builds the dynamic 'select' statement and returns it.
// It is built based on what the user specified as  the selection
// criteria

Commandstring = 'select ';

CommandString .Concat ( '
ANT_PLAN,ANTIG,CVE_INST,EDAD,EDO_CIVIL,EMPRESA,ESCOLAR,ESTATUS,EVAL1,EV
AL2,EVAL3,F_ING,F_NACI,F_PLAN,CVE_PTO,NOMBRE,OBRA,PLAN,PUESTO,RFC,SEXO,S
OCIO,SUELDO,UN' );

// Add the table name.
Commandstring.concat(' from PER_TODOICA');

CriteriaArray : Array of Criteria  = New;
```

```
Row_Index : Integer=1;


if Not (ANT_PLAN.IsNull) then
        CriteriaArray[Row_Index] = new;
        CriteriaArray[Row_Index].Column.SetValue( 'ANT_PLAN' );
        CriteriaArray[Row_Index].ColValue.SetValue(ANT_PLAN.Value);
        Row_Index = Row_Index + 1;
end If;

if  Not ( ANTIG.IsNull) then
        CriteriaArray[Row_Index] = new;
        CriteriaArray[Row_Index].Column.SetValue( 'ANTIG' );
        CriteriaArray[Row_Index].ColValue.SetValue(ANTIG.Value);
        Row_Index = Row_Index + 1;
end if;

if  Not ( CVE_INST.IsNull) then
        CriteriaArray[Row_Index] = new;
        CriteriaArray[Row_Index].Column.SetValue( 'CVE_INST' );
        CriteriaArray[Row_Index].ColValue.SetValue(CVE_INST.Value);
        Row_Index = Row_Index + 1;
end if;

if  Not ( EDAD.IsNull) then
        CriteriaArray[Row_Index] = new;
        CriteriaArray[Row_Index].Column.SetValue( 'EDAD' );
        CriteriaArray[Row_Index].ColValue.SetValue(EDAD.Value);
        Row_Index = Row_Index + 1;
end if;

if Not (EDO_CIVIL.IsNull) then
        CriteriaArray[Row_Index] = new;
        CriteriaArray[Row_Index].Column.SetValue( 'EDO_CIVIL' );
        CriteriaArray[Row_Index].ColValue.SetValue(EDO_CIVIL.Value);
        Row_Index = Row_Index + 1;
end if;

if Not (EMPRESA.IsNull) then
        CriteriaArray[Row_Index] = new;
        CriteriaArray[Row_Index].Column.SetValue( 'EMPRESA' );
        CriteriaArray[Row_Index].ColValue.SetValue(EMPRESA.Value);
        Row_Index = Row_Index + 1;
end if;

if Not (ESCOLAR.IsNull) then
        CriteriaArray[Row_Index] = new;
        CriteriaArray[Row_Index].Column.SetValue( 'ESCOLAR' );
        CriteriaArray[Row_Index].ColValue.SetValue(ESCOLAR.Value);
        Row_Index = Row_Index + 1;
end if;

if Not (ESTATUS.IsNull) then
        CriteriaArray[Row_Index] = new;
        CriteriaArray[Row_Index].Column.SetValue( 'ESTATUS' );
```

```
Row_Index : Integer=1;


if Not (ANT_PLAN.IsNull) then
        CriteriaArray[Row_Index] = new;
        CriteriaArray[Row_Index].Column.SetValue( 'ANT_PLAN' );
        CriteriaArray[Row_Index].ColValue.SetValue(ANT_PLAN.Value);
        Row_Index = Row_Index + 1;
end if;

if  Not ( ANTIG.IsNull) then
        CriteriaArray[Row_Index] = new;
        CriteriaArray[Row_Index].Column.SetValue( 'ANTIG' );
        CriteriaArray[Row_Index].ColValue.SetValue(ANTIG.Value);
        Row_Index = Row_Index + 1;
end if;

if  Not ( CVE_INST.IsNull) then
        CriteriaArray[Row_Index] = new;
        CriteriaArray[Row_Index].Column.SetValue( 'CVE_INST' );
        CriteriaArray[Row_Index].ColValue.SetValue(CVE_INST.Value);
        Row_Index = Row_Index + 1;
end if;

if  Not ( EDAD.IsNull) then
        CriteriaArray[Row_Index] = new;
        CriteriaArray[Row_Index].Column.SetValue( 'EDAD' );
        CriteriaArray[Row_Index].ColValue.SetValue(EDAD.Value);
        Row_Index = Row_Index + 1;
end if;

if Not (EDO_CIVIL.IsNull) then
        CriteriaArray[Row_Index] = new;
        CriteriaArray[Row_Index].Column.SetValue( 'EDO_CIVIL' );
        CriteriaArray[Row_Index].ColValue.SetValue(EDO_CIVIL.Value);
        Row_Index = Row_Index + 1;
end if;

if Not (EMPRESA.IsNull) then
        CriteriaArray[Row_Index] = new;
        CriteriaArray[Row_Index].Column.SetValue( 'EMPRESA' );
        CriteriaArray[Row_Index].ColValue.SetValue(EMPRESA.Value);
        Row_Index = Row_Index + 1;
end if;

if Not (ESCOLAR.IsNull) then
        CriteriaArray[Row_Index] = new;
        CriteriaArray[Row_Index].Column.SetValue( 'ESCOLAR' );
        CriteriaArray[Row_Index].ColValue.SetValue(ESCOLAR.Value);
        Row_Index = Row_Index + 1;
end if;

if Not (ESTATUS.IsNull) then
        CriteriaArray[Row_Index] = new;
        CriteriaArray[Row_Index].Column.SetValue( 'ESTATUS' );
```

```
        CriteriaArray[Row_Index].ColValue.SetValue(ESTATUS.Value);
        Row_Index = Row_Index + 1;
end if;

if Not (EVAL1.IsNull) then
        CriteriaArray[Row_Index] = new;
        CriteriaArray[Row_Index].Column.SetValue( 'EVAL1' );
        CriteriaArray[Row_Index].ColValue.SetValue(EVAL1.Value);
        Row_Index = Row_Index + 1;
end if;

if Not (EVAL2.IsNull) then
        CriteriaArray[Row_Index] = new;
        CriteriaArray[Row_Index].Column.SetValue( 'EVAL2' );
        CriteriaArray[Row_Index].ColValue.SetValue(EVAL2.Value);
        Row_Index = Row_Index + 1;
end if;

if Not (EVAL3.IsNull) then
        CriteriaArray[Row_Index] = new;
        CriteriaArray[Row_Index].Column.SetValue( 'EVAL3' );
        CriteriaArray[Row_Index].ColValue.SetValue(EVAL3.Value);
        Row_Index = Row_Index + 1;
end if;

if Not (F_ING.IsNull) then
        CriteriaArray[Row_Index] = new;
        CriteriaArray[Row_Index].Column.SetValue( 'F_ING' );
        CriteriaArray[Row_Index].ColValue.SetValue(F_ING.Value);
        Row_Index = Row_Index + 1;
end if;

if Not (F_NACI.IsNull) then
        CriteriaArray[Row_Index] = new;
        CriteriaArray[Row_Index].Column.SetValue( 'F_NACI' );
        CriteriaArray[Row_Index].ColValue.SetValue(F_NACI.Value);
        Row_Index = Row_Index + 1;
end if;

if Not (F_PLAN.IsNull) then
        CriteriaArray[Row_Index] = new;
        CriteriaArray[Row_Index].Column.SetValue( 'F_PLAN' );
        CriteriaArray[Row_Index].ColValue.SetValue(F_PLAN.Value);
        Row_Index = Row_Index + 1;
end if;

if Not (CVE_PTO.IsNull) then
        CriteriaArray[Row_Index] = new;
        CriteriaArray[Row_Index].Column.SetValue( 'CVE_PTO' );
        CriteriaArray[Row_Index].ColValue.SetValue(CVE_PTO.Value);
        Row_Index = Row_Index + 1;
end if;

if Not (NOMBRE.IsNull) then
        CriteriaArray[Row_Index] = new;
```

```
        CriteriaArray[Row_Index].Column.SetValue( 'NOMBRE' );
        CriteriaArray[Row_Index].ColValue.SetValue(NOMBRE.Value);
        Row_Index = Row_Index + 1;
end if;

if Not (OBRA.IsNull) then
        CriteriaArray[Row_Index] = new;
        CriteriaArray[Row_Index].Column.SetValue( 'OBRA' );
        CriteriaArray[Row_Index].ColValue.SetValue(OBRA.Value);
        Row_Index = Row_Index + 1;
end if;

if Not (PLAN.IsNull) then
        CriteriaArray[Row_Index] = new;
        CriteriaArray[Row_Index].Column.SetValue( 'PLAN' );
        CriteriaArray[Row_Index].ColValue.SetValue(PLAN.Value);
        Row_Index = Row_Index + 1;
end if;

if Not (PUESTO.IsNull) then
        CriteriaArray[Row_Index] = new;
        CriteriaArray[Row_Index].Column.SetValue( 'PUESTO' );
        CriteriaArray[Row_Index].ColValue.SetValue(PUESTO.Value);
        Row_Index = Row_Index + 1;
end if;

if Not (RFC.IsNull) then
        CriteriaArray[Row_Index] = new;
        CriteriaArray[Row_Index].Column.SetValue( 'RFC' );
        CriteriaArray[Row_Index].ColValue.SetValue(RFC.Value);
        Row_Index = Row_Index + 1;
end if;

if Not (SEXO.IsNull) then
        CriteriaArray[Row_Index] = new;
        CriteriaArray[Row_Index].Column.SetValue( 'SEXO' );
        SEXO1 : textdata = new;
        if (SEXO.IntegerValue = 1) then
          SEXO1.SetValue('M');
        else
                SEXO1.SetValue('F');
        end if;
        CriteriaArray[Row_Index].ColValue.SetValue(SEXO1.VAlue);
        Row_Index = Row_Index + 1;
end if;

if Not (SOCIO.IsNull) then
        CriteriaArray[Row_Index] = new;
        CriteriaArray[Row_Index].Column.SetValue( 'SOCIO' );
        CriteriaArray[Row_Index].ColValue.SetValue(SOCIO.Value);
        Row_Index = Row_Index + 1;
end if;

if Not (SUELDO.IsNull) then
        CriteriaArray[Row_Index] = new;
```

```
        CriteriaArray[Row_Index].Column.SetValue( 'SUELDO' );
        CriteriaArray[Row_Index].ColValue.SetValue(SUELDO.Value);
        Row_Index = Row_Index + 1;
end if;

if Not (UN.IsNull) then
        CriteriaArray[Row_Index] = new;
        CriteriaArray[Row_Index].Column.SetValue( 'UN' );
        CriteriaArray[Row_Index].ColValue.SetValue(UN.Value);
        Row_Index = Row_Index + 1;
end if;


if (CriteriaArray.Items > 0) then

// Start constructing the where clause but only if the user specified
// selection criteria.
HasWhere : Boolean = FALSE;
Columns : Integer = 0;

WhereClause = new;

WhereClause.concat(' where ');

// Cycle through the Criteria array to see if the user entered any values.
for i in 1 to CriteriaArray.Items do
                if (CriteriaArray[i].Column.Value = 'EDAD') or
                  (CriteriaArray[i].Column.Value = 'SUELDO') or
                  (CriteriaArray[i].Column.Value = 'ANTIG') or
                  (CriteriaArray[i].Column.Value = 'ANT_PLAN') then
                  auxil : TextData = new;
                  auxil.SetValue(CriteriaArray[i].ColValue.TrimLeading);
                  if (auxil.IsDigit() = TRUE) then
                                WhereClause.concat(CriteriaArray[i].Column).concat(' = ');
                                WhereClause.concat(CriteriaArray[i].ColValue);

                  else
                                WhereClause.Concat(CriteriaArray[i].Column).Concat(' ');
                                WhereClause.concat(CriteriaArray[i].ColValue);
                  end if;
                elseif (CriteriaArray[i].Column.Value = 'F_ING') or
                    (CriteriaArray[i].Column.Value = 'F_PLAN') or
                    (CriteriaArray[i].Column.Value = 'F_NACI') then
                  auxil2 : TextData = new;
                  auxil2.SetValue(CriteriaArray[i].ColValue.TrimLeading);
                  if (auxil2.IsDigit() = TRUE) then
                                WhereClause.concat(CriteriaArray[i].Column).concat(' = ');

        WhereClause.concat("'").concat(CriteriaArray[i].ColValue).concat("'");
                  else
                                offs : int = 1;
                                while offs < auxil2.ActualSize do
                                        if auxil2.IsDigit() = TRUE then
                                          exit;
                                        end if;
```

```
                                        offs = offs + 1;
                                        auxil2.Offset = offs;
                                end while;
                                auxil2.Offset = 0;
                                WhereClause.concat(CriteriaArray[i].Column).Concat(' ');
                                WhereClause.Concat(auxil2.CopyRange(0,offs)).Concat("");

        WhereClause.Concat(auxil2.CopyRange(offs,auxil2.ActualSize)).Concat("");

                end if;
        else
                        WhereClause.concat(CriteriaArray[i].Column).concat(' like ');
                        WhereClause.concat("").concat(CriteriaArray[i].ColValue).concat("");
                end if;

// If this is not the last row in the Criteria array then add the And/Or.
                if i < CriteriaArray.Items  then
                        WhereClause.concat(' and ');
                end if;
                HasWhere = TRUE;
end for;

// If we actually constructed a 'where' clause then add it to the query.
if HasWhere then
        CommandString.concat(WhereClause);

end if;


else
        Self.Window.MessageDialog(' Proporcione informacion en uno o mas
camposII',MT_INFO);
        <ANT_PLAN>.RequestFocus();
        CommandString = nil;

end if;
Return CommandString;
end method;

-----------------------------------------------------------------
method ICA_DataWin.DisplayDataRow(input DataArray: Framework.Array of
Framework.TextData,
        input ColArray: Framework.Array of Framework.TextData)
begin
/*
for i in 1 to dataArray.Items do
  task.part.logmgr.putline(dataarray[i].textvalue);
end for;
*/

for i in 1 to dataarray.Items do
        tempfldwidget : fieldwidget;
  tempfldwidget = <container>.GetFieldByName(colarray[i].textvalue);

 if ( colarray[i].textValue.IsEqual('SEXO') ) then
```

```
    if ( dataarray[i].textvalue.IsEqual('M')) then
       <SEXO>.IntegerValue = 1;
     else
      <SEXO>.IntegerValue =2;
    end if;

 else
 /*
        if ( colarray[i].textValue.Isequal('F_NACI') and Not(dataarray[i].textvalue.IsNull)) then
                <F_NACI>.datetemplate = 'dd-mmm-yy';
                 F_NACI.textValue = dataarray[i].textvalue;

  else

        if ( colarray[i].textValue.Isequal('F_PLAN')and Not(dataarray[i].textvalue.IsNull)) then
                <F_PLAN>.datetemplate = 'dd-mmm-yy';
                F_PLAN.textValue = dataarray[i].textvalue;

 else

        if ( colarray[i].textValue.Isequal('F_ING_EMP')and Not(dataarray[i].textvalue.IsNull))
then
                <F_ING_EMP>.datetemplate = 'dd-mmm-yy';
                F_ING_EMP.textValue = dataarray[i].textvalue;

 else
 */


  datafield(tempfldwidget).TextValue.SetValue(dataarray[i].textvalue);
 end if;

 //end if;
 // end if;
 //end if;
  end for;

end method;

-----------------------------------------------------------------
method ICA_DataWin.AddToImagesList: Framework.Array of ICADEMO.ImageList
begin
ImageListArray : Array of ImageList=NEW;

for i in 1 to 4 do
  ImageListArray[i] = new;
end for;
/*
ImageListArray[1].ImageName = 'MONA.BMP';
ImageListArray[2].ImageName = 'CEZANNE.BMP';
ImageListArray[3].ImageName = 'DEGAS.BMP';
ImageListArray[4].ImageName = 'KADINSKY.BMP';
ImageListArray[5].ImageName = 'DALI.BMP';
ImageListArray[6].ImageName = 'REMBRND.BMP';
ImageListArray[7].ImageName = 'GAUGUIN.BMP';
```

```
ImageListArray[8].ImageName = 'ROUSSEA.BMP';
ImageListArray[9].ImageName = 'RENOIR.BMP';
*/

ImageListArray[1].ImageName = 'M1.BMP';
ImageListArray[2].ImageName = 'M2.BMP';
ImageListArray[3].ImageName = 'M3.BMP';
ImageListArray[4].Imagename = 'W1.BMP';


return ImageListArray;
end method;

-----------------------------------------------------------------
method ICA_DataWin.GetImage(input ImageName: string): Framework.ImageData
begin
Img : ImageData = new;
filename : TextData = new;

filename.Setvalue(ImageName);

Img_file : File = new;
Img_file.SetLocalName('${FORTE_ROOT}');
Img_file.AddToPath('Install');
Img_file.AddToPath('examples');
Img_file.AddToPath('Images');
Img_file.AddToPath(filename);
Img.ReadFromFile(Img_file);
return Img;

end method;

-----------------------------------------------------------------
method ICA_DataWin.ClearAll
begin
                ANT_PLAN.TextValue.Clear();
                ANTIG.TextValue.Clear();
                CVE_INST.TextValue.Clear();
                EDAD.TextValue.Clear();
                EDO_CIVIL.TextValue.Clear();
                EMPRESA.TextValue.Clear();
                ESCOLAR.TextValue.Clear();
                ESTATUS.TextValue.Clear();
                EVAL1.TextValue.Clear();
                EVAL2.TextValue.Clear();
                EVAL3.TextValue.Clear();
                F_ING.TextValue.Clear();
                F_NACI.TextValue.Clear();
                F_PLAN.TextValue.Clear();
                CVE_PTO.TextValue.Clear();
                NOMBRE.TExtValue.Clear();
                OBRA.TextValue.Clear();
                PLAN.TextValue.Clear();
                PUESTO.TextValue.Clear();
                RFC.TextValue.Clear();
```

```
        SEXO.TextValue.Clear();
        SOCIO.TextValue.Clear();
        SUELDO.TextValue.Clear();
        UN.TextValue.Clear();
        ANT_PLAN.SetValue('');
        ANT_PLAN.IsNull = TRUE;
        ANTIG.SetValue('');
        ANTIG.IsNull = TRUE;
        CVE_INST.SetValue('');
        CVE_INST.IsNull = TRUE;
        EDAD.SetValue('');
        EDAD.IsNull = TRUE;
        EDO_CIVIL.SetValue('');
        EDO_CIVIL.IsNull = TRUE;
        EMPRESA.SetValue('');
        EMPRESA.IsNull = TRUE;
// PICT1.TEXTvALUE.cLEAR();
// PICT1 = new;
        pict1 = NIL;
        ESCOLAR.SetValue('');
        ESCOLAR.IsNull = TRUE;
        ESTATUS.SetValue('');
        ESTATUS.IsNull = TRUE;
        EVAL1.SetValue('');
        EVAL1.IsNull = TRUE;
        EVAL2.SetValue('');
        EVAL2.IsNull = TRUE;
        EVAL3.SetValue('');
        EVAL3.IsNull = TRUE;
        F_ING.SetValue('');
        F_ING.IsNull = TRUE;
        F_NACI.SetValue('');
        F_NACI.IsNull = TRUE;
        F_PLAN.SetValue('');
        F_PLAN.IsNull = TRUE;
        CVE_PTO.SetValue('');
        CVE_PTO.IsNull = TRUE;
        NOMBRE.SetValue('');
        NOMBRE.IsNull = TRUE;
        OBRA.SetValue('');
        OBRA.IsNull = TRUE;
        PLAN.SetValue('');
        PLAN.IsNull = TRUE;
        PUESTO.SetValue('');
        PUESTO.IsNull = TRUE;
        RFC.SetValue('');
        RFC.IsNull = TRUE;
        SEXO.SetValue('');
        SEXO.IsNull = TRUE;
        SOCIO.SetValue('');
        SOCIO.IsNull = TRUE;
        SUELDO.SetValue('');
        SUELDO.IsNull = TRUE;
        UN.SetValue('');
        UN.IsNull = TRUE;
```

```
end method;

-----------------------------------------------------------
method Totales_screen.Init
begin
super.Init;
Prom_Antig = new;
Prom_Antig_Plan = new;
Prom_Edad = new;
Prom_Sueldos = new;
Total_Personas = new;
Total_Sueldos = new;

end method;

-----------------------------------------------------------
method Totales_screen.Display(input WhereClause: Framework.TextData,
        input MySession: GenericDBMS.DBSession)
begin
// Copyright 1996, Digital Equipment de Mexico S.A. de C.V.

// Este metodo despliega la totalizacion del query hecho
// en la pantalla de consulta

// Se construye el query para obtener los datos ya totalizados
// directamente de la base de datos
QueryString : textdata = new;
QueryString.Concat ('SELECT  COUNT(*), SUM(SUELDO), Avg(EDAD), AVG(SUELDO),
AVG(ANTIG), AVG(ANT_PLAN) ');
QueryString.Concat(' from PER_TODOICA ');
QueryString.Concat(WhereClause);   // Se emplea la clausula where empleada en el query
TotalObj : TotalObject = new;
TotalObj = DoSql.Gettotal(QueryString,MySession);  // Se ejecuta el comando para obtener la
totalizacion

// Si un valor obtenido en la consulta es nulo se le pone un valor de 0
for i in 1 to totalobj.totals.items do
  if (TotalObj.totals[i].textvalue= NIL) then
    TotalObj.totals[i].textvalue = '0';
  end if;
end for;

// Pone los valores de la consultas en los campos correspondientes de despliegue
I : integer=1;
Total_Personae.textValue =TotalObj.totals[i].textvalue;
I = I + 1;
Total_Sueldos.textValue = TotalObj.totals[I].textvalue;
I = I+ 1;
Prom_Edad.textValue = TotalObj.totals[i].textvalue;
I = I + 1;
Prom_Sueldos.textValue = TotalObj.totals[I].textvalue;
I = I + 1;
Prom_Antig.textValue =TotalObj.totals[I].textvalue ;
I = I + 1;
Prom_Antig_Plan.textValue = TotalObj.totals[i].textvalue;
```

```
open;
event loop
        when task.shutdown do
                exit;

        when <return_button>.click do  // Salir
                exit;
end event;
close;
end method;

-----------------------------------------------------------
method Logon_win.Display
begin
// Copyright 1996, Digital Equipment de Mexico S.A. de C.V.

open;
event loop

        when task.shutdown do
                exit;

        when <cancel_button>.click do
                exit;

        when <accept_button>.click do
        // Verifica la clave de acceso al sistema de recursos humanos
                if username.value = 'sirh' and password.value = 'SIRH96' then
                ICA_DataWinDisplay:ICA_DataWin = new;
                ICA_DataWinDisplay.Display();

                else

                window.messagedialog ('Proporcione nuevamente la clave de
acceso...',MT_INFO);
                end if;

end event;
close;
end method;

-----------------------------------------------------------
method Logon_win.Init
begin
super.Init;
password = new;
username = new;
end method;

-----------------------------------------------------------
method About_window.Init
begin
```

```
super.Init;
end method;

-----------------------------------------------------------
method About_window.Display
begin
// Copyright 1996, Digital Equipment de Mexico S.A. de C.V.

open;
event loop
        when task.shutdown do
                exit;

        when <ok_button>.click do
                exit;

end event;
close;
end method;
-- END METHOD DEFINITIONS
HAS PROPERTY
        CompatibilityLevel = 0;
        ProjectType = APPLICATION;
        Restricted = FALSE;
        MultiThreaded = TRUE;
        LibraryName = 'icademo';
        StartingMethod = (class = Logon_win, method = Display);


end ICADEMO;
```

Programa para ICA Construcción Urbana.

```
-- START CURSOR DEFINITIONS
cursor Empleados
begin
  // Copyright 1996, Digital Equipment de Mexico S.A. de C.V.

// Declaracion del cursor
// Selecciona el rfc, base gravable sin aguinaldo exento,
// auxiliar para determinar el aguinaldo exento,
// factor 1, factor 2, salario minimo de la zona
select rfc11 "rfc",
     (nvl(d01,0)+nvl(d55,0)+nvl(o_pt,0)+nvl(o_ag,0)-nvl(d02,0)-nvl(o_pe,0))*1000 "bg",
     (nvl(d55,0)+nvl(o_ag,0))*1000 "aux_ae",
     (1-((1-nvl(factor1,0.8677))*2.0))*10000 "fac1",
     nvl(factor2,0.9533)*10000 "fac2",
     nvl(d10,0)*10000 "isr_ret_ac",
     smz*30*1000 "smz30"
  from compacta, zona_ec
 where zona = zec;

 end;
```

```
-- END CURSOR DEFINITIONS

-- START TYPEDEF DEFINITIONS
-- END TYPEDEF DEFINITIONS

-- START METHOD DEFINITIONS

------------------------------------------------------------
method Datos.Init
begin
super.Init();
rfc = new;
end method;

------------------------------------------------------------
method DatosCons.Init
begin
super.Init();
rfc11 = new;
nombre11 = new;
paterno = new;
materno = new;
end method;

------------------------------------------------------------
method DatosComp.Init
begin
super.Init();
nombre = new;
rfc11 = new;
end method;

------------------------------------------------------------
method GuardaRec.Init
begin
super.Init();
Informacion = new;
end method;

------------------------------------------------------------
method AuxDivide.Init
begin
super.Init();
end method;

------------------------------------------------------------
method AuxDivide.Divide(input output auxae: double,
        input output bg: double,
        input output fac1: double,
        input output fac2: double,
        input output isrret: double,
        input output smz30: double)
begin
auxae = auxae / 1000;
bg = bg / 1000;
```

```
fac1 = fac1 / 10000;
fac2 = fac2 / 10000;
isrret = isrret / 10000;
smz30 = smz30 / 1000;
end method;

-------------------------------------------------------------
method ImpuestWin.Init
begin
super.init;
obra = new;
obra.TextValue.Clear();
end method;

-------------------------------------------------------------
method ImpuestWin.Display
begin
// Copyright 1996, Digital Equipment de Mexico S.A. de C.V.

open;
// Se verifica que la maquina donde corra la particion sea PC
// de lo contrario se deshabilita la opcion de carga de informacion
// desde archivos .dbf
if task.Part.OperatingSystem.MachineArch <> OS_MA_PC then
 <ConsolidaDBF>.state = FS_DISABLED;
end if;
event loop
        when task.shutdown do
                exit;
        when <Salir>.Click do
                exit;

        // Carga de datos desde archivos ASCII
        when <Consolida>.Click do
                begin
                        linea_estado = 'Cargando información a consolida desde ascii...';
                        CargaAscii();   // Carga un archivo ascii a Oracle
                        obra.Clear();   // Limpia el campo obra de la pantalla
                exception
                        when e : DBResourceException do
                                task.ErrorMgr.ShowErrors(TRUE);
                        when e : AbortException do
                                task.ErrorMgr.ShowErrors(TRUE);
                end;

        // Coloca un letrero en la barra de estado
        when <ConsolidaDBF>.AfterFocusGain(FC_MOUSECLICK) do
                linea_estado = 'Cargando información a consolida desde archivos .dbf...';

        // Carga de datos desde archivos dbf
        when <ConsolidaDBF>.Click do
                begin
                        CargaDBF();   // Carga dbf a Oracle
                        obra.Clear();
                exception
```

```
                when e : DBResourceException do
                        task.ErrorMgr.ShowErrors(TRUE);
                when e : AbortException do
                        task.ErrorMgr.ShowErrors(TRUE);
        end;

// Coloca un letrero en la barra de estado
when <ConsolidaObra>.AfterFocusGain(FC_MOUSECLICK) do
        linea_estado = 'Cargando información a consolida desde tabla Obra...';

// Carga de datos desde la tabla OBRA en Oracle
when <ConsolidaObra>.Click do
        begin
                CargaObra(); // Carga de datos desde la tabla OBRA
                obra.Clear();
        exception
                when e : DBResourceException do
                        task.ErrorMgr.ShowErrors(TRUE);
                when e : AbortException do
                        task.ErrorMgr.ShowErrors(TRUE);
        end;

// Coloca un letrero en la barra de estado
when <Compacta>.AfterFocusGain(FC_MOUSECLICK) do
        linea_estado = 'Espere, se esta llevando a cabo la compactación...';

// Compacta la informacion de la tabla consolida y la deja en la tabla compacta
when <Compacta>.Click do
        begin
                CompactaInf(); // Compacta
                linea_estado = 'Compactación realizada';
        exception
                when e : DBResourceException do
                        task.ErrorMgr.ShowErrors(TRUE);
                when e : AbortException do
                        task.ErrorMgr.ShowErrors(TRUE);
        end;

// Coloca un letrero en la barra de estado
when <calcula>.AfterFocusGain() do
        linea_estado = 'Se esta actualizando la tabla compacta';

// Realizacion del calculo del ISPT
when <calcula>.Click do
        begin
                linea_estado = 'Se esta actualizando la tabla compacta';
                Calc_Ispt();  // Calculo del ISPT
                linea_estado = 'Finalizó el calculo del ISPT';
        exception
                when e : DBResourceException do
                        task.ErrorMgr.ShowErrors(TRUE);
                when e : AbortException do
                        task.ErrorMgr.ShowErrors(TRUE);
        end;
```

```
        // Se invoca una nueva pantalla para manejo de la informacion
        // de la tabla consolida
        when <ConsWin>.Click do
                consowin : ConsolidaWin = new;
                consowin.Display();

        // Se Invoca a una nueva pantalla para visualizar la informacion
        // resultante del calculo del ISPT
        when <CompWin>.Click do
                compacwin : Compacta = new;
                compacwin.Display();
end event;
close;
end method;

-------------------------------------------------------------
method ImpuestWin.CompactaInf
begin
// Copyright 1996, Digital Equipment de Mexico S.A. de C.V.

// Se elimina la informacion anterior de la tabla Compacta
// en Oracle

DefaultDBSession.ExecuteImmediate('delete compacta');

// Se crea el comando para insertar en la tabla Compacta
// los registros totalizados por rfc11 de la tabla
// consolida

comando : TextData=new;

comando.SetValue('insert into compacta ');
comando.Concat('select max(obra),max(paterno),max(materno),');
comando.Concat('max(nombre11),max(clave),max(alta),max(baja),');
comando.Concat('max(imss),max(puesto),max(sueldo),max(zona),max(factor1),');
comando.Concat('max(factor2),max(stat),sum(nvl(d01,0)),sum(nvl(d02,0)),');
comando.Concat('sum(nvl(d03,0)),sum(nvl(d04,0)),sum(nvl(d05,0)),');
comando.Concat('sum(nvl(d06,0)),sum(nvl(d07,0)),sum(nvl(d08,0)),');
comando.Concat('sum(nvl(d09,0)),sum(nvl(d10,0)),sum(nvl(d11,0)),');
comando.Concat('sum(nvl(d12,0)),sum(nvl(d13,0)),sum(nvl(d14,0)),');
comando.Concat('sum(nvl(d15,0)),sum(nvl(d16,0)),sum(nvl(d17,0)),');
comando.Concat('sum(nvl(d18,0)),sum(nvl(d19,0)),sum(nvl(d20,0)),');
comando.Concat('sum(nvl(d21,0)),sum(nvl(d22,0)),sum(nvl(d23,0)),');
comando.Concat('sum(nvl(d24,0)),sum(nvl(d25,0)),sum(nvl(d26,0)),');
comando.Concat('sum(nvl(d27,0)),sum(nvl(d28,0)),sum(nvl(d29,0)),');
comando.Concat('sum(nvl(d30,0)),sum(nvl(d31,0)),sum(nvl(d32,0)),');
comando.Concat('sum(nvl(d33,0)),sum(nvl(d34,0)),sum(nvl(d35,0)),');
comando.Concat('sum(nvl(d36,0)),sum(nvl(d37,0)),sum(nvl(d38,0)),');
comando.Concat('sum(nvl(d39,0)),sum(nvl(d40,0)),sum(nvl(d41,0)),');
comando.Concat('sum(nvl(d42,0)),sum(nvl(d43,0)),sum(nvl(d44,0)),');
comando.Concat('sum(nvl(d45,0)),sum(nvl(d46,0)),sum(nvl(d47,0)),');
comando.Concat('sum(nvl(d48,0)),sum(nvl(d49,0)),sum(nvl(d50,0)),');
comando.Concat('sum(nvl(d51,0)),sum(nvl(d52,0)),sum(nvl(d53,0)),');
comando.Concat('sum(nvl(d54,0)),sum(nvl(d55,0)),sum(nvl(d56,0)),');
comando.Concat('sum(nvl(d57,0)),sum(nvl(d58,0)),sum(nvl(d59,0)),');
```

```
comando.Concat('sum(nvl(d60,0)),sum(nvl(d61,0)),sum(nvl(d62,0)),');
comando.Concat('sum(nvl(d63,0)),sum(nvl(d64,0)),sum(nvl(d65,0)),');
comando.Concat('sum(nvl(d66,0)),sum(nvl(d67,0)),sum(nvl(d68,0)),');
comando.Concat('sum(nvl(d69,0)),sum(nvl(d70,0)),sum(nvl(d71,0)),');
comando.Concat('sum(nvl(d72,0)),max(fecha11),rfc11,max(codigo),');
comando.Concat('sum(nvl(isr_cau,0)),sum(nvl(sac_cau,0)),sum(nvl(sna_cau,0)),');
comando.Concat('sum(nvl(gra_exe,0)),sum(nvl(cred,0)),sum(nvl(o_pt,0)),');
comando.Concat('sum(nvl(o_pe,0)),sum(nvl(o_ag,0)),sum(nvl(o_isr,0)),');
comando.Concat('sum(nvl(o_sac,0)),sum(nvl(o_sna,0)),sum(nvl(op_dias,0)),');
comando.Concat('max(nvl(fac1, 0)),max(nvl(fac1,0)),max(rfc12),');
comando.Concat('sum(nvl(per_gra,0)),sum(nvl(tot_ing,0)) ');
comando.Concat('from consolida ');
comando.Concat('group by rfc11');

// Se ejecuta el comando

DefaultDBSession.ExecuteImmediate(source = comando.Value);
DefaultDBSession.ExecuteImmediate('commit');
return;
end method;

---------------------------------------------------------------
method ImpuestWin.CargaAscii
begin
// Copyright 1996, Digital Equipment de Mexico S.A. de C.V.

// Metodo para cargar en la tabla CONSOLIDA los datos de archivos ascii

begin
arch_aux : TextData = new;

// Los archivos deben estar en la ruta $FORTE_ROOT/install/examples/icaispt

arch_aux.Concat(source = obra.Value).Concat(source = '.TXT');
archivo_leer : File = new;
archivo_leer.SetLocalName('${FORTE_ROOT}');
archivo_leer.AddToPath('install');
archivo_leer.AddToPath('examples');
archivo_leer.AddToPath('icaispt');
archivo_leer.AddToPath(arch_aux);

archivo_leer.Open(accessMode = SP_AM_READ);

comando : TextData = new;
linea: TextData = new;
tipo : Int = 0;
num_registros : Int = 0; // Numero de registros cargados
// Debido a que hay al menos dos tipos de archivos ascii,
// se determina en forma rudimentaria cual de los dos es el que se esta
// accesando
archivo_leer.Readline(target = linea);
if linea.ActualSize = 1639 then
        tipo = 1;
else
        comas : Int = 0;
```

```
        while linea.IsAtEnd <> TRUE do
          if linea.IsChar(listOfChars = ','}=TRUE then
              comas=comas+1;
          end if;
              linea.MoveNext();
        end while;
    if comas >= 105 then
      tipo = 2;
    else
      tipo = 3;
    end if;
end if;
// Si el archivo es de uno de los dos formatos conocidos
if (tipo = 1) or (tipo = 2) then
// Se lee el archivo linea a linea
// cada linea es un registro que se inserta en la tabla
// si es que la informacion es correcta
        archivo_leer.Offset = 0;
begin transaction
        while archivo_leer.ReadLine(target = linea) >= 0 do
          begin
          num_registros = num_registros + 1;
              comando.SetValue('insert into consolida values (');
              if tipo = 1 then
                  aux : TextData = new;

        comando.Concat("").Concat(linea.CopyRange(startOffset=0,endOffset=5)).TrimTrailing()
.Concat("',');          // Obra*

        comando.Concat("").Concat(linea.CopyRange(startOffset=5,endOffset=45)).TrimTrailing
().Concat("',');        // Paterno*

        comando.Concat("").Concat(linea.CopyRange(startOffset=45,endOffset=85)).TrimTrailin
g().Concat("',');       // Materno*

        comando.Concat("").Concat(linea.CopyRange(startOffset=85,endOffset=125)).TrimTraili
ng().Concat("',');              // Nombre11*

        comando.Concat("").Concat(linea.CopyRange(startOffset=125,endOffset=127)).TrimTrai
ling().Concat("',');    // Clave*
              // En el caso que no haya dato para la ALTA se da valor NULL
                  aux.SetValue(linea.CopyRange(startOffset= 127, endOffset=
135).TrimLeading());
                  if aux.ActualSize <= 1 then
                      comando.Concat('NULL,');
                  else

        comando.Concat(linea.CopyRange(startOffset=127,endOffset=135).TrimLeading()).Con
cat(',');               // Alta
                  end if;
              // En el caso que no haya dato para la BAJA se da valor NULL
                  aux.SetValue(linea.CopyRange(startOffset= 135, endOffset=
143).TrimLeading());
                  if aux.ActualSize <= 1 then
                      comando.Concat('NULL,');
```

```
                    else

        comando.Concat(linea.CopyRange(startOffset=135,endOffset=143).TrimLeading()).Con
cat(',');           // Baja
                    end if;

        comando.Concat('"').Concat(linea.CopyRange(startOffset=143,endOffset=153)).TrimTrai
ling().Concat('",');    // IMSS*

        comando.Concat('"').Concat(linea.CopyRange(startOffset=153,endOffset=183)).TrimTrai
ling().Concat('",');    // Puesto*
                // El sueldo no puede ser nulo, en caso de que lo sea se le da valor de cero
                    aux.SetValue(linea.CopyRange(startOffset= 183, endOffset=
192).TrimLeading());
                    if aux.ActualSize <= 1 then
                        comando.Concat('0,');
                    else

        comando.Concat(linea.CopyRange(startOffset=183,endOffset=192).TrimLeading()).Con
cat(',');           // Sueldo
                    end if;

        comando.Concat('"').Concat(linea.CopyRange(startOffset=192,endOffset=193)).TrimTrai
ling().Concat('",');    // Zona*
                // El FACTOR1 no puede ser nulo, en caso de que lo sea se le da valor de cero
                    aux.SetValue(linea.CopyRange(startOffset= 193, endOffset=
201).TrimLeading());
                    if aux.ActualSize <= 1 then
                        comando.Concat('0,');
                    else

        comando.Concat(linea.CopyRange(startOffset=193,endOffset=201).TrimLeading()).Con
cat(',');           // Factor1
                    end if;
                // El FACTOR2 no puede ser nulo, en caso de que lo sea se le da valor de cero
                    aux.SetValue(linea.CopyRange(startOffset= 201, endOffset=
209).TrimLeading());
                    if aux.ActualSize <= 1 then
                        comando.Concat('0,');
                    else

        comando.Concat(linea.CopyRange(startOffset=201,endOffset=209).TrimLeading()).Con
cat(',');           // Factor2
                    end if;

        comando.Concat('"').Concat(linea.CopyRange(startOffset=209,endOffset=210)).TrimTrai
ling().Concat('",');    // Stat*
                    offs : Int = 210;
                    for I in 1 to 72 do
                        aux.SetValue(linea.CopyRange(startOffset= offs, endOffset=
offs + 16).TrimLeading());
                        if aux.ActualSize <= 1 then
                            comando.Concat('NULL,');
                        else
```

```
                                        comando.Concat(linea.CopyRange(startOffset= offs,
endOffset= offs + 16).TrimLeading()).Concat(',');// D01...D72
                                    end if;
                                    offs = offs + 16;
                                end for;
                        // En el caso que no haya dato para la FECHA11 se da valor NULL
                                aux.SetValue(linea.CopyRange(startOffset= 1362, endOffset=
1370).TrimLeading());
                                if aux.ActualSize <= 1 then
                                        comando.Concat('NULL,');
                                else

        comando.Concat(linea.CopyRange(startOffset=1362,endOffset=1370).TrimLeading()).C
oncat(',');         // Fecha11
                                end if;

        comando.Concat("").Concat(linea.CopyRange(startOffset=1370,endOffset=1383)).TrimT
railing().Concat('",');     // RFC11*

        comando.Concat("").Concat(linea.CopyRange(startOffset=1383,endOffset=1393)).TrimT
railing().Concat('",');     // Codigo*
// Puesto que se conoce que los siguientes campos son nulos, esta por demas el leerlos
                                comando.Concat('NULL,NULL,NULL,NULL,NULL,');     // ISR_CAU,
SAC_CAU, SNA_CAU, GRA_EXE, CRED
                                comando.Concat('NULL,NULL,NULL,NULL,NULL,');     // O_PT, O_PE,
O_AG, O_ISR, O_SAC
                                comando.Concat('NULL,NULL,NULL,NULL,');    // O_SNA, OP_DIAS,
FAC1, FAC2

        comando.Concat("").Concat(linea.CopyRange(startOffset=1593,endOffset=1606)).TrimT
railing().Concat('",');     // RFC12*
                                comando.Concat('NULL,NULL)');          // PER_GRA, TOT_ING
                        else
                                comando.Concat(source = linea.Value).Concat(')');
                        end if;
                        DefaultDBSession.ExecuteImmediate(source = comando.Value);
                        exception
                                when e : DBUsageException do
                                        Window.MessageDialog(messageText='Este registro no se
inserta. Presione More... en la ventana siguiente',
                                                messageType = MT_ERROR);
                                        task.ErrorMgr.ShowErrors(clearout=TRUE);
                                        num_registros = num_registros - 1;
                                        continue;
                                when e: DBValueException do
                                        Window.MessageDialog(messageText='Este registro no se
inserta. Presione More... en la ventana siguiente',
                                                messageType = MT_ERROR);
                                        task.ErrorMgr.ShowErrors(clearout=TRUE);
                                        num_registros = num_registros - 1;
                                        continue;
                end;
        end while;
        end transaction;
        DefaultDBSession.ExecuteImmediate('commit');
```

```
else
        Window.MessageDialog(messageText='El archivo no tiene formato conocido, favor de
verificarlo',
                        messageType = MT_WARNING);
        linea_estado = ' ';
end if;
archivo_leer.Close();
auxi : TextData = new;
auxi.SetValue('Carga finalizada - ');
auxi.Concat(num_registros).Concat(' registros insertados en Consolida -');
linea_estado = auxi.Value;
exception
        when e : FileResourceException do
        linea_estado=' ';
        Window.MessageDialog(messageText='No existe archivo ascii para la obra
proporcionada'),
        task.ErrorMgr.Clear();
end;
return;
end method;

-----------------------------------------------------------
method ImpuestWin.CargaObra
begin
// Copyright 1996, Digital Equipment de Mexico S.A. de C.V.

// Este metodo hace la insercion de registros a la tabla consolida
// desde la tabla obra en Oracle de acurdo a la obra que se proporcione

begin
        // Se lee la obra
        if obra.ActualSize = 0 then
                linea_estado=' ';
                Window.MessageDialog('Debe proporcionarse el nombre de la obra a
consolidar');
                return;
        end if;
        comando : TextData = new;
        cuenta : integer;

        // Se crea el comando SQL para realizar la tarea
        comando.SetValue('insert into consolida select * from obra where obra = '");
        comando.Concat(obra).Concat("");
        DefaultDBSession.ExecuteImmediate(comando);  // Se ejecuta el comando
        DefaultDBSession.ExecuteImmediate('commit');  // Commit
        sql select count(*) into :cuenta from obra    // Se cuenta la cantidad de registros
insertados
                where obra = :obra;
        comando.SetValue('Carga finalizada - ');
        comando.Concat(cuenta).Concat(' registros insertados en Consolida -');
  linea_estado=comando.Value;        // Se despliega la cantidad de registros insertados

  // manejo de excepciones
        exception
                when e : DBResourceException do
```

```
                task.ErrorMgr.ShowErrors(TRUE);

                when e : DBUsageException do
                        Window.MessageDialog(messageText='Registro no insertado. Para
detalles Presione More... en la sig. ventana',
                                        messageType = MT_ERROR);
                        task.ErrorMgr.ShowErrors(clearout=TRUE);

                when e: DBValueException do
                        Window.MessageDialog(messageText='Registro no insertado. Para
detalles Presione More... en la sig. ventana',
                                        messageType = MT_ERROR);
                        task.ErrorMgr.ShowErrors(clearout=TRUE);

end;
return;
end method;

-------------------------------------------------------------
method ImpuestWin.Calc_Ispt
begin
// Copyright 1996, Digital Equipment de Mexico S.A. de C.V.

// Este metodo realiza el calculo del ISPT de los trabajadores
// El calculo se hace en Oracle, en la tabla compacta

li : double;        // limite inferior 141
cuf : double;       // cuf 141
tas : double;       // tasa 141
sub : double;       // subsidio 141
fis : double;       // sub_lis 141
liop : double;      // limite inferior 141op
cufop : double;     // cuf 141op
tasop : double;     // tasa 141op
subop : double;     // subsidio 141op
fisop : double;     // sub_fis 141op
cas : double;               // credito al salario
isr_cal :  double;          // ISR de calculo
isr_cal2 : double;          // ISR de calculo opcional
sub_acre : double;          // subsidio acreditable
sub_acre2 : double;         // subsidio acreditable opcional
sub_nacre : double;         // subsidio no acreditable
sub_nacre2 : double;        // subsidio no acreditable opcional
isr_ret : double;           // ISR retenido de calculo
isr_ret2 : double;          // ISR retenido de calculo opcional
isr_foc : double;           // ISR a favor o en contra
aux : double;               // auxiliar en calculo
aux2 : double;              // auxiliar en calculo opcional

emp : Empleados;        // Instancia del cursor Empleados

ArrDatos : LargeArray of Datos = new;

sql open cursor emp;
```

```
sql fetch cursor emp into :ArrDatos;
sql close cursor emp;

begin transaction
for j in 1 to ArrDatos.items do
  begin

        Calcular.Divide(auxae=ArrDatos[j].aux_ae,bg=ArrDatos[j].bg,
                fac1=ArrDatos[j].fac1,fac2=ArrDatos[j].fac2,
                isrret=ArrDatos[j].isr_ret_ac,smz30=ArrDatos[j].smz30);
// Calculo del aguinaldo exento
        if ArrDatos[j].aux_ae > ArrDatos[j].smz30 then
                ArrDatos[j].bg = ArrDatos[j].bg - ArrDatos[j].smz30;
        else
                ArrDatos[j].bg = ArrDatos[j].bg - ArrDatos[j].aux_ae;
        end if;

// Se extrae informacion de las tablas de calculo del ISPT (regla 141)
sql select regla_141.lim_inf*1000, regla_141.isr_cuf*1000, regla_141.isr_tas*1000,
        sub_141.sub_sub*1000, sub_141.sub_fis*1000,
        regla_141op.lim_inf*1000, regla_141op.isr_cuf*1000, regla_141op.isr_tas*1000,
        sub_141op.sub_sub*1000, sub_141op.sub_fis*1000,
        credito*1000
            into :li, :cuf, :tas, :sub, :fis, :liop, :cufop, :tasop , :subop, :fisop, :cas
            from regla_141, sub_141, regla_141op, sub_141op, cred_sal
           where :ArrDatos[j].bg between regla_141.lim_inf and regla_141.lim_sup
            and :ArrDatos[j].bg between sub_141.lim_inf and sub_141.lim_sup
            and :ArrDatos[j].bg between regla_141op.lim_inf and regla_141op.lim_sup
            and :ArrDatos[j].bg between sub_141op.lim_inf and sub_141op.lim_sup
            and :ArrDatos[j].bg between cred_sal.lim_inf and cred_sal.lim_sup;
        li = li / 1000; cuf = cuf / 1000; tas = tas / 1000;
        sub = sub / 1000; fis = fis / 1000;
        liop = liop / 1000; cufop = cufop / 1000; tasop = tasop / 1000;
        subop = subop / 1000; fisop = fisop / 1000;
        cas = cas / 1000;

// Se hace el calculo para la ley 141

        aux= ArrDatos[j].bg;            // Base Gravable
        aux= aux - li;      // - Limite Inferior
        aux= aux * tas;     // * Tasa ISR
        sub_acre= aux;      // Se inicializa para calculo de subsidio acreditable
        aux= aux + cuf;     // + ISR Cuota Fija
        isr_cal= aux;       // ISR causado de calculo

    // Inicia calculo de subsidio acreditable
        sub_acre= sub_acre * fis;     // * Sub_fis
        sub_acre= sub_acre + sub;     // + Cuota fija subsidio
        sub_nacre= sub_acre;          // Se inicializa para calculo de subsidio no acreditable
        sub_acre= sub_acre * ArrDatos[j].fac1;    // * Factor de subsidio acreditable (factor1)
    // Termina calculo de subsidio acreditable

        aux= aux - sub_acre;                // - Subsidio acreditable
        aux= aux - cas;                     // - Credito al salario
        isr_ret= aux;                       // ISR retenido de calculo
```

```
        sub_nacre= sub_nacre - sub_acre;  // Se calcula el subsidio no acreditable

// Termina calculo para la 141

// Se hace el calculo para la 141 opcional

        aux2= ArrDatos[j].bg;                   // Base Gravable
        aux2= aux2 - liop;          // - Limite inferior
        aux2= aux2 * tasop;          // * Tasa ISR
        sub_acre2= aux2;            // Se inicializa para calculo de subsidio acreditable
        aux2= aux2 + cufop;         // + ISR Cuota Fija
        isr_cal2= aux2;             // ISR causado de calculo

    // Inicia calculo de subsidio acreditable
        sub_acre2= sub_acre2 * fisop;                   // * Sub_fis
        sub_acre2= sub_acre2 + (cufop * subop);  // + Cuota fija subsidio
        sub_nacre2= sub_acre2;                          // Se inicializa para calculo de subsidio no
acreditable
        sub_acre2= sub_acre2 * ArrDatos[j].fac2;                // * Factor de subsidio acreditable
(factor2)
    // Termina calculo de subsidio acreditable

        aux2= aux2 - sub_acre2;                 // - Subsidio acreditable
        aux2= aux2 - cas;                       // - Credito al salario
        isr_ret2= aux2;                         // ISR retenido de calculo opcional
        sub_nacre2= sub_nacre2 - sub_acre2;  // Se calcula el subsidio no acreditable

// Termina calculo para la 141 opcional

// Se calcula ISR a favor o en contra

        if isr_ret <= isr_ret2 then
                isr_foc= isr_ret - ArrDatos[j].isr_ret_ac;
        else
                isr_foc= isr_ret2 - ArrDatos[j].isr_ret_ac;
        end if;

// Se actualizan los campos del registro con los valores calculados
        if isr_ret <= isr_ret2 then
                sub_acre = sub_acre*1000.0;
                sub_nacre = sub_nacre*1000.0;
                cas = cas*1000.0;
                isr_ret = isr_ret*1000.0;
                isr_cal = isr_cal*1000.0;
                isr_foc = isr_foc*1000.0;
                sql update compacta
                  set sac_cau = :sub_acre,
                sna_cau = :sub_nacre,
                cred = :cas,
                isr_cau = :isr_ret,
                per_gra = :isr_cal,
                tol_ing = :isr_foc
                  where rfc11 = :ArrDatos[j].rfc;
        else
                sub_acre2 = sub_acre2*1000.0;
```

```
                sub_nacre2 = sub_nacre2*1000.0;
                cas = cas*1000.0;
                isr_ret2 = isr_ret2*1000.0;
                isr_cal2 = isr_cal2*1000.0;
                isr_foc = isr_foc*1000.0;
                sql update compacta
                  set sac_cau = :sub_acre2,
                sna_cau = :sub_nacre2,
                cred = :cas,
                isr_cau = :isr_ret2,
                per_gra = :isr_cal2,
                tot_ing = :isr_foc
                  where rfc11 = :ArrDatos[j].rfc;
        end if;
        exception
                when e : DBUsageException do
                        Window.MessageDialog(messageText='Se presento un error. Presione
More... en la ventana siguiente',
                                                messageType = MT_ERROR);
                        task.ErrorMgr.ShowErrors(clearout=TRUE);
                        continue;
                when e: DBValueException do
                        Window.MessageDialog(messageText='Se presento un error. Presione
More... en la ventana siguiente',
                                                messageType = MT_ERROR);
                        task.ErrorMgr.ShowErrors(clearout=TRUE);
                        continue;
  end;
end for;
end transaction;
sql update compacta
    set sac_cau = sac_cau/1000,
            sna_cau = sna_cau/1000,
            cred = cred/1000,
            isr_cau = isr_cau/1000,
            per_gra = per_gra/1000,
            tot_ing = tot_ing/1000;
return;
end method;

-----------------------------------------------------------
method ImpuestWin.CargaDBF
begin
// Copyright 1996, Digital Equipment de Mexico S.A. de C.V.

// Metodo para cargar en la tabla CONSOLIDA los datos de archivos .dbf

begin
arch_aux : TextData = new;
archivo_leer : File = new;
num_registros : int = 0; // Numero de registros cargados

// Se verifica que se haya proporcionado la obra
if obra.ActualSize = 0 then
        linea_estado=' ';
```

```
        Window.MessageDialog('Debe proporcionarse el nombre de la obra a consolidar');
        return;
end if;

// Los archivos deben estar en la ruta
$FORTE_ROOT/install/examples/icalspt/nombre_de_la_obra
// Se checa la existencia de los 4 archivos dbf necesarios
arch_aux.SetValue('nomper.dbf');
archivo_leer.SetLocalName('${FORTE_ROOT}');
archivo_leer.AddToPath('install');
archivo_leer.AddToPath('examples');
archivo_leer.AddToPath('icalspt');
archivo_leer.AddToPath(obra);
archivo_leer.AddToPath(arch_aux);
archivo_leer.Open(accessMode = SP_AM_READ);
archivo_leer.Close();

arch_aux.SetValue('nomdge.dbf');
archivo_leer.SetLocalName('${FORTE_ROOT}');
archivo_leer.AddToPath('install');
archivo_leer.AddToPath('examples');
archivo_leer.AddToPath('icalspt');
archivo_leer.AddToPath(obra);
archivo_leer.AddToPath(arch_aux);
archivo_leer.Open(accessMode = SP_AM_READ);
archivo_leer.Close();

arch_aux.SetValue('nomhis.dbf');
archivo_leer.SetLocalName('${FORTE_ROOT}');
archivo_leer.AddToPath('install');
archivo_leer.AddToPath('examples');
archivo_leer.AddToPath('icalspt');
archivo_leer.AddToPath(obra);
archivo_leer.AddToPath(arch_aux);
archivo_leer.Open(accessMode = SP_AM_READ);
archivo_leer.Close();

arch_aux.SetValue('ppfhis.dbf');
archivo_leer.SetLocalName('${FORTE_ROOT}');
archivo_leer.AddToPath('install');
archivo_leer.AddToPath('examples');
archivo_leer.AddToPath('icalspt');
archivo_leer.AddToPath(obra);
archivo_leer.AddToPath(arch_aux);
archivo_leer.Open(accessMode = SP_AM_READ);
archivo_leer.Close();

// Se incluye en la ruta de los archivos la obra que quiere
// consolidarse

arch_aux.SetValue('h:\\forte\\install\\examples\\icalspt\\');
arch_aux.Concat(obra);
arch_aux.Concat('\\nomper.dbf');

// Se instancian objetos y variables
```

```
        ObjetoDBF : dbf_ICA = new;
        comando : TextData = new;
        linea: TextData = new;
        tot_reg : long;
        p_obra : pointer to char;
        p_arch : pointer to char;

// Se obtiene la cantidad de registros del archivo nomper.dbf
        p_arch = strdup(arch_aux.Value);
        tot_reg = ObjetoDBF.CountRec(p_arch);
        p_obra = strdup(obra.Value);
// Se hace un ciclo para insertar los registros en Tabla Consolida
// Se hace llamada al metodo CreaRec, hecho en lenguaje 'C'
  begin transaction
        for j in 1 to tot_reg do
        begin
                num_registros = num_registros + 1;
                comando.SetValue('insert into consolida values (');
                comando.Concat(ObjetoDBF.CreaRec(j, p_obra));
                comando.Concat(')');
                DefaultDBSession.ExecuteImmediate(source = comando.Value);
                exception
                        when e : DBUsageException do
                                Window.MessageDialog(messageText='Error en algún dato del
registro, no será insertado.',
                                                messageType = MT_ERROR);
                                task.ErrorMgr.ShowErrors(clearout=TRUE);
                                num_registros = num_registros - 1;
                                continue;
                        when e: DBValueException do
                                Window.MessageDialog(messageText='Error en algún dato del
registro, no será insertado.',
                                                messageType = MT_ERROR);
                                task.ErrorMgr.ShowErrors(clearout=TRUE);
                                num_registros = num_registros - 1;
                                continue;
        end;
        end for;
 end transaction;
        free(p_obra);
        free(p_arch);
        auxl : TextData = new;
        auxl.SetValue('Carga finalizada - ');
        auxl.Concat(num_registros).Concat(' registros insertados en Consolida -');
        linea_estado = auxl.Value;
        DefaultDBSession.ExecuteImmediate('commit');
exception
        when e : FileResourceException do
        linea_estado=' ';
        Window.MessageDialog(messageText='Alguno o todos los archivos .dbf de la obra no
se localizaron');
        task.ErrorMgr.ShowErrors(TRUE);
end;
return;
end method;
```

```
------------------------------------------------------------
method ConsolidaWin.Init
begin
super.Init();
end method;

------------------------------------------------------------
method ConsolidaWin.Display
begin
// Copyright 1996, Digital Equipment de Mexico S.A. de C.V.

// Intanciacion de clases
Verifica     : boolean = FALSE;
MatrActualiza : Array of IntegerData = new;
MatrBackup   : LargeArray of DatosCons = new;
Evento       : DisplayEvent = new;
Registro     : Integer;

self.Open();
event loop
        when task.Shutdown do
                exit;

        // Realiza el query que se indica en las opciones
        when <QueryCon>.Click do
                Verifica = TRUE;
                // Query por rfc11
                if Opcion = 1 then
                        sql select rfc11,paterno,materno,nombre11
                                into :MatrizDatos
                                from consolida
                                order by 1;
                        sql select rfc11,paterno,materno,nombre11
                                into :MatrBackup
                                from consolida
                                order by 1;
                // Query por Nombre
                elseif Opcion = 2 then
                        sql select rfc11,paterno,materno,nombre11
                                into :MatrizDatos
                                from consolida
                                order by 2,3,4;
                        sql select rfc11,paterno,materno,nombre11
                                into :MatrBackup
                                from consolida
                                order by 2,3,4;
                // Query por filtro
                elseif Opcion = 3 then
                        FWin : Filtro = new;
                        OpFiltro : integer;
                        Condicion : TextData;
                        // Despliegue de la ventana del filtro
                        FWin.Display(OpcFiltro=OpFiltro,ResFiltro=Condicion);
                        // De acuerdo al valor de la opcion obtenida en el filtro se
```

```
// realiza la consulta adecuada
case OpFiltro is
        when 11 do
                sql select rfc11,paterno,materno,nombre11
                    into :MatrizDatos
                from consolida
                        order by 1;
                  sql select rfc11,paterno,materno,nombre11
                    into :MatrBackup
                from consolida
                        order by 1;
         when 21 do
                sql select rfc11,paterno,materno,nombre11
                    into :MatrizDatos
                from consolida
                        order by 2,3,4;
                  sql select rfc11,paterno,materno,nombre11
                    into :MatrBackup
                from consolida
                        order by 2,3,4;
        when 31 do
                sql select rfc11,paterno,materno,nombre11
                    into :MatrizDatos
                from consolida
                        order by 4,2,3;
                  sql select rfc11,paterno,materno,nombre11
                    into :MatrBackup
                from consolida
                        order by 4,2,3;
        when 12 do
                sql select rfc11,paterno,materno,nombre11
                    into :MatrizDatos
                from consolida
                where rfc11 like :Condicion.Value
                        order by 1;
                  sql select rfc11,paterno,materno,nombre11
                    into :MatrBackup
                from consolida
                where rfc11 like :Condicion.Value
                        order by 1;
        when 22 do
                sql select rfc11,paterno,materno,nombre11
                    into :MatrizDatos
                from consolida
                where paterno like :Condicion.Value
                        order by 2,3,4;
                  sql select rfc11,paterno,materno,nombre11
                    into :MatrBackup
                from consolida
                where paterno like :Condicion.Value
                        order by 2,3,4;
        when 32 do
                sql select rfc11,paterno,materno,nombre11
                    into :MatrizDatos
                from consolida
```

```
                                        where nombre11 like :Condicion.Value
                                                order by 4,2,3;
                                          sql select rfc11,paterno,materno,nombre11
                                              into :MatrBackup
                                         from consolida
                                        where nombre11 like :Condicion.Value
                                                order by 4,2,3;
                        end case;
                end if;
                <MatrizDatos>.AllowsAppend = FALSE;

                // Cuando se modifica un campo del registro se guarda el numero de registro
                // para llevar control de los registros modificados
                when <MatrizDatos>.AfterRowValueChange(Evento,Registro) do
                        intAux : IntegerData = new;
                        intAux.SetValue(Registro);
                        MatrActualiza.AppendRow(intAux);

                // Se actualiza la Base de Datos de acuerdo a las modificaciones
                // que se hayan realizado en la pantalla
                when <Actualiza>.Click do
                        if MatrActualiza.Items = 0 then
                                self.Window.MessageDialog('No se ha efectuado ningun
cambio');
                        else
                                // Actualizacion de la tabla
                                if self.Window.QuestionDialog('Esta seguro de actualizar la
tabla Consolida con los nuevos valores',
                                                        BS_YESNO, BV_NO) = BV_YES then
                                        indice : integer;
                                        begin transaction
                                                // Se hace un ciclo para actualizar todos los
registros que
                                                // fueron modificados, con los valores que
aparecen en
                                                // pantalla
                                                for j in 1 to MatrActualiza.Items do
                                                        indice = MatrActualiza[j].Value;
                                                        sql update consolida
                                                  set rfc11 = :MatrizDatos[indice].rfc11,
                                                  paterno = :MatrizDatos[indice].paterno,
                                                        materno =
:MatrizDatos[indice].materno,
                                                        nombre11 =
:MatrizDatos[indice].nombre11
                                                       where rfc11 =
:MatrBackup[indice].rfc11
                                                        and paterno =
:MatrBackup[indice].paterno
                                                        and materno =
:MatrBackup[indice].materno
                                                        and nombre11 =
:MatrBackup[indice].nombre11;
                                                end for;
                                        end transaction;
```

```
                                        MatrActualiza.Clear(); // Limpia la matriz que lleva el
control de los registros modificados
                                    end if;
                                end if;
                        when <Salir>.Click do
                        exit;
end event;
self.Close();
end method;

----------------------------------------------------------------
method Filtro.Init
begin
super.Init();
TextoFiltro = new;
end method;

----------------------------------------------------------------
method Filtro.Display(output OpcFiltro: Integer,
        output ResFiltro: Framework.TextData)
begin
// Copyright 1996, Digital Equipment de Mexico S.A. de C.V.

ResFiltro = new;
self.Open();
event loop
        when task.Shutdown do
                exit;
        when <Aceptar>.Click do
        // Verifica cual opcion de consulta fue la seleccionada
        // y verifica el valor que se dio en el campo de Filtro,
        // ambos valores los deja en las variables de salida OpcFiltro y ResFiltro
                if OpcionFiltro = 1 then
                        if TextoFiltro.ActualSize=0 then
                                self.Window.MessageDialog('No se especifico el filtro de RFC,
la consulta sera general');
                                OpcFiltro = 11;
                        else
                                OpcFiltro = 12;
                                ResFiltro.SetValue(TextoFiltro);
                        end if;
                elseif OpcionFiltro = 2 then
                        if TextoFiltro.ActualSize=0 then
                                self.Window.MessageDialog('No se especifico el filtro de
Apellido Paterno, la consulta sera general');
                                OpcFiltro = 21;
                        else
                                OpcFiltro = 22;
                                ResFiltro.SetValue(TextoFiltro);
                        end if;
                else
                        if TextoFiltro.ActualSize=0 then
                                self.Window.MessageDialog('No se especifico el filtro de
Nombre, la consulta sera general');
                                OpcFiltro = 31;
```

```
                    else
                        OpcFiltro = 32;
                        ResFiltro.SetValue(TextoFiltro);
                    end if;
                end if;
                exit;
end event;
self.Close();
end method;

------------------------------------------------------------
method Compacta.Init
begin
super.Init();
end method;

------------------------------------------------------------
method Compacta.Display
begin
// Copyright 1996, Digital Equipment de Mexico S.A. de C.V.

self.Open();
event loop
    when task.Shutdown do
        exit;

    // Despliega algunos campos de la tabla compacta
    // Dichos campos corresponden a valores obtenidos del calculo del ISPT
    when <Consulta>.Click do
  // Consulta con ordenamiento por rfc11
        if Opcion = 1 then
            sql select rfc11,paternoll' 'llmaternoll' 'llnombre11 "nombre",
                nvl(d01,0) "PT", nvl(d02,0) "PE", nvl(d56,0) "Aguinaldo",
                nvl(d54,0) "AguinaldoExento", nvl(isr_cau,0) "IsrRet",
                nvl(per_gra,0) "IsrCal", nvl(tot_ing,0) "DifIsr"
                    into :MatrizRes
                    from compacta
                    order by 1;
        // Consulta con ordenamiento por apellido paterno, apellido materno, nombre
        elseif Opcion = 2 then
            sql select rfc11, paternoll' 'llmaternoll' 'llnombre11 "nombre",
                nvl(d01,0) "PT", nvl(d02,0) "PE", nvl(d56,0) "Aguinaldo",
                nvl(d54,0) "AguinaldoExento", nvl(isr_cau,0) "IsrRet",
                nvl(per_gra,0) "IsrCal", nvl(tot_ing,0) "DifIsr"
                    into :MatrizRes
                    from consolida
                    order by paterno,materno,nombre11;
        end if;
        <MatrizRes>.AllowsAppend = FALSE;  // No se permite insertar registros

    when <Salir>.Click do
        exit;
end event;
self.Close();
end method;
```

```
-- END METHOD DEFINITIONS
HAS PROPERTY
        CompatibilityLevel = 0;
        ProjectType = APPLICATION;
        Restricted = FALSE;
        MultiThreaded = TRUE;
        LibraryName = 'impuesto';
        StartingMethod = (class = ImpuestWin, method = Display);


end Impuestos;
```

Programa para manipulación de multimedia.

```
-- START METHOD DEFINITIONS

-----------------------------------------------------------
method W1.Init
begin
super.Init;
Video = new;

end method;

-----------------------------------------------------------
method W1.Display
begin
// Copyright 1996, Digital Equipment de Mexico S.A. de C.V.
open;
event loop

        when <SalirBoton>.click do
                exit;

        when task.shutdown do
                exit;

        when <VerBoton>.Click do
                // Se verifica que la maquina donde corre sea una PC
                if task.Part.OperatingSystem.MachineArch <> OS_MA_PC then
                        Window.MessageDialog('La VideotICA por el momento solo puede
correrse en una PC');
                else
                        // Se hace llamada al metodo Play de la clase QuickTime
                        // al cual se le da como parametro el nombre de la
                        // pelicula a ver
                        qt : QuickTime = new;
                        qt.Play(Video.TextValue.value);
                end if;
end event;
close;
end method;
```

```
-- END METHOD DEFINITIONS
HAS PROPERTY
        CompatibilityLevel = 0;
        ProjectType = APPLICATION;
        Restricted = FALSE;
        MultiThreaded = TRUE;
        LibraryName = 'videotic';
        StartingMethod = (class = W1, method = Display);


end VideotICA;

-- START METHOD DEFINITIONS

-----------------------------------------------------------
method QuickTime.Init
begin
super.Init;
end method;

-----------------------------------------------------------
method QuickTime.Play(input c: string)
begin
// Copyright 1996, Digital Equipment de Mexico S.A. de C.V.

// Se declara el comando a ejecutar
cmd : Textdata = new(value='c:\\windows\\player.exe c:\\multimed\\');

// Se le pasa como parametro el video en particular a exhibir
cmd.ConCat(c);

// Se ejecuta el comando
task.part.operatingsystem.RunCommand(Command=cmd, isSynchronous=FALSE);

end method;
-- END METHOD DEFINITIONS
```

Bibliografía.

Donovan John J.
Business Re-Engineering with Technology. An Implementation Guide
Cambridge Technology Group, Inc.
Cambridge MA., 1993

Inmon William H.
Developing Client/Server Applications
Revised Edition
QED Publishing Group
U.S.A., 1993

Pressman Roger S.
Ingeniería del Software. Un Enfoque Práctico
Segunda Edición
McGraw-Hill
España, 1990

Vaskevitch David
Client/Server Strategies. A Survival Guide for Corporate Reengineers
IDG Books Worldwide, Inc.
U.S.A., 1993

A Guide to the Forté Workshops
Release 2
Forté Software, Inc.
U.S.A., 1995

System Management Guide
Release 2
Forté Software, Inc.
U.S.A., 1995

Integrating with External Systems
Release 2
Forté Software, Inc.
U.S.A., 1995

TOOL Reference Manual
Release 2
Forté Software, Inc.
U.S.A., 1995

Accesing Databases
Release 2
Forté Software, Inc.
U.S.A., 1995

System Monitor Library
Release 2
Forté Software, Inc.
U.S.A., 1995

Framework Library
Release 2
Forté Software, Inc.
U.S.A., 1995

Display Library
Release 2
Forté Software, Inc.
U.S.A., 1995