CAPITULO I

INTRODUCCIÓN

1.1 ASPECTOS GENERALES

En la actualidad la Tecnología Informática tiende a implantar nuevos modelos para diseño hardware y software, y una de ellas es la Tecnología Cliente/Servidor que en nuestro medio local no es muy conocida a profundidad, al igual que el uso de aplicaciones de Base de Datos Distribuidas que utilicen réplicas de datos e Internet en la solución de aplicaciones informáticas. Además que existen Técnicas de Automatización de Software que facilitan el diseño de sistemas que no son muy explotadas, tanto por su desconocimiento como por sus costos.

La tendencia al momento en las organizaciones es trabajar en conjunto para alcanzar metas en equipos interdepartamentales, cuyos miembros no necesariamente pueden estar en el mismo lugar físicamente y que necesitan estar al día en cuanto al manejo de su volumen de información.

Para algunas tareas, el grupo podría requerir de información en línea producida por sistemas transaccionales, para otras solo se necesitaría de copias de ciertos datos específicos en diferente espacio y tiempo.

El tema de este trabajo de investigación es precisamente estudiar la distribución de datos como un método que permita la globalización de la información.

El capítulo I realiza una introducción al Tema: "Replicación en Aplicaciones Distribuidas y su Aplicación al Sistema de Administración Académica", ofrece una visión de los objetivos y alcances del proyecto, así como justifica la realización del mismo. El capítulo II inicia con los fundamentos básicos del diseño de Aplicaciones Cliente/Servidor 2 y n-capas, incluye claro está el diseño para la Internet. El Capítulo III es en sí el más importante de la presente Investigación ya que se pone énfasis en la distribución de datos y el Uso de la Replicación como metodología principal.

Se explica el cuando y él porque distribuir datos, y se brinda algunos ejemplos de distribución de datos como las soluciones móviles. El Capítulo IV pone en práctica el Estudio realizado en el Diseño del prototipo para la administración académica. En el cual se analiza los distintos productos del mercado en cuanto a su performance y facilidad de implementación. Se utiliza una metodología orientada a prototipos y objetos basados en UML (Unified Modeling Languaje, estándar de la industria para este tipo de soluciones) para un diseño acorde a las necesidades de la Universidad. Finalmente el capítulo V contiene las conclusiones y recomendaciones que se obtuvieron al finalizar la presente investigación.

1.2 JUSTIFICACION DEL TEMA

A continuación se describen los aspectos importantes que llevo a la decisión de realizar el presente trabajo de investigación.

- El aporte del proyecto en el ámbito de la investigación es muy importante debido a que en nuestro medio y específicamente en la UTN se conoce muy poco de la tecnología Cliente/Servidor, la utilización de CASE en la automatización de Software además de la Aplicaciones Distribuidas. Es decir, se limita solo a los textos y a una experiencia casi nula en aplicaciones informáticas en ese entorno, razón por la cual mediante el

aporte investigativo y a la práctica en sí, se dejará un gran legado para que los nuevos estudiantes que deseen realizar sus proyectos en base a esta Tecnología les sirva como consulta o guía para el desarrollo de sus aplicaciones.

- Tendencia de la Tecnología Informática al Internet.
- Uso de CASE (Computación Asistida por Ingeniería de Software) para mejorar la productividad del desarrollo de software.
- Aplicaciones Distribuidas como solución al manejo académico de los datos en la Universidad, debido a la topología existente y esquemas de replicación.
- Este Sistema es el más importante de la UTN por que administrará los datos de la formación profesional de los estudiantes, que es la esencia de la Universidad.
- El proyecto tendrá la capacidad de realizar todos los procesos que se refieren a la matriculación y manejo de los datos, sean notas de los estudiantes y todo lo referente a la información personal de los mismos, todo esto manejado en las diferentes unidades académicas.
- Es fundamental anotar que a través del proyecto mediante el INTERNET la Universidad mantendrá una página Web con los datos académicos.
- El proyecto dará un acceso a la información estudiantil de una forma rápida y un buen manejo de la misma evitando de esta forma contratiempos tanto para el personal administrativo como estudiantil.

- Con el desarrollo del proyecto se favorecerá directamente la Universidad Técnica del Norte, Estudiantes, Personal Administrativo y Académico.

1.3 OBJETIVOS

1.3.1 Objetivo General

Desarrollar e Implementar un Modelo Informático en la Universidad Técnica del Norte para la gestión de la información académica-estudiantil de alta calidad y bajo costo, generado en todas las unidades Académicas de la Institución, basándose en la utilización de Técnicas de Replicación en entornos Distribuidos, Aplicaciones n niveles, y el uso de la Metodología UML para el diseño de Sistemas.

1.3.2 Objetivos Específicos

1. Investigar a profundidad el uso de la Metodología UML Unified Modeling Languaje asistida por CASE para construir una aplicación Informática.
2. Estudiar varios tipos de soluciones de DBMS que se puedan aplicar a este tipo de aplicaciones.
3. Analizar las diferentes topologías de Replicación en Sistemas Distribuidos.
4. Implementar una aplicación Intranet que permita un control y automatización de los datos estudiantiles. Además definiendo las seguridades para la protección de los datos.
5. Contribuir al incremento de la calidad de los servicios que brinda la UTN a la sociedad.
6. Realizar un Estudio minucioso y consensual sobre las alternativas y costos a usarse en el modelo informático.
7. Llevar un registro computarizado del récord académico de todos los estudiantes de la UTN

8. Mejorar los procesos de Matriculación, registro de notas y promociones mediante la aplicación de los últimos adelantos de la Tecnología Informática.
9. Brindar a través de un software de desarrollo de aplicaciones Cliente/Servidor la mejor interfaz gráfica para que esta sea amigable y confiable hacia todas las personas que se beneficiarán con la misma.
10. Investigar el Diseño, Administración y Comunicación de Redes Locales para a futuro implantar las mejores soluciones hardware y software para beneficio de la Universidad y de la Región Norte del País.

1.4 ALCANCE DEL PROYECTO

El presente trabajo de investigación empieza definiendo lo que es Cliente/Servidor a través de sus diferentes tipos y metodologías. Como la Tecnología Informática ha evolucionado este concepto desde los tradicionales sistemas de 2 capas a lo que hoy se conoce como computación distribuida y aplicaciones n-capas hacia la Internet. Haciendo énfasis en sus ventajas y limitaciones en cuánto a performance, productos, plataformas y costos.

Definirá de la manera más clara todos los conceptos referentes a este tipo de soluciones. Una vez puesto en claro esta fase entraremos en las definiciones de lo que es la distribución de aplicaciones y datos basados en lo que es la Replicación.

El porque y cuando distribuir datos que es uno de los paradigmas de este tipo de soluciones.

Analizaremos los 2 tipos de Replicación tanto Asincrónica como Sincrónica (Two Phase Commit) y sus metodologías de implementación para distintas necesidades.

El alcance de este trabajo incluirá el Desarrollo de un prototipo inicial que permita gestionar la Administración Académica de la Universidad Técnica del Norte generado a través de la Metodología UML y en los resultados de nuestra investigación para generar la mejor solución tanto en costo-beneficio.

Con los datos generados por esta Aplicación se puede implementar Sistemas de Tipo Gerencial utilizando herramientas business-intelligence dando así paso a nuevos trabajos de investigación.

Cabe mencionar que gracias al apoyo de las Empresas Aseinfors y Uniplex S.A. se utilizó productos como Informix Dynamic Server, Data Director, Enterprise Application Studio, Enterprise Application Server, Power Designer y Adaptive Server Enterprise para el Diseño, Implementación y Pruebas finales del prototipo.

CAPITULO II

TECNOLOGÍA INFORMATICA IT

2.1 Cliente/Servidor

Es un ambiente abierto y flexible, en la que la plataforma predominante será la que tiene el mejor costo/desempeño en la mayoría de los servicios. El ambiente cliente/servidor promueve la apertura y flexibilidad, es decir provee participación competitiva de una amplia variedad de plataformas cliente y servidor, sistemas manejadores de bases de datos (SQL), y varias herramientas para el desarrollo de la interfaz gráfica, que es la parte que verá el usuario, con la que interactuará con los datos.

En una aplicación tradicional c/s, un programa corre en una estación de trabajo cliente, y este accede a una base de datos corriendo en una máquina servidor o host. En este modelo, la interfaz de usuario y la lógica del negocio están juntos en la computadora cliente.

En la parte cliente se encuentra el código que implementa la funcionalidad de la aplicación. En este código se encuentran tanto los elementos necesarios para desarrollar la interfaz gráfica de usuario (GUI) y los métodos para soportar la lógica del negocio. En la parte del servidor (brinda servicios al cliente), estarán los datos, y el servidor se encargará de procesar las peticiones y requerimientos del cliente, y devolver la información hacia el cliente en un formato entendible para este, y a través del, se presentarán los datos al usuario final, que se beneficiará de los mismos. En la parte media están los protocolos e interfaces de red, las cuales son gestionadas desde el cliente, para la comunicación con el servidor.

El término servidor se aplica a cualquier programa que ofrece un servicio y que puede ser accedido a través de la red. Los servidores aceptan peticiones recibidas a través de la red, realizan el servicio y regresan el resultado al que generó la petición.

Un programa en ejecución, se convierte en cliente cuando envía una petición a un servidor y espera una respuesta.

Los servidores pueden realizar tareas simples o bastante complejas. Si el principal objetivo de una máquina es el de soportar un programa servidor en particular, entonces el término "servidor" puede ser aplicado también a dicha máquina.

Un servidor comienza su ejecución antes del inicio de cualquier interacción y (usualmente) continúa aceptando peticiones y enviando respuestas indefinidamente. Un cliente es cualquier programa que efectúa una petición y espera una respuesta, (generalmente) termina después de usar un servidor un número finito de veces. Un servidor espera una petición sobre un puerto bien conocido, el cual ha sido reservado para un cierto servicio. Un cliente reserva un puerto arbitrario y no usado para poder comunicarse.

2.1.1 ARQUTECTURA

Este modelo, tiene básicamente 2 capas o niveles: la del cliente y la del servidor. Adicional en la construcción de una aplicación cliente/servidor, tenemos términos, que se refieren al lado en el que se localiza la lógica del negocio, que puede ser en el lado del cliente o en el lado del servidor.

Si la lógica del negocio está en el lado del cliente, como ocurre en las aplicaciones con bases de datos, se denomina al cliente: **cliente grande,** y si por el contrario, el código, o la gran mayoría

de este, se encuentra en el lado del servidor, como procedimientos almacenados o disparadores, se denomina al servidor: **servidor amplio.**

Esta arquitectura se fundamenta principalmente en el acceso remoto a los datos, y el procesamiento y manejo de los mismos se lo realiza en un gran porcentaje en el lado del cliente, esto nos lleva a pensar en varias posibilidades:

a) Acceso de varios clientes al servidor.
b) Capacidad y ancho de banda de la red por la que circularán los datos.
c) Bloqueos sobre los datos que están en escritura.

Las respuestas a esta interrogantes es variada y no está enmarcada en algo rígido, y puede tomar matices intermedios, y no extremos.

Las aplicaciones tienen una capa de presentación, que es mostrada al usuario, y permite que interactúe con la aplicación. La capa de presentación, o interfaz de usuario, consiste de varios elementos visuales, y el código de la aplicación que controla la interacción del usuario con estos elementos. Este código de aplicación corre en la computadora del usuario final, también llamada computadora cliente.

El tipo de capa de presentación que se implementará, depende de el tipo de actividad que el usuario realizará, y de cómo el negocio necesite se implemente la solución informática. El tipo de capa de presentación afecta directamente a la cantidad de software que será instalado y configurado en la computadora cliente, y por lo tanto afecta también a los recursos de la máquina.

Existen básicamente tres capas de presentación:

a) De plataforma específica: es una GUI designada por el sistema operativo de la máquina cliente, es compilada y corrida en un sistema operativo específico. El desarrollo está supeditado directamente al sistema operativo que se quiere soportar, y también a las herramientas de desarrollo del mismo. También se puede utilizar componentes ActiveX[1], Applets o Plugins para correr GUI de otra plataforma.

Los beneficios palpables de esta GUI es que se manejaran controles a los que están acostumbrados los usuarios, y por lo tanto, puede deducir rápidamente la forma de interactuar con la interfaz. Puede ser desarrollada en cualquier leguaje de programación visual ejemplo Power Builder.

b) **De plataforma independiente:** La programación en el lenguaje Java, hace posible el desarrollo de una GUI de plataforma independiente, es decir, se puede implementar el mismo módulo a diferentes sistemas operativos.

c) **Interfaz de página web:** El usuario ve la aplicación en un web browser. Una interfaz de página web, es básicamente desarrollada para aplicaciones de internet o intranets, Estas aplicaciones pueden consistir solamente en páginas web, o también incluir en ellas applets Java, conexión a base de datos, uso de componentes, etc .

Como beneficio es que la aplicación no debe ser instalada en cada máquina de usuario, sino que el browser las requiere a un servidor web, y de esta forma, la páginas son bajadas como se necesiten.

Este tipo de modelo permite realizar cambios rápidos de interfaz en una sola parte centralizada (el servidor web). Además de esto, es independiente del sistema operativo en que se presenta, en el lado cliente sólo se requiere el browser.

[1] Componente de Software Estándar Microsoft utilizado en páginas html.

En cuanto a las desventajas, tenemos que los web browsers tienen diferente soporte para las versiones de HTML, lo que podría repercutir en la forma de mostrar las páginas en el browser. A demás de esto, pueden tener diferencias de soporte para el código de script embebido en las páginas.

En cuanto a lo que se refiere del **acceso a datos o conectividad**, es decir, cuando la aplicación requiere conectarse a bases de datos, o a algún otra fuente de datos, lo hace de varias formas. Esto permite trabajar con varios DBMS, y si se cambia el DBMS, la aplicación seguirá corriendo, mientras se mantenga el mismo convenio de interfaz de red. En la ecuación cliente/servidor, este elemento se denomina **middleware.**

Existen cuatro clases generales de interfaz:

a) **ODBC:** se puede utilizar esta interfaz para acceder a bases de datos que soporten la misma. ODBC es un API de conectividad abierta a bases de datos desarrollada por Microsoft, que da un acceso estándar a los data sources. Esta interfaz permite a la aplicación hablar a ODBC, y la misma Habla al actual data source.

   Dentro de esta clasificación tenemos también a **ADO, RDO y DAO[2].**

b) **JDBC:** esta es utilizada para aplicaciones Java, especifica un estándar de acceso a las bases de datos y a sus tablas.

[2] Estándares de Microsoft para conexión con base de datos relacionales. Actualmente ADO es el mejor.

c) **OLE DB:** es un API desarrollado por Microsoft. Este es un componente de el software Microsoft's Data Acces Components (MDAC).

d) **Interfaz nativa:** esta interfaz provee quizá el mejor acceso a las bases de datos, ya que el rendimiento es mejor, porque está construida para una base de datos específica, y entiende las características de la misma de una mejor forma que ODBC. También pueden ser usadas en los casos que tal vez la DBMS no tenga un soporte para ODBC, y si el proveedor de la herramienta frontal lo da.

Una aplicación, por lo general no pertenece a ninguna de estas clasificaciones, si no que puede tener una mezcla de las mismas.

Los términos cliente y servidor, suponen que cada parte se ejecuta en diferentes ordenadores, y que los mismos estarán unidos entre sí a través de una red, aunque también se puede dar el caso de que ambos corran en la misma máquina. También se debe tomar en cuenta que un programa puede ser servidor, cliente o jugar ambos papeles.

2.1.2 CARACTERISTICAS

Entre las principales características que ofrece este sistema tenemos las siguientes:

a) Servicio: se refiere a que en esta relación, debe existir comunicación entre procesos que corren en máquinas diferentes. EL proceso **SERVIDOR**: Proveedor de servicios a los clientes.
El proceso **CLIENTE:** Consumidor de servicios brindados por el servidor. En virtud de esto, hay una separación funcional clara basada en la idea de servicio.

b) Recursos Compartidos: un servidor puede servir (en base a los recursos que el servidor tiene)a varios

clientes al mismo tiempo, es decir, soporta varias conexiones con los clientes, siendo en servidor el ente regulador de acceso a los recursos que requieren los clientes.

c) Transparencia de ubicación: esta característica se refiere a que para el cliente, el encontrar donde está él servidor, debe ser transparente.

d) El cliente y el servidor se comunican en base a mensajes, que son iniciados por el cliente. Si en una arquitectura cliente/servidor, se cambia la parte del servidor, el cliente no debe sufrir impacto alguno, si el medio y la forma de comunicarse entre los dos no cambia.

e) Esta es una arquitectura escalable, tanto en lo que se refiere a cantidad de clientes como a las capacidades del servidor. En el primer caso, el crecimiento se denomina horizontal, mientras que en el otro se llama crecimiento vertical.

El sistema cliente/servidor, debe residir en un ambiente completamente seguro para los datos, es decir, debe estar sobre una red fiable y veloz. Durante años, las grandes empresas se han convencido de que la "red" corporativa es la arteria por donde fluye la sangre que mantiene vivo su negocio. Desde el gran servidor de sus oficinas centrales, hasta los servidores de las delegaciones, las estaciones de trabajo de los programadores, la información va fluyendo de unos a otros. Para muchas compañías, la Red es la Empresa.

Si esta red no se mantiene sana, los pedidos no llegan, el inventario no se actualiza, el software no se desarrolla adecuadamente, los clientes no están satisfechos y, fundamentalmente, el

dinero no entra. La necesidad de diagnosticar y reducir la arterioesclerosis de la red, hace que se estén inyectando continuamente nuevas metodologías que subsanen este grave problema.

Una de las consideraciones principales es el desarrollo de la interfaz gráfica, debemos tener en cuenta que el usuario trabajará en la mayoría del tiempo con el mouse, y el programa estará orientado a eventos, es decir que no estará sujeto a un determinado camino o flujo de programa, si no que el usuario será el que guíe la ejecución del mismo, y es el encargado de seleccionar la secuencia de funcionamiento dentro de la aplicación. El sistema realizará alguna tarea en respuesta al evento realizado. El suceso hecho por el usuario, es capturado por unas rutinas del sistema operativo subyacente, estas rutinas son llamadas Gestores de eventos (Event Handlers).

Para que el usuario se sienta satisfecho con la IGU, esta deberá ser:

a) Intuitiva: las operaciones que se pueden realizar en cada ventana deben ser inmediatamente comprendidas.

b) Sencilla: no se debe utilizar muchos elementos de interfaz en una ventana

c) Consistente: se refiere a que operaciones parecidas en distintos sitios, se deben realizar de igual forma.

d) Colores: los usuarios deben ser capaces de trabajar varias horas delante del computador sin molestias en la vista, por el uso indiscriminado de colores fuertes.

e) Productivo: para realizar alguna tarea, el usuario no debe tener que pasar por varios procedimientos largos.

f) Amable: debe tener ayuda en línea, así como también tener un buen manejo de errores y que los mensajes provocados por los mismos sean claros y ayuden a resolver el problema.

g) Validado: los datos ingresados a través de la interfaz, deberán ser válidos en cada caso particular, y no se podrán aceptar datos que no concuerden con el tipo de dato que se quiere obtener.

2.1.3 Aplicaciones:

Existen en la actualidad muchos sistemas que son catalogados como cliente/servidor. Cada una de estas soluciones, se distinguen por la naturaleza del servicio que brindan a sus clientes.

En los **servidores de archivos,** el cliente envía solicitudes de registros de archivos al servidor. Es un servicio de datos muy primitiva, que requiere de varios mensajes entre cliente y servidor para obtener los resultados. Sirven para tener una localización centralizada de documentos de gran tamaño, y se logra compartir archivos en una red.

En los **servidores de bases de datos,** el cliente envía solicitudes de SQL (Structured Query Language) en forma de mensajes, los mismos que son recibidos por el servidor, que entiende este lenguaje y lo interpreta, haciendo uso de la capacidad de proceso del host y del manejo del árbol de ejecución y el plan de sentencias, así como de los índices, para que el SQL enviado sea lo mejor tratado posible.

Los resultados son devueltos al cliente a través de la red, y son procesados por este para presentar los resultados en su propio formato, que a la vez deben ser amigables, para mostrarlos en la interfaz gráfica al usuario final.

Los **servidores de transacciones,** el cliente invoca a componentes o procedimientos remotos que residen en el servidor, los cuales ejecutan componentes SQL como una sola unidad; a estas aplicaciones se las denomina **OLTP** (Online Transaction Procesing), que se utilizan para la toma de decisiones.

Los **servidores de groupware,** permiten el manejo de datos no tradicionales o no estructurados, que son manejados por los servidores de bases de datos, tales como datos de tipo entero o carácter. Los datos no estructurados, tales como imágenes, texto, documentos, flujo de trabajo, son manejados por esta clase de servidores.

Los **servidores de objetos o componentes,** son servidores que suelen ubicarse en medio de un cliente y una base de datos, que juntos, forman un cliente/servidor de tres capas o arquitectura distribuida. En el servidor están componentes (clases), paquetes (como una librería de clases), que cuando se instancian para procesar las solicitudes del cliente, se denomina objetos. Estos no son mas que piezas de software que contiene la lógica necesaria del negocio. En este modelo, los componentes deben ser utilizados concurrentemente por los clientes, por lo que, si alguna regla del negocio cambia, no se necesita cambiar en todas las localizaciones en las que esté instalada el software cliente (en una arquitectura cliente/servidor tradicional), sino que solamente se cambia en el componente dentro del servidor que implementa esa regla del negocio. Para esto, se debe tener un proxy del objeto del servidor, que es el que informa al cliente de los métodos públicos que tienen el componente para ser invocados, instanciar a la clase, y luego realizar el proceso y devolver los resultados al cliente.

En los **servidores web,** los clientes solicitan un documento escrito en el lenguaje HTML en base a una dirección URL, la cual señala la localización del documento, el servidor envía el documento, que es interpretado por el cliente, que es un explorador o browser en el lado del cliente. Esta comunicación se la realiza a través del protocolo HTTP. Los documentos traídos desde el servidor, pueden tener ligaduras a otros documentos. En la actualidad, las páginas HTML, están pasando de ser estáticas y de solo presentación, a ser páginas dinámicas. Java es el lenguaje que permite hacer esto en gran medida.

2.1.4. Desventajas de la arquitectura Cliente/Servidor

- Complejidad en la red el servidor es un dispositivo extra que puede o no utilizar el mismo sistema operativo de los otros servidores.

- Utilización de protocolos especializados en comunicación de esta arquitectura[1] discutidos anteriormente.

- Si la aplicación es grande se hace imprescindible la contratación de un administrador que maneje el sistema, lo cual implica un gasto extra.

2.1.5 Dónde tiene sentido utilizar cliente/servidor?

Las aplicaciones cliente/servidor pueden adoptar muchas formas. Existen casos en que no puede ser factible debido a equipos requisitos de usuario o disponibilidad del software. Adernás se debe tomar en cuenta que para desarrolladores de sistemas que se inclinan a utilizar los más populares motores de base de datos, tendrán toda la infraestructura necesaria para

desarrollar este tipo de arquitectura.

A fin de cuentas siempre que haya un grupo de computadoras trabajando en red y un proceso compartido como base de datos o proceso de mensajes, habrá una oportunidad de utilizar un diseño cliente/servidor.

2.2 Aplicaciones Distribuidas

2.2.1 Concepto de Computación Distribuida

La computación distribuida tiene varios conceptos para cada persona. Desde el punto de vista de Microsoft se tienen las siguientes definiciones:

Para desarrolladores. - Es arquitectura y herramienta que permiten a las apilcaciones estar divididas en partes que operan en múltiples computadoras en ambientes heterogéneos

Para administradores de Sistemas de Tecnología.- Uso eficiente de recursos del sistema mediante la creación de aplicaciones que son ampliamente integradas con el negocio, y que puede ser modificados fácilmente para mantener siempre el ambiente de cambio

Para usuarios finales, es una mejora en acceso a información de una gran variedad de fuentes, sin tener que aprender complejos procedimientos y múltiples passwords.

La computación distribuida es un avance sobre las arquitecturas cliente/servidor. Mientras que la arquitectura cliente/servidor suele proporcionar información y servicios sólo entre servidores y sus clientes, la computación distribuida extiende la compartición de dispositivos de forma que los servidores tengan que compartirse unos a otros y los clientes puedan conectarse a diferentes servidores y compartirse de igual manera. Así los datos no están centralizados[1] sino que se

comparten entre varios servidores que pueden estar geográficamente dispersos y conectados por redes de área extensa(WAN).

La figura muestra un esquema de computación distribuida.

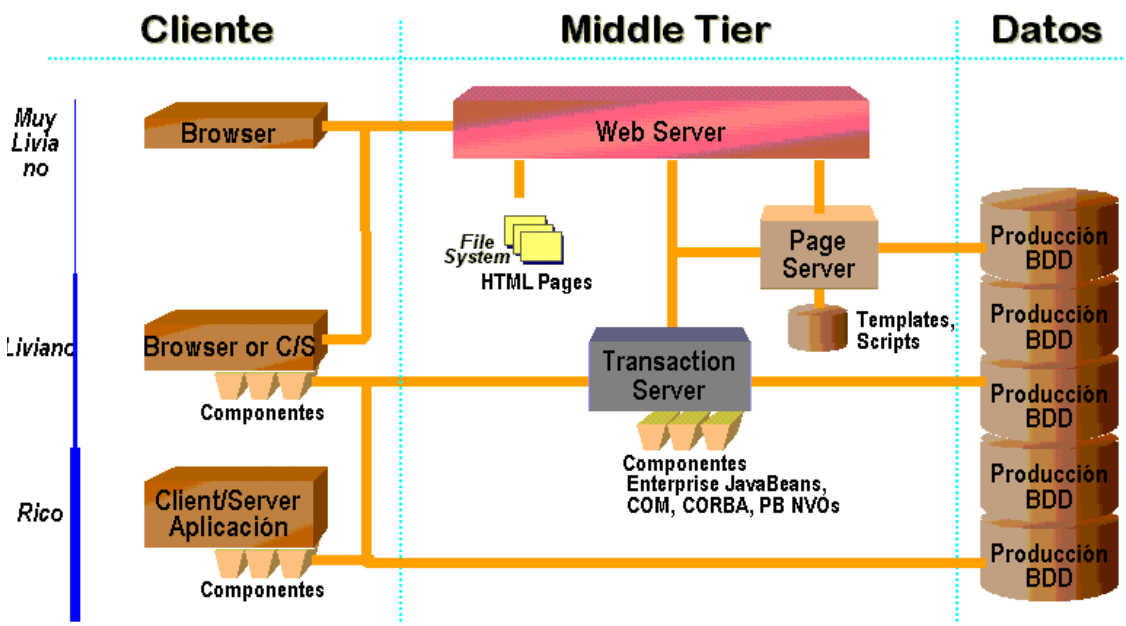


Figura 1 : Arquitectura Distribuida

Fuente: Curso de Sybase EAServer

Descripción:

Se destacan tres capas principales Cliente, Servidor Intermedio, Fuente de Datos. Las aplicaciones en este tipo de arquitectura pueden ser tan livianas como un browser el cual se conecta a una lógica que puede estar ejecutándose en un servidor web o en un servidor de aplicaciones con el fin de que pueda ser reutilizada desde otro tipo de aplicaciones como cliente/servidor.

En resumen proveer un acceso total a los recursos de una organización sin importar su localización y estructura, es el principal objetivo de la computación distribuida. La replicación de datos forma parte del llamado grupo *Middleware*, que no es mas que un conjunto de herramientas y servicios para obtener ambientes distribuidos.

2.2.2 Objetivos

Los objetivos básicos que persigue la computación distribuida pueden ser:

- Construir ambientes de una plataforma corporativa para una organización[1] que sirva como sistema de comunicación *«conectar y listo»>* al que se pueda conectar diversos componentes computacionales.
- Conectar redes y recursos de computación que ya existan como *«islas»* en departamentos y grupos de trabajo
- Proporcionar a los usuarios un sencillo acceso a grandes sistemas informáticos y sistemas de minicomputadoras.
- Proporcionar a los usuarios acceso a datos situados en varios lugares dispersos. Manteniendo al mismo tiempo integridad de información
- Incorporar aplicaciones y tecnologías cliente/servidor en la organización

2.2.3 Servidores de Componentes o Aplicación

Muchos desarrolladores de TI tienen la necesidad de reutilizar muchas de sus aplicaciones para diferentes entornos de programación. Un ejemplo es si tenemos la función que me calcule la tasa de interés en mi sistema de Recursos humanos, De que forma se puede reutilizar este mismo código en mi aplicación de ventas?

La necesidad de responder a esta pregunta obligó a la Industria Informática a crear los llamados Componentes que encapsulan ciertos métodos y eventos que pueden ser reutilizados por diversas aplicaciones y en múltiples plataformas.

Las principales tecnologías o estándares para la creación de componentes son las siguientes:

Common Object Model ++:

COM++ proporcionada por Microsoft. Este tipo de componente es programado por herramientas como Visual basic, Visual C++, Power Builder, pero solo funciona en entornos Windows. Otra forma de llamar a esta tecnología es Active X.

Enterprise Java Bean

Proporcionada por la casa Sun Microsystem, este tipo de componente es programado en lenguaje Java y funciona en todas las plataformas, tanto unix, as/400, windows, etc.

En la actualidad se está convirtiendo en el estándar de la Industria superando a CORBA.

CORBA

Este tipo de componente proporcionado por OMG, es independiente del lenguaje de programación y la plataforma, siendo con este el de mayor portabilidad en la industria.

Todos estos componentes necesitan de un lugar donde recidir, a esto se lo denomina Servidor de Componentes. Los principales en la industria son Microsoft Transaction Server que puede utilizar sólo componentes COM ++, y Enterprise Application Server más conocido como Jaguar CTS que es el líder en la industria al soportar no solo todos los estándares de componentes, sino que se puede utilizar como servidor Web y además implementa seguridades tales como SSL[3]

[3] Secure socket layer, estándar de encripción utilizado para implementar seguridades.

2.3 DESARROLLO PARA LA INTERNET

Lo que en un principio se utilizó para estrategias de comunicación militar, se convirtió en una red llena de información de todo tipo, que además hizo que las comunicaciones entre personas en cualquier parte del mundo sea más rápida y eficiente y que la tendencia actual es comprar y vender en forma electrónica, hoy en día se denomina el Internet.

2.3.1 Tecnologías para Construir Sitios Web Dinámicos.

Cuando se muestra una página estática, solamente se puede observar información, que sólo se va a poder modificar si se edita la página. En una página dinámica en cambio, la información presentada en ella, variará, según la información que requiera el usuario.

Una página dinámica puede contener la siguiente información:

- Texto HTML estático, visible siempre que la página web es desplegada.

- Consultas SQL incrustadas e instrucciones de programación (JAVA).

- Datos provenientes de la base de datos, cuando la página es generada.

La mayoría de sitios web ofrecen contenido dinámico es decir el usuario a través de su browser envía una petición al servidor web y este retorna la información de acuerdo a este criterio, un ejemplo clásico de esto es yahoo. En la figura 2 podemos ver como se realiza este proceso.

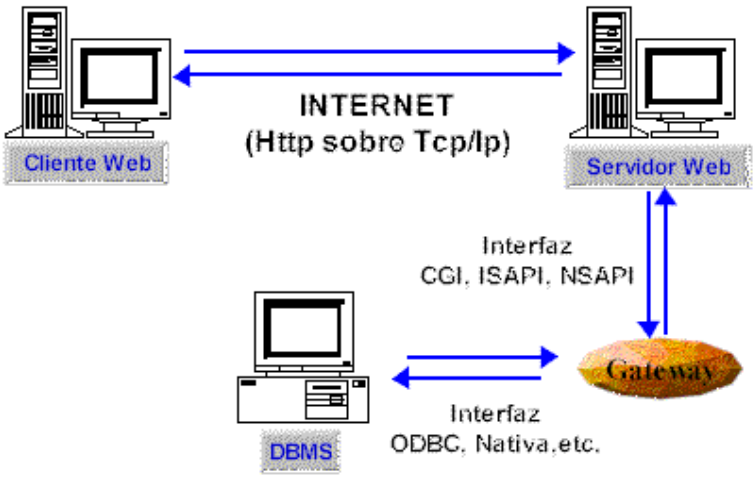


Figura 2: Esquema de comunicaciòn dinàmica para un sitio web

Fuente: Internet

Entre los principales estándares propuestos para crear aplicaciones dinámicas para internet tenemos las siguientes:

CGI, ISAPI, NSAPI, ASP, JSP.

CGI (Common Gateway Interface)

Es una interfaz estándar para servidores web. Cuando un servidor web recibe una URL que corresponde a un recurso CGI, este comienza un programa CGI (tal como un script perl) que conecta a la DBMS, realiza las consultas en la base de datos, y retorna la información al servidor web para ser manejada por el cliente web (web browser).

CGI es un estándar que proporciona una interfaz para aplicaciones externas al servidor web, no solamente a un DBMS. Las aplicaciones CGI podrían también simplificar el proceso de archivos. La salida de un programa CGI o script es devuelto al servidor web.

Esta devolución podría ser un documento HTML con datos de la base de datos, una imagen, a referencia a otro documento, o algún otro tipo de información entendida por el servidor web.

NSAPI- ISAPI

Algunos proveedores de servidores web también proveen interfaces propietarias para sus servidores, por ejemplo: NSAPI (Netscape Application Programming Interface) y Microsoft ISAPI (Internet Server Application Programming Interface).

Este tipo de soluciones funciona de la siguiente manera:

En primer lugar en la página web se crea una forma de ingreso de datos la cual envía la información a una plantilla la cual contiene la consulta a la base de datos, esta a su vez a través de una dll propia de la plataforma se encarga de generar el contenido dinámico que es visualizado en el browser del cliente.

Active Server Pages(ASP)- Java Server Pages(JSP)

ASP es propiedad de Microsoft, es decir solo puede ser implementada en entornos Windows y en el Servidor Web Internet Information Server IIS.

Cuando el usuario envía la petición de consulta está es codificada en Visual Basic Script a través de un tag special que es entendido por el IIS y le permite generar el resultado a través de un archivo .asp.

ASP puede ser programado desde el bloc de notas así como también en herramientas más avanzadas como Visual Studio.net.

La diferencia con JSP radica en primer lugar en el lenguaje de programación que este caso es Java y lo más importante el alcance en funcionalidad y portabilidad que tiene está tecnología.

Es por eso que JSP es el estándar actual para la creación de sitios en Internet, además los principales fabricantes de Software lo incorpora en sus productos.

Adicional a estas tecnologías existen otras que se denominan de Arquitectura Cerrada o Tecnologías Paralelas.

Tales es el caso de los productos de Informix u Oracle, que solo pueden realizar consultas hacia estos DBMS[4].

En el desarrollo de nuestro prototipo se utilizó la Arquitectura propietaria de Informix, es decir nosotros utilizamos un producto llamado Data Director for Web, que nos permitió crear de manera fácil y rápida la interfaz hacia el Web de nuestro sistema.

Cabe anotar que como señalamos anteriormente esta solución funciona en cualquier plataforma pero solo con base de datos Informix.

[4] Sistema de Manejo de Base de Datos

CAPITULO III

3. REPLICACIÓN EN ENTORNOS DISTRIBUIDOS

3.1.-Conceptos y definiciones

El diseño de un sistema de base de datos distribuido implica la toma de decisiones sobre la ubicación de los programas que accederán a la base de datos y sobre los propios datos que constituyen esta última, a lo largo de los diferentes puestos que configuren una red de ordenadores. La ubicación de los programas, a priori, no debería suponer un excesivo problema dado que se puede tener una copia de ellos en cada máquina de la red Sin embargo, cuál es la mejor opción para colocar los datos: en una gran máquina que albergue a todos ellos, encargada de responder a todas las peticiones del resto de las estaciones (sistema de base de datos centralizado), o podríamos pensar en repartir las relaciones, las tablas, por toda la red.

En él supuesto que se tienda a utilizar la segunda opción, ¿qué criterios se deberían seguir para llevar a cabo tal distribución? ¿Realmente este enfoque ofrecerá un mayor rendimiento que el caso centralizado? ¿Podría optarse por alguna otra alternativa? En los párrafos sucesivos se tratará de responder a estas cuestiones.

Tradicionalmente se ha clasificado la organización de los sistemas de bases de datos distribuidos sobre tres dimensiones: el nivel de compartición, las características de acceso a los datos y el nivel de conocimiento de esas características de acceso (Figura 3). El nivel de compartición presenta tres alternativas: inexistencia, es decir, cada aplicación y sus datos se ejecutan en un ordenador con ausencia total de comunicación con otros programas u otros datos; se comparten sólo los datos y no los programas, en tal caso existe una réplica de las aplicaciones en cada máquina y los datos viajan por la red; y, se reparten datos y programas, dado un programa ubicado en un determinado sitio,

éste puede solicitar un servicio a otro programa localizado en un segundo lugar, el cual podrá acceder a los datos situados en un tercer emplazamiento. Como se comentó líneas atrás, en este caso se optará por el punto intermedio de compartición.

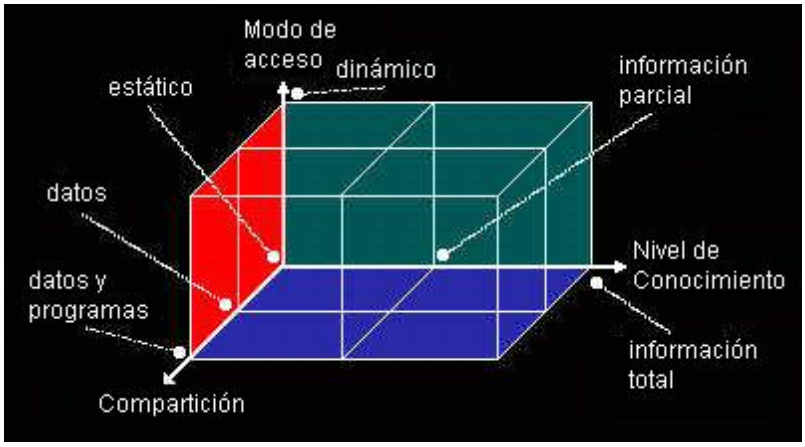


Figura 3. Enfoque de la distribución.

Fuente: Internet

Respecto a las características de acceso a los datos existen dos alternativas principalmente: el modo de acceso a los datos que solicitan los usuarios puede ser estático, es decir, no cambiará a lo largo del tiempo, o bien, dinámico. Ustedes podrán comprender fácilmente la dificultad de encontrar sistemas distribuidos reales que puedan clasificarse como estáticos. Sin embargo, lo realmente importante radica, estableciendo el dinamismo como base, cómo de dinámico es, cuántas variaciones sufre a lo largo del tiempo. Esta dimensión establece la relación entre el diseño de bases de datos distribuidas y el procesamiento de consultas.

La tercera clasificación es el nivel de conocimiento de las características de acceso. Una posibilidad es, evidentemente, que los diseñadores carezcan de información alguna sobre cómo los usuarios

acceden a la base de datos. Es una posibilidad teórica, pero sería muy laborioso abordar el diseño de la base de datos con tal ausencia de información.

Lo más práctico sería conocer con detenimiento la forma de acceso de los usuarios o, en el caso de su imposibilidad, conformarnos con una información parcial de ésta.

El problema del diseño de bases de datos distribuidas podría enfocarse a través de esta trama de opciones. En todos los casos, excepto aquel en el que no existe compartición, aparecerán una serie de nuevos problemas que son irrelevantes en el caso centralizado.

A la hora de abordar el diseño de una base de datos distribuida podremos optar principalmente por dos tipos de estrategias: la estrategia ascendente y la estrategia descendente. Ambos tipos no son excluyentes, y no resultaría extraño a la hora de abordar un trabajo real de diseño de una base de datos que se pudiesen emplear en diferentes etapas del proyecto una u otra estrategia. La estrategia ascendente podría aplicarse en aquel caso donde haya que proceder a un diseño a partir de un número de pequeñas bases de datos existentes, con el fin de integrarlas en una sola. En este caso se partiría de los esquemas conceptuales locales y se trabajaría para llegar a conseguir el esquema conceptual global. Aunque este caso se pueda presentar con facilidad en la vida real, se prefiere pensar en el caso donde se parte de cero y se avanza en el desarrollo del trabajo siguiendo la estrategia descendente. La estrategia descendente (Figura 4) debería resultar familiar a la persona que posea conocimientos sobre el diseño de bases de datos, exceptuando la fase del diseño de la distribución. Pese a todo, se resumirán brevemente las etapas por las que se transcurre.

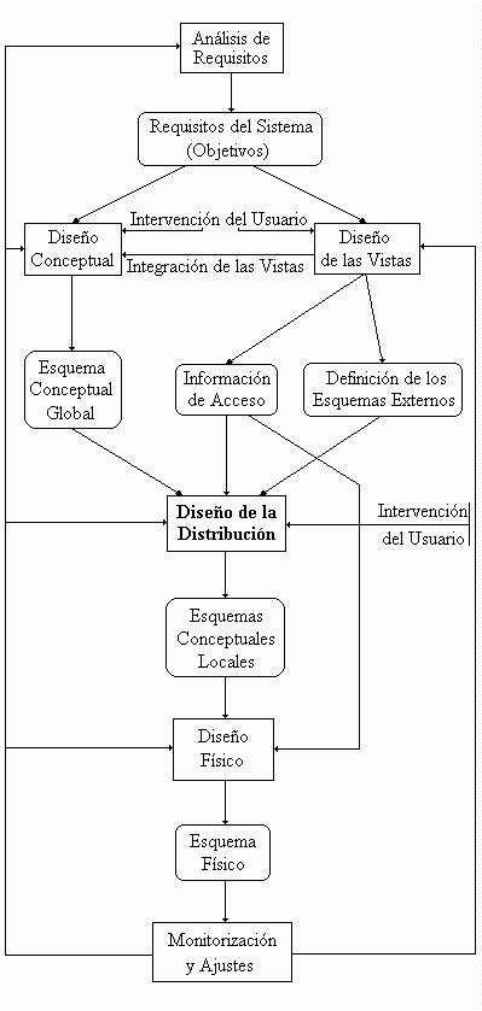


Figura 4: Estrategia Descendente

Fuente: Internet

Todo comienza con un análisis de los requisitos que definirán el entorno del sistema en aras a obtener tanto los datos como las necesidades de procesamiento de todos los posibles usuarios del banco de datos.

Igualmente, se deberán fijar los requisitos del sistema, los objetivos que debe cumplir respecto a unos grados de rendimiento, seguridad, disponibilidad y flexibilidad, sin olvidar el importante aspecto económico. Como puede observarse, los resultados de este último paso sirven de entrada para dos actividades que se realizan de forma paralela.

El diseño de las vistas trata de definir las interfaces para el usuario final y, por otro lado, el diseño conceptual se encarga de examinar la empresa para determinar los tipos de entidades y establecer la relación entre ellas. Existe un vínculo entre el diseño de las vistas y el diseño conceptual. El diseño conceptual puede interpretarse como la integración de las vistas del usuario, este aspecto es de vital importancia ya que el modelo conceptual debería soportar no sólo las aplicaciones existentes, sino que debería estar preparado para futuras aplicaciones. En el diseño conceptual y de las vistas del usuario se especificarán las entidades de datos y se determinarán las aplicaciones que funcionarán sobre la base de datos, así mismo, se recopilarán datos estadísticos o estimaciones sobre la actividad de estas aplicaciones. Dichas estimaciones deberían girar en torno a la frecuencia de acceso, por parte de una aplicación, a las distintas relaciones de las que hace uso, podría afinarse más anotando los atributos de la relación a la que accede. Desarrollado el trabajo hasta aquí, se puede abordar la confección del esquema conceptual global. Este esquema y la información relativa al acceso a los datos sirven de entrada al paso distintivo: el diseño de la distribución. El objetivo de esta etapa consiste en diseñar los esquemas conceptuales locales que se distribuirán a lo largo de todos los puestos del sistema distribuido. Sería posible tratar cada entidad como una unidad de distribución; en el caso del modelo relacional, cada entidad se corresponde con una relación. Resulta bastante frecuente dividir cada relación en subrelaciones menores denominadas fragmentos que luego se ubican en uno u otro sitio.

De ahí, que el proceso del diseño de la distribución conste de dos actividades fundamentales: la fragmentación y la asignación. El último paso del diseño de la distribución es el diseño físico, el cual proyecta los esquemas conceptuales locales sobre los dispositivos de almacenamiento físico disponibles en los distintos sitios. Las entradas para este paso son los esquemas conceptuales locales y la información de acceso a los fragmentos. Por último, se sabe que la actividad de desarrollo y diseño es un tipo de proceso que necesita de una monitorización y un

ajuste periódicos, para que si se llegan a producir desviaciones, se pueda retornar a alguna de las fases anteriores.

Replicación de datos a través de la organización involucra mucho mas que la simple copia de información de un sitio a otro. Replicación de datos es importante debido a que facilita a tas organizaciones proveer a sus usuarios acceso a información actualizada donde y cuando ellos lo necesiten. La replicación de datos puede proveer un amplio espectro de beneficios a saber:

Incluye mejora en rendimiento cuando los recursos centralizados lleguen a saturarse incrementa la disponibilidad de datos, capacidad, y soporte para datawarehousing, además de facilidad para soporte de decisiones. En adición, hace unos pocos años atrás los datos corporativos residieron en localizaciones centralizadas, departamentos remotos accedían a la información que ellos necesitan mediante el establecimiento de conexiones directas a la fuente central de información además a reportes impresos de la unidad central. Todo esto fue sin embargo métodos caros no confiables limitados mientras que los reportes fueron inflexibles y no a tiempo.

3.1.1.-Qué es replicación?

Se ha podido consultar los siguientes conceptos de replicación de base de datos:

ORACLE.- "*Es una tecnología en ambientes distribuidos que mantiene vanas copias de un objeto de base de datos en diferentes sitios, sus diferentes mecanismos aseguran que 105 cambios realizados sobre una de las copias se reflejen sobre los demás y logren convergencia de valores a través del tiempo cuando se presentan actualizaciones simultáneas"* [5]

SQLSERVER.- *"La habilidad de compartir información corporativa en forma efectiva usando todos los recursos de la empresa"*[6]

[5] Traducido al español del libro Oracle Replication Manager , Manuales del Producto
[6] Tomado de Microsoft SQL Server 2000 A fondo, Editorial Microsoft Press 2001

INFORMIX.- ***"****Replicación puede ser simplemente definido como el proceso de generar y reproducir múltiples copias de datos de un sitio a otro."*[7]

A juicio de los autores replicación es acceder y actualizar datos heterogéneos, utilizando recursos propios para consolidar información disponible. Además de distribuir y administrar en forma confiable y oportuna de manera autónoma.

Replicación es mucho más que simplemente mover datos de un lado a otro. Esta técnica demanda de una tecnología extremada. Por lo tanto; el éxito de los sistemas de replicación depende de la tecnología y directrices de un amplio rango de necesidades en la organización.

3.2.- Inicios de la replicación de datos

El concepto de replicación de datos a través de una organización no es nuevo. Las organizaciones han estado distribuyendo copias de sus datos mediante copias en cintas magnéticas(cargando y bajando datos en cintas de diferentes sitios de la empresa). Este método es conocido como *Sneakerware(doméstico).* Las unidades de la organización toman decisiones basados en datos que pueden estar en días o semanas desactualizados. Como resultado de la distribución *copiar y bajar datos,* los datos de una localización puede estar disponible en otras localizaciones. Los datos en cada sitio no son actuales. Por lo tanto, el proceso es a menudo manual y no automático

3.3 Cómo saber si se necesita replicación?

En la organización se pueden originar muchas situaciones de requerimientos de negocios y tópicos de tecnología que debería el administrador de la base de datos empezar a estimar a la replicación como una solución.

[7] Traducido al español del libro Informix Replication Manager, Manuales del producto

A continuación se describen algunas de las directrices a considerar.

Se debería implementar replicación cuando:

- Si se requiere una vista corporativa consolidada de operaciones distribuidas quienes corren en una variedad de DBMSs

- Si se quiere reducir tráfico de red centralizadas

  Si se quiere distribuir datos vi direccionales entre un mainframe, cliente/servidor y sistemas desktops para así asegurar la entrega confiable de datos a todas las localizaciones en la empresa.

- Si se está migrando de sistemas mainframes y necesita proveer períodos transaccionales para mover datos mientras se mantiene los sistemas sincronizados.

- Si se necesita abastecer de información en toda la empresa en ó cerca de tiempo real

- Si la empresa ejecuta operaciones globales en función del tiempo$_1$ en la cual no es factible tener todos los datos en un sitio donde todos puedan acceder.

- Si se quiere dar un control a la unidad de negocios sobre los datos que necesitan, mientras éstos se aíslan de fallas en cualquier parte de la red o en el mainframe$_1$ y mantener su enlace corporativo.

- Si se planifica emplear la técnica datawarehouse o datamart para automatizar el movimiento de los datos entre sistemas.

La replicación puede resolver muchos problemas inherentes al proceso distribuido, es importante reconocer que existen al menos dos casos en los cuales la replicación no es aconsejable:

- Un gran nùmero de registros actualizables en múltiples réplicas.- Estos son propensos a obtener conflictos, lo que significa poner más atención a la resolución de los mismos.

- Cuando la consistencia de datos es crítica.- Particularmente cuando hay datos que tienden a cambiar con bastante frecuencia, aquí la consistencia en muchos de los casos no se podrá garantizar o será difícil.

3.4- Consideraciones Generales para Replicación

Como se ha explicado anteriormente la replicación de datos a través de la organización es mucho más que copiar o mover datos. Un sistema de replicación debe tomar en cuenta las siguientes reglas:

- **Alta disponibilidad de datos.-** Los sistemas de replicación deberían ser confiables **y** no debería exponer las operaciones del negocio a fallas del sistema

- **Reparto o entrega de Información consistente.-** Los sistemas de distribución deben proteger la integridad de datos.

- **Administración fácilmente centralizada.-**

  Los administradores capaces de administrar todos los componentes distribuidos del Replicación desde su escritorio.

- **Acceso a fuentes de datos heterogéneos**.- Los sistemas de deberían ser capaces de mover datos a través de diferentes datos y diferentes vendedores.
- **Autonomía local.-** Cada sitio que recibe datos replicados debería ser libre de decidir qué conjuntos de datos desean recibir, cómo éste vería los datos, cómo accesará a los datos y cómo éste modificará los datos.

  El reto más importante para crear un ambiente distribuido es la fase de diseño. Es extremadamente importante que él administrador o responsable de la base de datos defina: qué datos necesita replicar qué tan frecuente se quiere propagar cambios entre las replicaciones y principalmente cómo evitar o resolver conflictos de actualización en múltiples sitios de replicación

3.5.- Arquitecturas comunes de replicación

Muchos ambientes de replicación son descritos en términos de sus eventuales arquitecturas. Los clientes hablan a menudo de sus necesidades de replicación para referirse a un método de replicación que ellos pueden emplear. Las siguientes son algunas de las arquitecturas comunes de replicación:

Distribución.- También llamado "*Diseminación de datos"*[8] describe un ambiente donde los datos son actualizados en una localización central y luego replicados a sitios remotos pero solo de lectura.

[8] Técnica utilizada para implementar esquemas de distribución de datos.

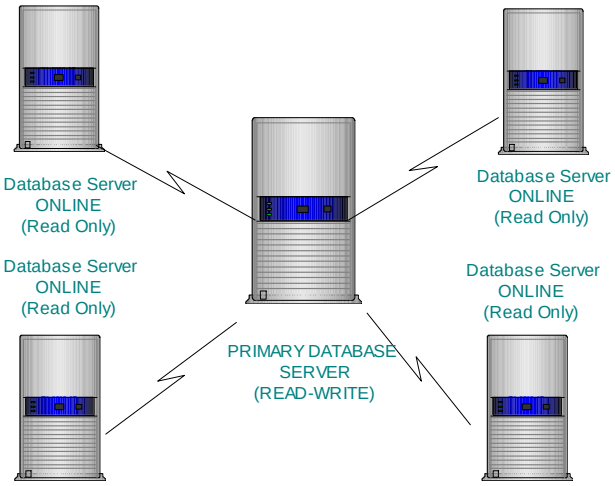


Figura 5: Distribución

Consolidación: Describe un ambiente donde los datos pueden ser actualizados regionalmente y luego ser traídos juntos a repositorios solo de lectura en un servidor central de base de datos.

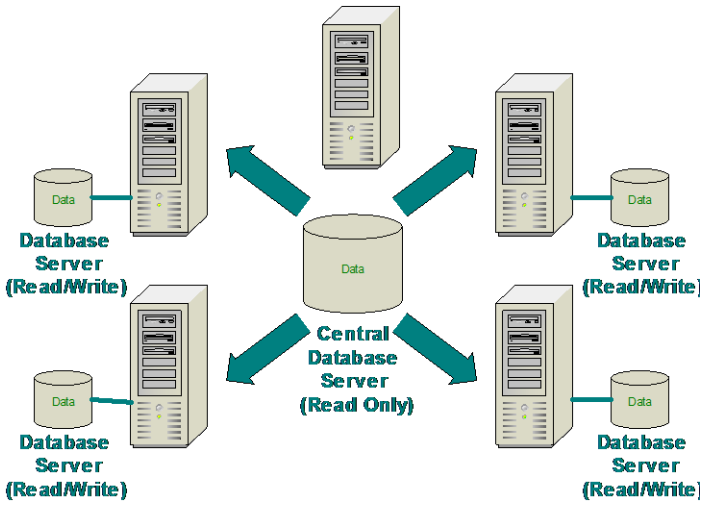


Figura 6: Consolidación De Datos

Fuente: Administration Informix Dynamic Server 2000 Vol. I

3.6 Elemento de Replicación

Una réplica es una definición de los datos (base de datos tablas y columnas) a ser replicadas y una lista a los servidores a los cuales los datos serán replicados.

3.6.1 Mecanismos de Captura de cambios

Los mecanismos más frecuentes usados para la captura de cambios en replicación son: capturas basadas en log y capturas basadas en triggers. En la captura basada en triggers las modificaciones de tos datos disparan triggers que están en la base de datos. Esto activa códigos específicos de replicación en la base de datos origen. Estos procesos ocurren simultáneamente en la misma unidad de trabajo al igual que una transacción, por lo tanto, impacta en el rendimiento del proceso de transacción de la base de datos origen.

Otro problema importante que los sistemas basados en triggers tienen, es como distinguir de los triggers inmersos en la base de datos. Los triggers de la base de datos fueron originalmente introducidos por los vendedores de motores de base de datos para proteger la integridad referencial y corporativa. Para ellos, los sistemas de replicación basados en triggers fueron una consecuencia lógica para soporte de sus productos. Como resultado de esto, la integridad referencial se protegía pero un severo problema en rendimiento y riesgo de administración se venía.

Los sistemas basados en log, por otro lado, puede operar como parte normal en el proceso de log de una base de datos. Este mecanismo incurre en un gasto mínimo del sistema. El concepto en si es que el log de transacciones es la mejor fuente de captura de cambios en la fuente de datos.

Las modificaciones en la base de datos fuente son detectadas y propagados sin interferir con las operaciones normales del sistema fuente. Sin. embargo, no todos los sistemas de replicación basados en log son los mismos. Una diferenciación clave es cómo el mecanismo integrado de captura está en la base de datos- Por ejemplo, los productos basados en log que

utilizan servidores externos y procesos adicionales tienen ventajas sobre la captura de transacciones basados en log debido al gasto adicional de transferencia de datos. Otra característica que determina el éxito basado en log es que cada entrada en el log forma parte del proceso de captura de transacciones.

3.7- Técnicas de replicación

Existen dos categorías fundamentales de replicación: sincrónica y asincrónica. Cada una ofrece capacidades y opciones para diferentes propósitos. En la replicación sincrónica, los datos replicados son actualizados inmediatamente cuando la fuente es actualizada. En la replicación sincrónica la integridad de datos es protegida y garantizada por el protocolo two-phase commit. Más adelante se detallará el estudio de esta técnica. Por otro lado la replicación asincrónica es un método en el cual la base destino es actualizada después de que la base origen ha sido actualizada. En resumen, las dos principales técnicas de replicación tienen sus propias definiciones formas de utilización, ventajas y desventajas que merecen especial estudio en orden de obtener una visión global de los principales servicios de las mismas. En el desarrollo del Aplicativo de nuestra tesis se utilizó la técnica de replicación asincrónica basada en el Modelo de Consolidación.

3.7.1.- Replicación Sincrónica.- Two-Phase Commit

La introducción de la tecnología two-phase commit en los últimos años de la década de los 80, fue la mejor y eficiente forma de distribuir y compartir datos. Two-phase commit permite la sincronización de distribución de datos. Una transacción sólo será aceptada si todos los sitios interconectados de distribución están listos.

Un mecanismo elaborado de *handshake* a través de la red permite a los sitios de distribución coordinar la aceptación de cada transacción.

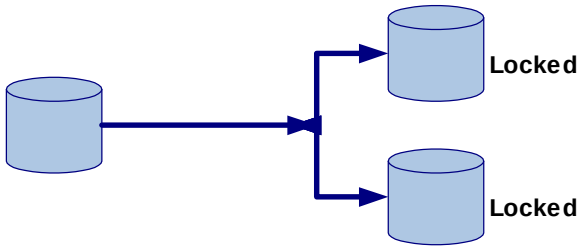


Figura 8: Two-Phase Commit

En la figura una transacción no será aceptada a menos que todos los sitios interconectados estén listos. Consecuentemente, las operaciones de negocios están expuestas a las fallas de los sistemas individuales.

Mientras two phase commit puede ser apropiado donde una corporación necesita absolutamente sincronizar datos distribuidos, éste tiene un precio; todos los sitios distribuidos necesitan estar sincronizados para aprobar una transacción antes de que ésta sea aceptada, la información corporativa está expuesta a comportamientos individuales de sus componentes. Si uno de los sitios no está disponible, la transacción tendrá que esperar. Por lo tanto, el mecanismo elaborado de handshaking con mensajes de ida y vuelta entre los sitios para coordinar la aceptación de datos pone un significante cuidado en las redes corporativas.

Debido a que las conexiones de red y los componentes individuales en un sistema distribuido pueden fallar muchas organizaciones empezaron a buscar maneras prácticas y formas baratas de compartir datos corporativos con menor riesgo de sus operaciones.

Esto deja claro que el tiempo de distribución de copias actuales de datos a través de una organización se debería tomar en cuenta. Con una copia de datos disponible en un sitio las localizaciones individuales no distantes necesitan preocuparse por la disponibilidad de la red o de los sitios remotos. Las mismas que pueden continuar sus operaciones locales con las copias locales de datos. El objetivo es que tos sitios distribuidos necesitan estar disponibles para trabajar con datos que no están sincronizados.

En general se puede concluir que: los cambios son replicados a nivel de fila mediante el mecanismo de TWO PHASE COMMlT.

Las modificaciones de datos deben aplicarse exitosamente en todos los lugares; si en alguno de los sitios no se puede aplicar entonces se genera un rollback de la transacción en todos los sitios. Este tipo de replicación es aconsejable en ambientes de redes estables y en donde los sitios requieran sincronización continua.

No obstante es posible crear ambientes de replicación híbridos en donde alguno de los nodos propaguen sus cambios en forma asincrónica y otros lo hagan en forma sincrónica. Por último se puede afirmar que, la técnica sincrónica asegura la integridad de datos en sacrificio de disponibilidad y rendimiento del tiempo de respuesta. Además no tiene que preocuparse por la resolución de conflictos.

3.7.2.- Replicación Asincrónica

En esta técnica la base de datos destino es actualizada después que la base de datos origen fue actualizada. El tiempo de actualización puede oscilar entre unos pocos segundos a varias horas, depende de la configuración. Sin embargo, los datos están eventualmente sincronizados en el mismo valor en todos los sitios. En un caso particular que falle un sitio o no esté disponible, la replicación asincrónica necesita de requerimientos locales para continuar.

La técnica asincrónica maximiza la disponibilidad y rendimiento de tiempo de respuesta, aunque ésta requiere de planificación y diseño para asegurar integridad de datos y fundamentalmente resolver conflictos de actualización.

3.7.2.1 Diseño de Aplicaciones con Replicación Asincrónica

Antes de escoger la arquitectura de replicación, es necesario entender las necesidades de replicación en la organización. No existe un método actualizado que se pueda seguir para definir los requerimientos, sin embargo se pueden hacer las siguientes consideraciones de carácter general:

- Típicamente, los datos pueden ser descritos en términos de dónde ellos son creados o en términos de estados actuales. Algunos datos no pueden ser caracterizados en términos de su localización, pero sí un porcentaje de ellos en términos de su función. Por lo tanto, el diseño de un ambiente de replicación debería empezar con una investigación de datos.

- Una vez que se ha definido las características de los datos, llega a ser más fácil entender quién necesita acceso a ellos desde las diferentes localizaciones de la organización.

  El administrador debe ser capaz de definir dónde los datos necesitan ser visibles y qué localizaciones pueden actualizar datos.

- Con una matriz de datos y requerimientos de acceso en mano, se puede empezar a planificar el ambiente de replicación. Se debe tomar en cuenta que los requerimientos de acceso a datos desde un sitio no significa que los datos deban estar localizados en dicho sitio. En general, la replicación debería ser utilizada solo si un enlace cliente-servidor a un servidor central tiene un rendimiento inadecuado, o si se quiere una alta disponibilidad. Después de situar los datos en

las localizaciones o sitios en la organización, se estará listo para decidir cuántas localizaciones se desea mantener sincronizadas.

A continuación se describe con mayor detalle los criterios y actividades para diseñar la replicación en aplicaciones de bases de datos.

3.7.2.2 Planificación de la Replicación

Planificar replicación es similar a planificar un diseño de base de datos lógico, debido a que éste es necesario para tener un modelo de trabajo antes que se pueda implementar replicación en forma satisfactoria. Hay varias características que deberían ser tomadas en cuenta. Incluyendo decisión de qué replicar, quién necesita replicación de datos y cómo replicarlos.

Se citan a continuación algunas consideraciones esenciales cuando se planifica replicación:

- Cuando se distribuyen datos, es importante replicar solo los datos necesarios y no toda la base de datos. Es necesario entonces considerar replicar un subconjunto de datos, que es lo que se utilizo en nuestro Sistema.
- Si los sitios son solo de lectura y requieren solo información específica de una base de datos, entonces se puede usar replicación. Si los sitios deben tener capacidades de actualización, probablemente no se debería considerar replicación, ó aún mejor tener un adecuado diseño de detección y resolución de conflictos si se emplea replicación.
- La actualización de la información esta condicionada por las necesidades de la organización. Por ejemplo, empresas de marketing, necesitan tener información lo más actualizada posible.
- El sitio central donde se consolida la información debería ser el candidato a distribuir la información.

3.7.2.3- Determinación del escenario de Replicación

Para determinar el escenario de replicación se debe realizar las siguientes consideraciones:

- Determinar el propósito de la replicación de datos (información corporativa, generación de reportes, sistemas de soporte de decisiones)
- Establecer las conexiones de acceso a la red. Notar que si hay un enlace lento sé beneficiaria de tener servidores separados para cada propósito
- Determinar el número de servidores

Su estimación ayuda a determinar la carga en el servidor y el tamaño apropiado para el log de transacciones de cada base de datos marcado de replicación.

- Estimar la frecuencia de replicación

Esta característica ayuda a determinar el tiempo apropiado para que cada sitio reciba los datos replicados, la estrategia apropiada de backup[9], el tamaño apropiado para la base de datos de distribución y los requerimientos de balanceo de carga

- Determinar el plan de implementación

Esto Determinar lo siguiente:

- El tamaño de la base de datos y logs de transacciones
- Un horario de replicación(frecuencia)

[9] Tèrmino utilizado para describir el proceso de respaldos de una base de datos.

- Si se emplea actualización en otra parte que no sea en la central, diseñar métodos y rutinas de detección y resolución de conflictos si es necesario

3.8 DETECCIÓN Y RESOLUCIÓN DE CONFLICTOS

Los Sistemas Distribuidos y más específicamente las bases de datos distribuidas tienen factores esenciales que determinan su rendimiento en una red de computadoras. Existen varios problemas técnicos que necesitan ser resueltos para obtener una confiabilidad del sistema de base de datos- Entre los cuales se deben tomar en cuenta los siguientes:

- Diseño de base de datos distribuidas.
- Proceso de consultas distribuidas.
- Administración distribuida.
- Control de concurrencia distribuida.
- Administración de Deadlocks distribuidos
- Confiabilidad de los DBMS distribuidos
- Soporte para sistemas abiertos.
- Bases de datos heterogéneas

A la hora de replicación de datos, el aspecto más importante a considerar es el Control de concurrencia. Dicho factor incide en el rendimiento y confiabilidad del proceso de replicación.

El Control de concurrencia involucra la sincronización del acceso a la base de datos distribuida, en orden de mantener la integridad de datos. El Control de concurrencia en un ambiente distribuido es diferente que para los sistemas centralizados. Esto no es solo un problema de integridad de una base de datos, sino que debe tomar en cuenta la consistencia de múltiples copias de la BDD.

Existen numerosas alternativas de solución para este problema. Pero las más difundidas de aquellas son: **pesimista y optimista**. En la alternativa *pesimista*, sincroniza la ejecución de los requerimientos del usuario antes de iniciar la ejecución. En la alternativa optimista, ejecuta el requerimiento y luego chequea si la ejecución compromete la consistencia de la base de datos. Dos primitivas fundamentales pueden ser usadas con las dos alternativas mencionadas anteriormente: lock y timestamping. Lock está basada en la exclusión *mutua* de acceso a ítems de datos.

Con timestamping, las transacciones son ejecutadas en el mismo orden.

Finalmente se puede comentar que existe un tercer tipo de variación que intenta combinar las dos primitivas. Estos son los algoritmos híbridos.

3.8.1 Consideraciones generales de Control de concurrencia

Cuando dos consultas intentan actualizar el mismo ìtem de datos, o si el sistema falla durante la ejecución de una consulta son dos de las principales consideraciones de concurrencia a tener muy en cuenta. Aquí entra un concepto fundamental como es **Transacción**.

Las transacciones son en síntesis la unidad básica de la computación confiable y consistente. La administración de transacciones trata con los problemas de mantener siempre las BDD en un estado consistente aún cuando exista fallas en el sistema.

3.8.1.1 Definición de una transacción

Como ya se mencionó anteriormente una transacción es la unidad básica de una computación confiable y consistente. Una transacción toma una BDD, realiza una acción sobre ella y luego genera una nueva versión de la BDD, generando un estado de transición. En resumen, la transacción se puede pensar como un programa empotrado en la Base de Datos.

Una transacción siempre termina, aún si hay fallas, si una transacción puede completar su tarea satisfactoriamente, se dice que la transacción fue completada(commit). Por otro lado, si una transacción se para sin completar su tarea, se dice que ésta ha abortado(Rollback).

Adicionalmente, los DBMS pueden abortar una transacción debido a dos factores: deadlocks o alguna otra condición. Cuando una transacción es abortada, su ejecución es parada y todos sus efectos de acción ya ejecutados son deshechos(undone) mediante el retomo de la BDD al estado antes de esta ejecución. Esto es conocido también como rollback.

3.8.1.2- Propiedades de las transacciones

Los aspectos de consistencia y confiabilidad de transacciones son debido a las siguientes cuatro propiedades:

- Atomicidad.- Se refiere en general que una transacción es tratada como una sola unidad de operación. Por lo tanto, o toda la transacción es completa o nada de ella.
- Consistencia.- La consistencia de una transacción es simplemente su correctitud.

En otras palabras, una transacción es un programa correcto que mapea una BDD de un estado consistente a otro.

- Aislamiento.- Propiedad con la cual cada transacción quiere ver la BDD consistente todo el tiempo
- Durabilidad.- Se refiere a que se asegura que las transacciones son commited(completas) solo una vez, sus resultados son permanentes y no pueden ser borrados de la BDD.
- Recuperación.- Cuando ocurre una falla, un DBMS debe mantener alguna información sobre el estado en el tiempo de la falla en orden de traer a la BDD al estado que ésta tenía antes que la falla ocurriera.

3.8.2. Mecanismos de Control de concurrencia

Dos grupos de mecanismos serán discutidos en esta parte: métodos de control de concurrencia pesimista y métodos de control de concurrencia optimista. Los Algoritmos Pesimistas sincronizan la ejecución concurrente de transacciones inicialmente en su ciclo de vida. Mientras que los algoritmos optimistas demoran la sincronización de las transacciones hasta su terminación.

Los dos grupos consisten de algoritmos basados en lock y de algoritmos basados en timestamp.

En la aproximación basada en lock la sincronización de transacciones es llevada a cabo mediante locks(seguros) tanto físicos como lógicos en alguna porción o gránulo de la BDD. El tamaño de esa porción (usualmente llamada locking granuly) es una característica importante.

La clase timestamp ordering involucra el orden de ejecución que las transacciones deben mantener mutua interconsistencia. Este orden es mantenido mediante la asignación de tiempos tanto para las transacciones como para los ítems de datos que están almacenados en la BDD.

3.8.3.- Conflictos en la modificación de datos

Sobre todo en la Replicación Asincrónica la combinación de requerimientos de acceso y la localización de objetos-datos ayudarán a definir los potenciales conflictos en modificación de datos. Si se utiliza la propagación asincrónica en las actividades de la organización muchos conflictos pueden ocurrir. Esto pasa cuando usuarios en diferentes localizaciones actualizan una misma fila a diferentes valores.

Se cita a continuación una lista de potenciales tipos de conflictos y cuando ellos ocurren:

Cuando se crea una fila

Los valores para la fila resultante en una violación de unicidad.

Otra fila tiene la misma clave primaria.

Cuando se remueve una fila

La fila ya ha sido removida

La fila fue cambiada después de ser removida desde la localización original

Cuando se actualiza una fila

La fila anterior fue removida

La fila anterior cambió su valor

El nuevo valor de fila resulta en una violación de unicidad

3.8.4.-Cuándo utilizar métodos de resolución de conflictos

Los propósitos de la resolución de conflictos son:

- Asegurar convergencia de datos.
- Evitar errores en cascada.

En una transmisión asincrónica es necesario tomar en cuenta.

- Monitorear la concurrencia de algunos conflictos no resueltos.
- Usar alguna flotación para aprender de algún conflicto inesperado.

3.8.5.- Detección de Conflictos

Cuando se propaga cambios en un ambiente de replicación puede ocurrir varios conflictos. Por lo tanto, la persona encargada de administrar replicación debe estar preparada con métodos que permitan su detección y su resolución también. Cada motor de base de datos tiene sus métodos de detección y resolución.

3.8.5.1.- Tipos de Conflictos

De acuerdo a las fuentes de información tres errores pueden ocurrir los cuales son:

- Conflictos de actualización
- Conflictos de unicidad
- Conflictos de eliminación

Un **conflicto de actualización** es detectado cuando hay alguna diferencia entre el valor anterior de la fila replicada y el valor actual en la misma fila en el sitio que recibe los datos.

Un **conflicto de unicidad** es detectado cuando una restricción única es violada en las operaciones insert o update sobre una fila replicada.

Un **conflicto de eliminación** es detectado si una fila cambia en un sitio remoto después que se ha borrado dicha fila del sitio local. Este tipo de error ocurre debido a que el valor anterior de la fila borrada en el sitio local no hace un match(no se ajusta) con el valor actual de la misma fila en el sitio remoto.

Para finalizar es importante citar lo siguiente; debido a que la clave primaria es usada para determinar qué fila comparar, hacer modificaciones sobre la clave primaria puede ser riesgoso para la integridad de los datos.

3.8.5.2.Tipos de Errores

Cada vendedor de Base de Datos tiene sus propias técnicas para detectar conflictos. Los tipos de errores que se ha podido citar de acuerdo a la investigación realizada son los siguientes:

- Errores de diseño.- Ocurre cuando un cambio en el diseño maestro hace conflicto con un cambio de diseño en la réplica. Las fallas de sincronización y el contenido de las diferentes réplicas empiezan a divergir

- Conflictos de sincronización.- Ocurre cuando los usuarios actualizan el mismo registro en dos réplicas. La sincronización es exitosa, pero los cambios de una sola tabla son replicadas en las demás.
- Errores de sincronización.- Ocurre cuando un cambio en los datos en una réplica no puede ser aplicados a otra réplica debido a que se violaría una restricción, como por ejemplo la integridad referencial. La sincronización es exitosa, pero el contenido de las réplicas son diferentes.

Al finalizar, los errores de sincronización y de diseño son problemas más significativos que los conflictos de sincronización.

3.8.6.- Consideraciones de diseño para evitar conflictos

Las aplicaciones pueden ser diseñadas para evitar la presentación de conflictos mediante el particionamiento de datos, técnica de muy amplia utilización En general, existen algunas maneras para tratar los potenciales conflictos de actualización: particionamiento de datos, workflow, token passing e instrucciones para resolución de conflictos.

3.8.6.1.- Particionamiento de datos.-

Es una técnica para evitar conflictos donde los subconjuntos de datos son permitidos de actualizar en una localización solamente. A menudo se le conoce a esta técnica como modelo Propietario Sitio primario estático. Bajo esta técnica se elimina algunos potenciales conflictos, requiere de cambios de aplicación(todas las aplicaciones deben registrarse(log) en la localización propietaria para modificar datos. Además, no ofrece una alta disponibilidad para modificación de datos(todas las actualizaciones deben ser procesadas en una sola localización). Introduce cuellos de botella para todas las actualizaciones que pasan a través de la localización propietaria.

3.8.6.2.- Workflow

Es otra técnica para evitar conflictos la cual asegura que un dato puede ser modificado en una sola localización en un instante dado. Esta técnica se diferencia de la anterior en que la localizaci6n propietaria de datos puede moverse de un sitio a otro. A veces esta técnica es llamada modelo propietario Sitio Dinámico. Mientras esta técnica elimina potenciales conflictos ésta requiere de *cambios* de aplicación al igual que la anterior.

3.8.6.3.- Token Passing

Es una técnica para evitar conflictos donde cada ítem de dato contiene un token propietario. Las localizaciones pueden difundir sus requerimientos para ser propietaria de un ítem de dato y debe estar completamente de acuerdo con las otras localizaciones antes de obtener el token del **ítem** de dato. Este método elimina algunos potenciales conflictos, pero requiere de una difusión asincrónica de requerimientos y acuerdos con los propietarios de los ítems de datos.

3.8.7 Resolución de conflictos.-

Permite a cada BDD decidir que hacer en caso de una emergencia ante un conflicto. Esto se lleva a cabo mediante un adecuado manejo y un consistente manual de procedimiento en cada localización. Cuando un conflicto ocurre en una localización, el sistema es lo suficientemente inteligente para buscar que hacer en este playbook (manual de procedimiento) y proceder de manera correcta. Debido a que todos los sistemas utilizan el mismo conjunto de instrucciones, todos los datos convergen. No se requieren comunicación entre los sistemas para llevar a cabo una resolución de conflictos.

3.9.- Distribución VS. Replicación?

Los datos son distribuidos por las siguientes razones:

Mejora en el rendimiento.- Cada servidor es administrado sólo por los datos asociados con ese servidor. Las tablas son las más compactas posibles.

Reducción en el costo.- Muchos servidores pequeños pueden ser menos costosos y más baratos de mantener que un servidor monolítico. Un solo servidor representa un sólo punto de falla. Si un servidor falla en un escenario de múltiples servidores, los otros servidores pueden continuar sirviendo a sus sitios.

Los datos son replicados por las siguientes razones:

Para soportar distribución de otros datos.- Todo el tiempo las tabas de datos distribuidos contendrán claves de tablas replicadas. Para mantener la integridad referencial, las tablas de soporte deben ser replicadas a los sitios de distribución de datos.

Para permitir accesibilidad a un mismo dato.- Por ejemplo, este puede ser

Deseable para replicar un resumen de información transaccional en cada servidor de tal manera que las estadísticas puedan ser generadas por otros servidores.

3.10 Aplicaciones

A continuación se describe brevemente las aplicaciones que utilizan los beneficios de la Replicación en la Informática Actual.

3.10.1. Sistemas distribuidos

Un sistema distribuido de bases de datos es una colección de localidades, cada una de las cuales mantiene un sistema de base de datos local, con capacidad de procesar tanto transacciones locales como globales. Las transacciones locales son aquellas que solamente accesan datos de dicha localidad, mientras que las globales accesan a datos repartidos en varios lugares y por lo tanto para su ejecución se requiere la comunicación entre los diferentes puntos.

Hay muchos aspectos involucrados en el almacenamiento de información en una base de datos distribuida, incluyendo replicación y fragmentación.

Mediante la replicación, el sistema mantiene varias copias idénticas (réplicas) de la información y cada réplica es almacenada en una localidad diferente.

Mediante la fragmentación, la información es particionada en varios fragmentos, cada uno de los cuales puede ser almacenado en un lugar diferente.

Entendiéndose por fragmento a un subconjunto de columnas o de filas de una tabla.

En general, se puede decir que un servidor de replicación puede mantener copias de tablas en múltiples bases de datos que pueden estar geográficamente distribuidas.

En un sistema de replicación, una de las copias de la tabla es designada como la copia o versión primaria. Todas las demás son copias replicadas o secundarias.

Las copias replicadas proveen dos ventajas comparadas con el acceso directo sobre una red de área extendida (WAN: Wide Area Network) a los usuarios en las diferentes localizaciones de las bases de datos: respuesta más rápida.y mayor disponibilidad.

A continuación, se muestra un esquema de replicación utilizado en Sybase Replication Server, el cual muestra la replicación de fragmentos de información entre diferentes sitios:

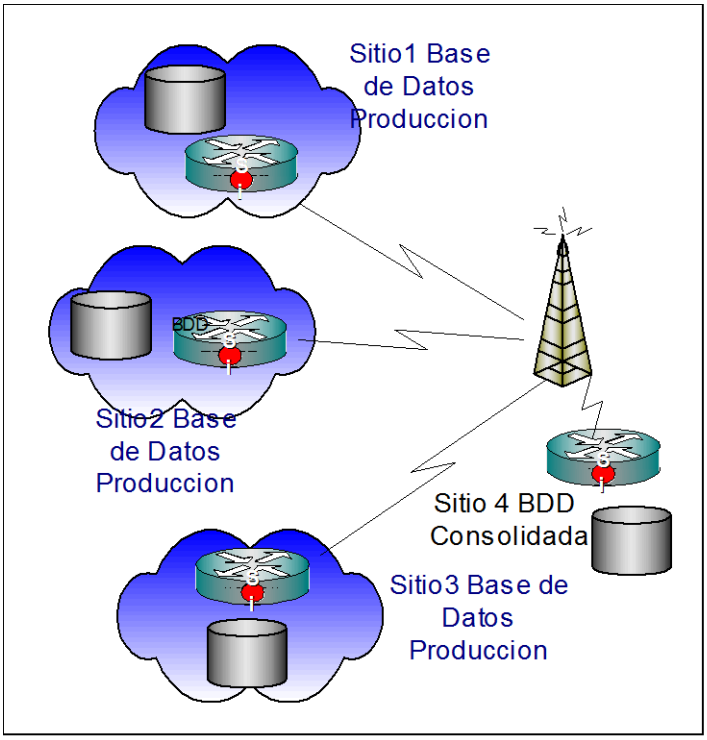
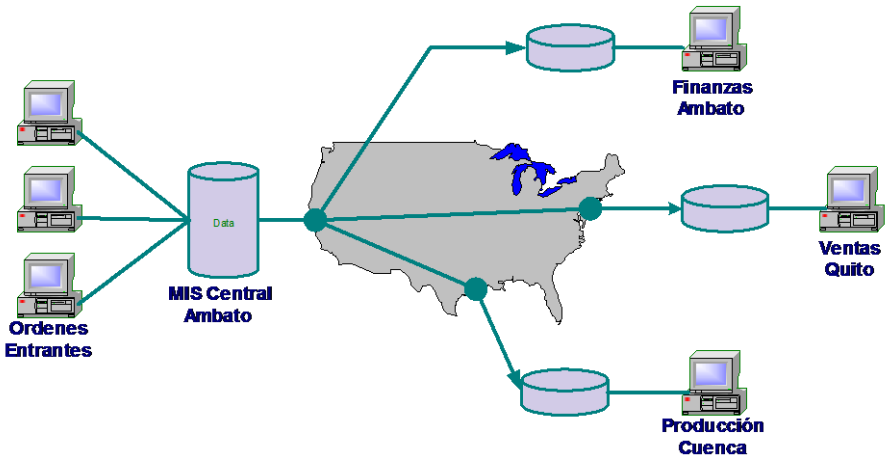


Figura 9: Esquema de replicación en sistemas distribuidos.

A continuación se mencionará ejemplos de como las corporaciones pueden utilizar replicación en sus ambientes distribuidos cliente-servidor.

Ejemplo 1: Un nodo primario - Múltiples secundarios (Réplicas para la toma de decisiones)



En su forma más básica, la replicación provee una forma no costosa de distribuir la información desde un nodo centralizado hacia muchos nodos secundarios en donde la información será solamente leída y no será modificada.

A diferencia de los snapshots, los mecanismos de distribución mantienen la integridad transaccional de los datos.

La figura ilustra una compañía que usa una base de datos OLTP (la cual de manera general reside en un mainframe) localizada en Ambato para procesar las órdenes entrantes. Las aplicaciones de entradas de órdenes son conectadas directamente (como terminales o clientes) al sistema OLTP central el cual es controlado por el personal de sistemas de la compañía. Un gran número de personas encargadas de tomar decisiones en otras áreas de la organización, por ejemplo, finanzas que está localizada también en Ambato, producción en Cuenca y ventas en Quito, requerirían visualizar la información de las órdenes para tomar decisiones oportunas en sus respectivas localizaciones.

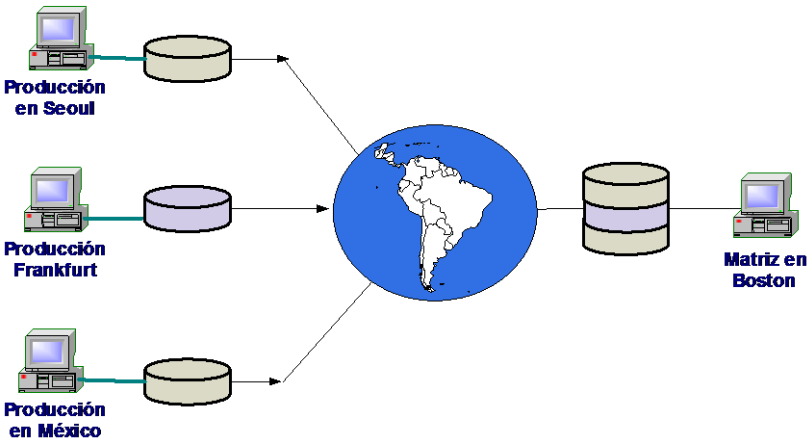
De cualquier manera, debido a la naturaleza variable de las consultas especificas, a los reportes batch personalizados de cada localidad o simplemente debido a su gran número, los nodos no están permitidos a accesar el sistema OLTP. Por lo tanto, las personas encargadas de tomar

decisiones tienen que buscar la información que necesitan en reportes estandarizados generados por los diseñadores del sistema.

Si se utiliza replicación, las localizaciones se pueden "suscribir" a un subconjunto de información del nodo central y visualizar dicha información localmente. Cada sitio puede visualizar el subconjunto de información que es requerido localmente. Con una copia local de la información necesaria para el soporte a decisiones, finanzas, ventas y producción pueden llevar a cabo sus operaciones sin tener que conectarse a la localización central en Ambato. Su sistema de soporte a decisiones es independiente de tallos de la red o de sistemas remotos.

Además, los usuarios locales pueden recibir los resultados de sus consultas más rápidamente, en vista de que se están procesando localmente (en la réplica) en lugar de en un sitio remoto.

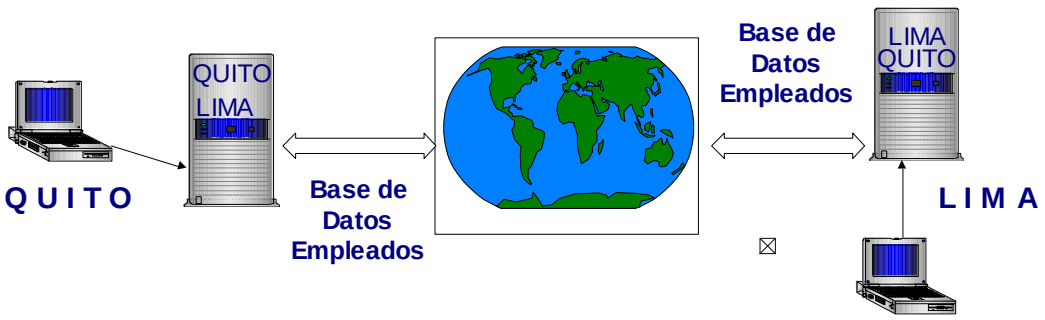
Por consiguiente, la replicación puede proveer una manera simple de incrementar la escalabilidad del sistema, permitiendo un acceso rápido a la información del personal que toma las decisiones de la compañía.



Ejemplo 2: Un nodo secundario.Múltiples nodos primarios -(Consolidación de información corporativa)

Además de distribuir información desde un nodo provee un mecanismo simple para consolidar la varias localidades en un solo sitio.

Un ejemplo para esta configuración es una corporación que recopila información de sitios remotos donde se realiza la producción, con la finalidad de hacer un análisis de los datos.



Ejemplo 3: Configuración punto a punto sin conflictos de actualización. (Compartición de información corporativa.)

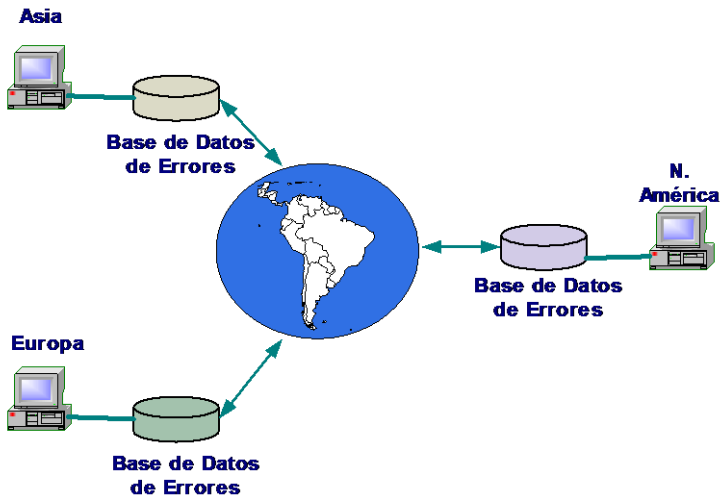
Los dos ejemplos previos describen casos en los cuales la replicación ocurre en una única dirección, en el primer caso de un nodo primario hacia muchos nodos secundarios, y en el segundo ejemplo desde muchos sitios hacia uno,

La replicación no necesariamente implica la transmisión de datos en una única dirección. Una empresa que comparte su información, es un claro ejemplo de replicación en los dos sentidos. Aquí muchos sistemas distribuidos incluyen información primaria e información replicada a la vez. Por ejemplo, una compañía con varias localidades a través de Sudamérica (Quito y Lima) desean consolidar la información de su personal. Cada empleado tiene una y sola una oficina local. Cada oficina maneja localmente la información de sus empleados La replicación provee un mecanismo para que la Corporación comparta el conjunto de información distribuida de empleados.

En vista de que cada empleado tiene solo una oficina local, el ejemplo ilustra los casos en los que distinta información es combinada de cada nodo y la aplicación distribuida no presenta conflictos de actualización.

En vista de que cada localidad tiene información tanto primaria como replicada, no se permite que dos nodos modifiquen la información del mismo empleado.

Ello implica que los empleados de Quito, únicamente pueden modificar los datos de la oficina de Quito. Lo mismo sucede con la información de Lima respectivamente. El resultado final es que todos los nodos pueden visualizar colectivamente casi en tiempo real la información de los empleados de toda la corporación.



Ejemplo 4: Configuración punto a punto con (Integración de la información corporativa)

El ejemplo previo se enfoca en la combinación de información diferente de diversos sitios en donde dos localidades diferentes no pueden modificar la misma parte de la información. Mientras es mucho más simple el manejar y diseñar una aplicación que evite conflictos de actualización este tipo de diseños no es siempre posible. En muchas situaciones, los procesos de negocios requieren que más de un nodo modifique la información común.

Una compañía multinacional de software tiene analistas de apoyo en tres sitios diferentes: uno en Asia, uno en Europa y uno en América del Norte. Los analistas de soporte responden a las llamadas y solicitudes de los clientes de cada región. Todos los analistas comparten una base de datos de errores común en donde la información de los problemas de los clientes son registrados La base de datos necesita ser accedida por los tres nodos. Además los analistas en las tres localidades requieren ingresar y modificar la información en la base de datos de errores. Este es el caso en que los datos pueden ser actualizados por analistas de cualquiera de los nodos.

En vista de que más de un sitio puede actualizar la misma información al mismo tiempo, las aplicaciones de estas organizaciones necesitan incluir un mecanismo de resolución de conflictos de actualización.

3.10.2. Aplicaciones warm standby

Una aplicación *warm standby*[10] consiste de un par de bases de datos, una de las cuales funciona como una copia standby de la otra, entendiéndose por standby a una copia de la base de datos que está lista para ser utilizada cuando se lo requiera, por ejemplo en el caso de una caída del sistema, después de la cual se necesite recuperar la base original. Este tipo de aplicaciones se utilizan básicamente como un mecanismo de respaldo de la información.

La base de datos standby es una copia de la base de datos activa y las aplicaciones cliente se encargan de actualizar la base de datos activa.

El proceso de replicación mantiene la base de datos standby consistente con la base de datos activa por medio de la propagación de transacciones recibidas por el log de transacciones de la base de datos activa.

Si la base de datos activa falla, el cliente se puede conectar a la base de datos standby a fin de que las aplicaciones puedan continuar ejecutándose con una mínima interrupción.

Las dos bases de datos en una aplicación warm standby aparecen como una sola base de datos lógica en el sistema de replicación. Dependiendo de su aplicación, esta base de datos lógica puede o no participar en replicación o puede ser una base de datos primaria o una base replicada con respecto a otras bases de datos en el sistema de replicación.

Ciertos servidores de replicación soportan aplicaciones *warm standby*, por ejemplo, Replication Server de Sybase.

[10] "Espera en línea", tèrmino utilizado para describir configuraciones replicación con alta disponibilidad.

En una aplicación warm standby se crean tres conexiones:

- Una conexión lógica que el servidor de replicación mapea a la base de datos activa al momento.

  ***** Una conexión física con la base de datos activa.

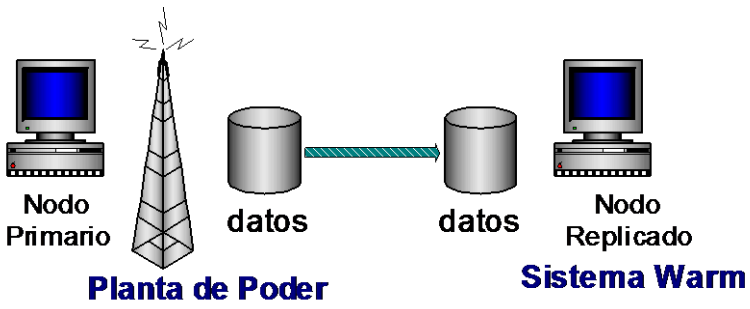
  ***** Una conexión física con la base de datos *standby*.

Existe otro tipo de aplicaciones denominadas hot standby, en las cuales la base de datos standby puede ser puesta en servidor sin interrumpir las aplicaciones en el cliente y sin perder ninguna transacción, a diferencia de las aplicaciones warm standby en las que si hay una mínima interrupción.

La siguiente figura muestra el caso de una gran empresa eléctrica que tiene un sistema de base de datos en la planta central, que controla la producción y la transmisión de energía desde la planta industrial a los hogares de cada cliente. La aplicación que controla la transmisión de energía tiene claramente una misión crítica, en la que se requiere una arquitectura redundante, esto significa que la información se copia a múltiples discos en la misma localidad, las conexiones de red son redundantes, etc. Lastimosamente existe un problema:

De suceder una catástrofe mayor, como un terremoto, inundación, incendio, etc. en la planta de energía, el sistema que controla los switches de poder y las líneas de transmisión sufrirá daños por lo que será imposible que el sistema de base de datos siga funcionando.

La arquitectura redundante de la compañía requiere que un sistema standby se mantenga fuera del área, por si ocurre un evento catastrófico en la localidad de la planta primaria. De esta manera los administradores podrán direccionar las aplicaciones al nodo remoto que contiene la información redundante.



En el caso de la planta eléctrica, ésta se beneficiará por tener la opción de controlar las líneas de transmisión de energía desde una localización remota a la planta de energía primaria.

3.10.3. Aplicaciones Data Warehouse

Literalmente las palabras que forman este término tienen las siguientes definiciones:

•*Data:* hechos e información acerca de algo.

•*Warehouse:* un lugar o sitio para almacenar bienes y mercaderías.

Para Tom Hammergren, *data warehouse* se puede definir de las siguientes maneras:

"Es un lugar que almacena información objetiva o basada en hechos, con la finalidad de realizar análisis o tomar decisiones.

"Es una base de conocimientos corporativa."

Según el mismo autor, un sistema de *data warehouse* tiene las siguientes características:

- Provee centralización de los datos corporativos.
- Se almacena en un ambiente administrado adecuadamente.
- Tiene procesos periódicos y consistentes definidos para cargar los datos operacionales.
- Se construye utilizando arquitecturas abiertas y escalables que soportan el incremento de la información con el paso del tiempo.

Se dice que un *data warehouse* está orientado a hechos, porque se orienta a las principales áreas de la corporación que han sido definidas en el modelo de datos. Estas áreas finalmente son implementadas como una serie de tablas relacionadas en el *data warehouse.*

Por ejemplo, un sistema operacional típicamente se encuentra organizado tomando en cuenta los campos de acción de la empresa, que en el caso de una compañía de seguros serían seguros de vida, salud, accidentes y bienes, mientras que un *data warehouse*, considera las principales áreas que serian clientes, políticas, reclamos y primas.

Un sistema operacional es aquel que maneja las operaciones diarias de una empresa y generalmente se ejecuta en un ambiente OLTP

Se dice que un *data warehouse* es integrado, porque realiza una integración de sus datos al pasar de un ambiente orientado a operaciones a un *data warehouse.*

La tercera característica de *data warehouse* es la no volatilidad. La información en un sistema operacional es generalmente accesada actualizada y manipulada de registro en registro. Por el contrario, los datos en las aplicaciones *data warehouse* se cargan en masa y se accesan, pero en general, no pueden ser actualizados.

Ocurre cuando se añaden entidades nuevas a un *datawarehouse* o se recupera un componente después de una caída del sistema. Puede ser la única manera de transmitir datos operacionales a un datawarehouse. La última característica de *datawarehouse* es la capacidad de variar en el tiempo. La relación que tiene *datawarehouse* con el tiempo se puede explicar de varias formas:

- El horizonte en el tiempo para *datawarehouse* es significativamente más largo que para sistemas operacionales. Un horizonte de 60 a 90 días es normal para sistemas operacionales, mientras que para *datawarehouse* el horizonte promedio está entre 5 y 10 años.
- Las bases de datos operacionales contienen datos actuales, cuya exactitud es válida en el momento que son accesados. Por lo tanto, los valores de dichos datos pueden actualizarse.

Por el contrario, la información contenida en un *datawarehouse* no es más que una sofisticada serie de *snapshots* tomadas en cierto instante de tiempo.

La estructura de claves de un sistema operacional puede o no contener algún elemento de tiempo, tal como, años, meses, días, etc. Por el contrario, la estructura de claves en un *data warehouse* siempre contiene un elemento de tiempo.

Una vez que un *datawarehouse* ha sido construido, la atención se debe centrar en las operaciones que se realizan día a día en él. Una de las operaciones más importantes es la renovación de la información. La replicación juega allí un papel importante ya que evita el realizar la lectura secuencial de tablas completas de información en el sistema original a fin de detectar las modificaciones realizadas y así actualizar los datos en el *datawarehouse.* Mediante replicación se toman solamente los datos que han sido modificados en el sistema legacy y se los envía al *datawarehouse* para que posteriormente se procesen sin necesidad de estar en línea con el DBMS original.

La replicación requiere que los datos que van a ser captados sean identificados antes de la actualización y con este propósito se configura un trigger.

Una de las ventajas de la replicación es que el proceso de captura puede ser controlado en forma selectiva. Otra ventaja es que el formato de los datos se define en forma clara y correcta, lo que hace que sean comprensibles. Como desventajas se pueden anotar que se requieren de muchas operaciones de lectura/escritura y que el sistema necesita de constante atención a los parámetros y triggers que controlan el proceso de captura de los datos, debido a la naturaleza inestable y variable del *datawarehouse.*

3.10.4. Aplicaciones Workflow.

El proceso de replicación se lo utiliza en el caso de requerirse la implementación de "workflow transaccional" .Es precisamente cuando se requiere la comunicación entre las bases textuales y las bases relacionales que se utiliza un proceso de replicación.

3.10.5 Aplicaciones de computación móvil.

En la actualidad, la computación móvil se puede utilizar generalmente en equipos portátiles para trabajar fuera del sitio habitual de trabajo, ya sea en casa o durante un viaje. Por ejemplo, si un empleado se encuentra en una oficina regional, desde un computador y a través de un módem, podría llamar a un servidor de la oficina central. O simplemente a través de un celular wap o un dispositivo palm se conecte a la central y envíe sus datos.

Cuando una persona se encuentra fuera de la oficina podrá trabajar de dos formas:

- Interactivamente, esto es, realizando una llamada al servidor y permaneciendo conectado mientras trabaja directamente con las bases de datos situadas en él.
- Localmente, para esto el usuario estando conectado al servidor (vía módem o red) debe haber replicado las bases de datos situadas en el servidor a su estación de trabajo. Luego, sin necesidad de estar conectado al servidor, el usuario puede trabajar en las réplicas locales. Posteriormente, puede llamar al servidor e intercambiar la información entre las réplicas locales y las bases de datos ubicadas en él.

Por ejemplo, en Lotus Notes, que además tiene correo electrónico, se tienen las siguientes ventajas:

- Al trabajar en forma interactiva, el correo se entrega inmediatamente y si la dirección de un mensaje es incorrecta, se recibe una notificación al respecto en ese instante.
- Al trabajar en forma local, se puede enviar al servidor de una sola vez, todos los mensajes de correo creados.

Otro ejemplo es lo que hoy en día se llama Soluciones Ultralite implementadas tanto por Sybase como por Oracle. En las cuales se programa en un dispositivo móvil (celular wap, palm, pocket pc, etc) y la información luego es replicada a un repositorio central.

Una aplicación real es un cotizador de seguros donde el empleado visita al Cliente le pide sus datos son almacenados en el dispositivo móvil y luego replicados al servidor central

3.11.- Técnica Usada de Replicación para el Sistema (Informix)

La técnica de replicación usada para el sistema es asincrónica debido a que es un modelo de consolidación de la información. Todo este esquema fue implementado en Informix Dynamic Server.A continuación describiremos como IDS2000 utiliza la replicación y sus métodos de resolución de conflictos.

3.11.1 Replicación en INFORMIX

INFORMIX basa su estrategia en lo que ésta denomina solución avanzada en replicación: Continous data replication(CDR). CDR está construida bajo la arquitectura denominada *Dynamic Scalable Architecture*(DSA)[11]. Bajo ésta técnica INFORMIX basa su implementación de replicación en todas las ventajas que ofrece las arquitecturas del procesamiento en paralelo para satisfacer la mayor cantidad de demandas de las organizaciones. La replicación asincrónica es una característica fundamental en el CDR. La protección de la integridad de transacciones es cubierta bajo una manera transaccional de trabajo que permite el CDR el cual además, provee de un amplio número de opciones para la detección y solución de conflictos. Una gran característica es la de permitir definir a las organizaciones su propio medio ambiente de replicación basados en requerimientos específicos de la organización. Los modelos con los que trabaja CDR son: maestrolesclavo, workflow updates y update anyware.

Los principales componentes son los siguientes:

a) Configuración y control

b) Captura de transacción

c) Colas de transacciones y reparto de las mismas

d) Inserción de transacciones en los servidores destinos

e) Detección y solución de conflictos

Configuración y control

CDR provee un método flexible para configurar las definiciones de replicación que asegura la consistencia en la captura de transacciones y de su transmisión también. La replicación puede ser definida a nivel de tabla, un subconjunto de éstas, particionamiento o vistas. Provee de la característica denominada grupo de replicación que permite definir replicaciones que pueden ser agrupadas. Esta característica simplifica la administración y son más fácil de manejar. Además todos los comandos a nivel de tabla son aplicados a nivel de grupo también. Para finalizar, se puede afirmar que ésta característica permite al CDR tener mayor velocidad de proceso en las bases de datos destino mediante la aplicación de transacciones en paralelo que participan en diferentes grupos de replicación.

[11] Arquitectura de desarrollo de motores de base de datos, basada en criterios de alta disponibilidad.

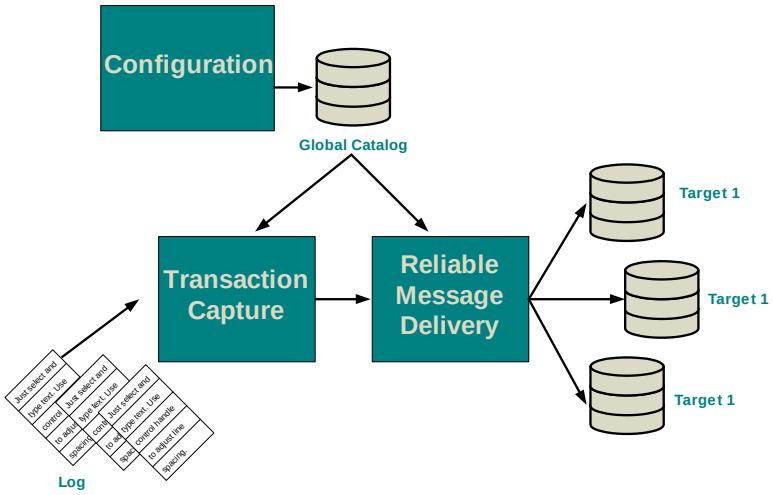


Figura 10: Configuración de Replicación

Fuente: Replication with Informix Dynamic Server 2000

Captura de transacciones

Para capturar transacciones CDR emplea un método de procesamiento de transacciones basado en log. El post-proceso es una ventaja en rendimiento en la minimización de impacto en procesos de transacciones. En primera instancia CDR lee el archivo log para los registros que son destinados para replicación en vez de el archivo entero de log. Los registros participantes son luego filtrados para determinar si ellos necesitan replicación. Los registros seleccionados para replicación son coleccionados en un grupo de transacción.

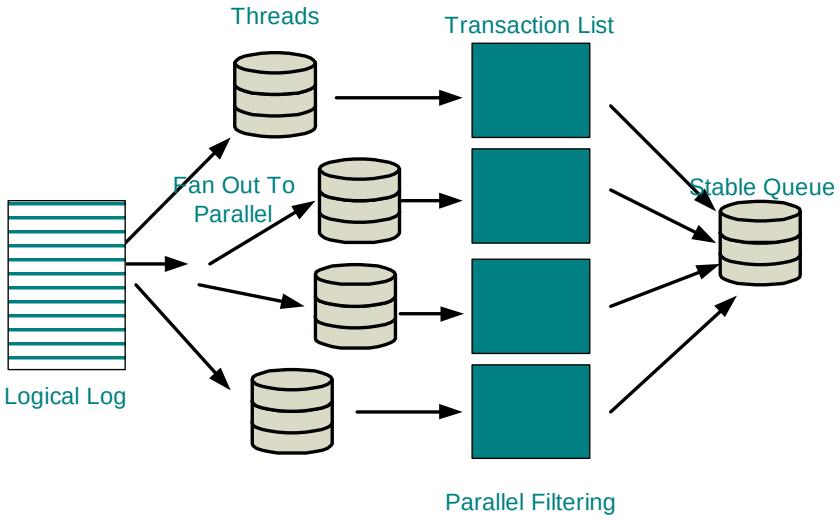


Figura 11: Captura de transacciones

Fuente: Replication with Informix Dynamic Server 2000

Cola de transacciones y Transmisión

Cuando una transacción replicada es completa(commited), la transacción es localizada en una cola estable para transmisión. Las transacciones abortadas son descartadas En el caso que exista una falla en la red la cola estable mantiene las transacciones hasta que la red es restablecida. Para asegurar la integridad, la orden commit es mantenida durante la transmisión.

Una vez que el sitio destino recibe los cambios un mensaje es enviado al servidor origen para asegurar un transmisión confiable.

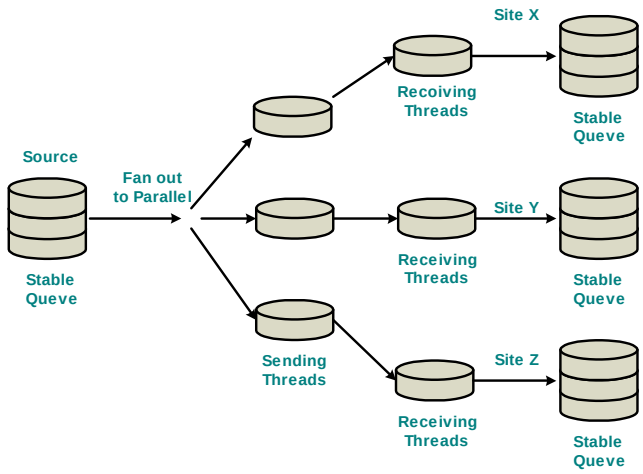


Figura 12: Cola de Transacciones

Fuente: Replication with Informix Dynamic Server 2000

Inserción de transacciones en el servidor destino

Los grupos de transacciones son movidos desde la cola estable e insertados en la tabla apropiada como transacción de la base de datos. Para mejorar el rendimiento esta operación soporta la opción de inserción en paralelo. Si un registro replicado tiene la misma clave primaria que un registro existente en una tabla destino, las reglas preconfiguradas de detección y resolución de actualizaciones es aplicada para determinar qué registro prevalece. Todas las configuraciones necesarias son almacenadas en el catálogo global.

3.11.2 Detección de Conflictos en Informix

En el modo de replicación soporta dos reglas de resolución de conflictos para cada réplica. Se necesita definir una regla primaria y una secundaria. A continuación se lista una tabla de combinación de reglas.

Regla Primaria	Regla Secundaria
Timestamp	No
Timestamp	Store Procedure
Store Procedure	Ninguna
Ignorar	Ninguna

INFORMIX define las reglas de resolución de conflictos y el ambiente de manejo de errores mediante:

- Reglas de Resolución de conflictos.
  - Ignore
  - Timestamp
  - Stored Procedure
- Alcance de resolución de conflictos
  - Fila a fila
  - Transaction

Cada regla de resolución de conflictos se comporta diferente, depende del alcance de resolución que se escoja. A continuación se describe el alcance las resoluciones de conflictos citados anteriormente.

Alcance Fila a fila

Se evalúa una sola fila al a vez. Sólo aplica para las filas replicadas que obtiene una regla de resolución con la fila destino. Además, si una transacción replicada recibe una evaluación fila a fila, algunas filas replicadas son aplicadas y otras no pueden ser aplicadas tal vez.

Alcance de la transacción

Esta es aplicada a una transacción completa solo si la transacción replicada obtiene una regla de resolución de conflicto. De modo contrario, si la transacción destino obtiene la regla de resolución de conflicto (u otro ocurre en la BDD), la transacción replicada no es aplicada y un rollback es aplicado. Además, ésta garantiza la integridad transaccional.

lmplementaciones de Solución.-

La replicación continua de datos provee facilidades integradas de detección y resolución de conflictos, las mismas que invocan a rutinas específicas para asegurar la consistencia de datos. Cuando un conflicto es detectado las siguientes rutinas son aplicadas:

- Latest timestamp.- Da el registro con la última prioridad en tiempo
- Métodos definidos por el usuario(User-defines methods).- Permite a los administradores desarrollar stored procedures específicos para determinar tal o cual prioridad.
- Obtener resolución manual (Hoid for manual resolution). Poner los registros en conflicto en un archivo preconfigurado log con el cual el DBA puede revisar manualmente y determinar la prioridad.
- Ignorar.- Usada para deshabilitar la detección y resolución de conflictos, la cual es muy usada para configuraciones maestro/esclavo.

CAPITULO lV

4. ANÁLISIS Y DEFINICIÓN DE LA ARQUITECTURA DEL SISTEMA

A continuación se realiza una descripción de la metodología que se utilizó para la implementación del Sistema de Administración Académica ACADUTN, así como también de las herramientas analizadas (Manejadores de Base de datos, Herramientas CASE, Front-Ends) para la construcción del software en sí.

4.1 . Metodología Orientada a Objetos

En primer lugar es importante describir los conceptos de Método y Metodología.

4.1.1 Método.-

"Conjunto de Reglas que deben seguirse para el estudio de un arte o ciencia"[12]

4.1.2. Metodología.-

"Estudio formal de los procedimientos utilizados en la adquisición o exposición del conocimiento científico. Es la parte de la lógica que aplica los procedimientos utilizados en el estudio del pensamiento en general a la esfera del pensamiento científico".[13]

4.2 Diseño Orientado a Objetos

La TI ha ido evolucionando en el desarrollo y concepción de proyectos de software desde el modelo Tradicional en Cascada(Metodología Estructurada- Diagrama Conceptual) que partía desde la etapa de análisis, diseño, implementación y pruebas en forma secuencial, sin poder pasar de etapa a otra sin antes haber concluido la anterior, hasta finalmente la construcción de Software basada en Orientación a Objetos que tiene los siguientes componentes:

[12] Tomado del Diccionario Enciclopédico Salvat Tomo XII, Edición 1985
[13] Tomado del Diccionario Enciclopédico Salvat Tomo XII, Edición 1985

- Conceptos y diagramas
- Etapas y definición de entregas en cada una de ellas
- Actividades y recomendaciones

Cuyo principal expositor y su respectiva metodología, la cual sirvió de base para la construcción de ACADUTN se describen a continuación:

4.2.1 Diseño Orientado a Objetos por Grady Booch

4.2.1.1 Conceptos y Diagramas

La metodología de Booch usa los siguientes tipos de diagramas para describir las decisiones de análisis, diseño, tácticas y estrategias, que deben ser utilizadas en la creación de un sistema orientado por objetos:

1. **Diagrama de Clases**. Consiste en un conjunto de clases y relaciones entre ellas. Puede contener clases, clases paramétricas, utilidades y metaclases. Los tipos de relaciones son asociaciones, contenencia, herencia, uso, instanciación y metaclase.
2. **Diagrama de Especificación de Clases**. Es usado para capturar toda la información importante acerca de una clase en formato texto.
3. **Diagrama de Categorías**. Muestra clases agrupadas lógicamente bajo varias categorías
4. **Diagramas de Objetos**. Muestra objetos en el sistema y su relación lógica. Pueden ser diagramas de escenario, donde se muestra cómo colaboran los objetos en cierta operación; o diagramas de instancia, que muestra la existencia de los objetos y las relaciones estructurales entre ellos.
5. **Diagramas de Tiempo**. Aumenta un diagrama de objetos con información acerca de eventos externos y tiempo de llegada de los mensajes.

6. **Diagramas de módulos**. Muestra la localización de objetos y clases en módulos del diseño físico de un sistema. Un diagrama de módulos representa parte o la totalidad de la arquitectura de módulos del sistema.

7. **Diagrama de Subsistemas**. Un subsistema es una agrupación de módulos, útil en modelos de gran escala.

8. **Diagramas de procesos**. Muestra la localización de los procesos en los distintos procesadores de un ambiente distribuido.

4.2.1.2 Etapas

Dentro de esta metodología existen diferentes etapas de construcción de software que a su vez generan su propia documentación, es muy importante en la generación de software el proceso de documentación. A continuación se describen cada una de estas etapas:

a) Etapa de Análisis de requerimientos

- **Funciones primarias del sistema**: Principales entradas y salidas del sistema, referencias a políticas, sistemas existentes o procedimientos, etc.

- **Conjunto de mecanismos claves que el sistema debe proveer**: estado de entrada, estado de salida y estados esperados.

b) Etapa de Diseño

- **Descripción de arquitectura**, describe las decisiones más importantes de diseño como son el conjunto de procesos, manejadores de bases de datos, sistemas operativos, lenguajes, etc.

- **Descripciones de prototipo**, que describen las metas y contenido de las implementaciones sucesivas de prototipos, su proceso de desarrollo y la forma de probar requerimientos.

- **Diagramas de clases en diseño**, detallan las abstracciones de análisis con características de implementación.

- **Diagramas de objetos en diseño**, muestran las operaciones necesarias para desarrollar un proceso.

- **Especificaciones de clases corregidas**, muestra la especificación completa de los métodos con algoritmos complicados, la implementación de relaciones y el tipo de atributos.

4.2.1.3 Actividades

Las principales actividades a realizar dentro de esta metodología son las siguientes:

a)Análisis de requerimientos

En esta etapa se define qué quiere el usuario del sistema. Es una etapa de alto nivel que identifica las funciones principales del mismo, el alcance del modelamiento del mundo y documenta los procesos principales y las políticas que el sistema va a soportar. No se definen pasos formales, ya que éstos dependen de qué tan nuevo es el proyecto, la disponibilidad de expertos y usuarios y la disponibilidad de documentos adicionales Tomando como base la información existente en la Escuela de Ingeniería en Sistemas se procedió a la toma de requisitos, adicional se tomó la experiencia del anterior sistema de control de notas ESCOSOFT, la documentación para la toma de requisitos se lo realiza utilizando diagramas de casos de uso.

b) Análisis de Dominio

Proceso de definir de una manera concisa, precisa y orientada a objetos la parte del modelo del mundo que engloba al sistema. Las siguientes actividades son parte de esta etapa:

Definición de Clases

Definir relaciones de contenencia

Encontrar atributos

Definir herencia

Definir operaciones

Validar e iterar sobre el modelo

Diseño

Es el proceso de determinar una implementación efectiva y eficiente que realice las funciones y tenga la información del análisis de dominio.

Las siguientes actividades se plantean en esta etapa :

-Determinar la arquitectura inicial: decisiones acerca de recursos de implementación, categorías y prototipos a desarrollar.

-Determinar el diseño lógico: detallar el diagrama de clases

-Implementación física: interfaz a dispositivos o características propias de la implementación

-Refinamiento del diseño: Incorporar el aprendizaje debido a los prototipos y cumplir con requerimientos de desempeño.

Tradicionalmente todos los desarrollos y documentación de software se lo realizaba en base a diagramas entidad/relación(diseño conceptual), hoy en día debido al incremento de metodologías OO para construir software, surgieron metodologías y/o estándares que combinan tanto los diseños conceptuales como el orientado a objetos, las cuales permiten a los ingenieros de software construir sistemas de alta calidad .

Una de ellas es la Metodología FOAD "Frame Objet Analysis Diagrams" de Andleigh y Gretzinger la cual pretende combinar la metodología Entidad - Relación con las metodologías de Diseño Orientado por Objetos para diseñar de forma adecuada los sistemas de información modernos.

A través de marcos gráficos de distintos tipos, el diseñador modela simultáneamente las Entidades del sistema de información, sus relaciones y las funciones desde el punto de vista del usuario en forma de jerarquías de menús que determinarán la interfaz gráfica con el usuario. Después de este primer modelaje de Entidades y Funciones, la metodología FOAD contempla su refinamiento a través de nuevos marcos para poder determinar las clases de Objetos que componen el sistema (incluyendo atributos y comportamiento de estos objetos, relaciones de herencia, composición, etc.).

La otra Metodología que es propiedad de OMG es una mezcla de los conceptos de metodologías OO proporcionados por Booch, Yourdon y Jacobson y que en la actualidad se ha convertido en un estándar, y del cual surgió el Unified Modeling Languaje (UML) que es utilizado por todos los CASE orientados a objetos como Rational Rose, Together, Power Designer. En base a este estándar y a la metodología de Booch se implementó ACADUTN.

A continuación se describirá al Lenguaje de Modelamiento Unificado y su utilización en el análisis y diseño de ACADUTN.

4.3 UML

Lenguaje de Modelamiento Unificado (UML - Unified Modeling Language) este es un lenguaje gráfico para visualizar, especificar y documentar cada una de las partes que comprende el desarrollo de software. UML entrega una forma de modelar cosas conceptuales como son procesos de negocio y funciones de sistema, además de cosas concretas como son escribir clases en un lenguaje determinado, esquemas de base de datos y componentes de software reusables.

Para la implementación de este estándar se utilizó inicialmente la herramienta CASE Power Designer 7.0(y luego la versión 8.0), la cual implementa cada uno de los diagramas de orientación a objetos e interactúa directamente con los diagramas entidad – relación lo cual permite utilizar una sola metodología para construir software.

Elementos notacionales de UML

A Continuación se revisan en niveles de complejidad los diversos elementos notacionales que presenta el Unified Modeling Language. Estos elementos pretenden ser un lenguaje común para el modelamiento de cualquier sistema informático.

Esta descripción no pretende ser exhaustiva en términos sintácticos, semánticos y de presentación de los elementos de la notación. Debe entenderse como una guía inicial al tema. Se agrupan los conceptos básicos por tipo de diagrama:

1. Diagrama de Estructura Estática
2. Diagrama de Casos de Uso
3. Diagrama de Secuencia
4. Diagrama de Colaboración
5. Diagrama de Estados
6. Diagrama de Actividades
7. Diagrama de Implementación

A continuación se describen brevemente los tres diagramas de UML utilizados en el Análisis, Diseño e Implementación de ACADUTN.

4.3.1 Conceptos de un diagrama de Casos de Uso

Muestra las distintas operaciones que se esperan de una aplicación o sistema y cómo se relaciona con su entorno (usuarios u otras aplicaciones). Es análogo al diagrama de flujo de datos o DFD que es parte del diseño tradicional, su función básica es la documentación para la toma de requisitos funcionales de un sistema. Es el primer diagrama que se debe construir en UML

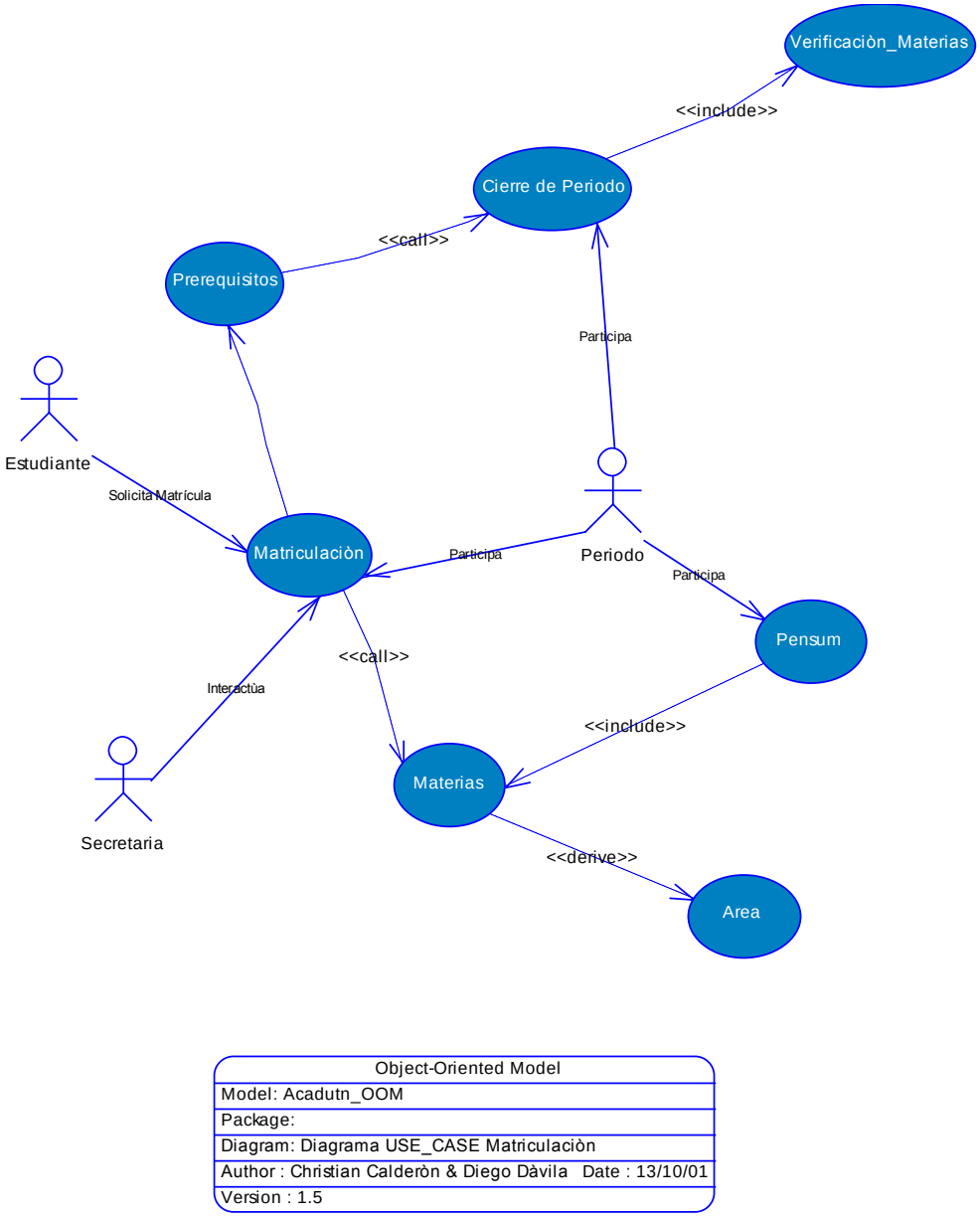


Figura 14: Diagrama de Casos de Uso para el proceso de Matriculación

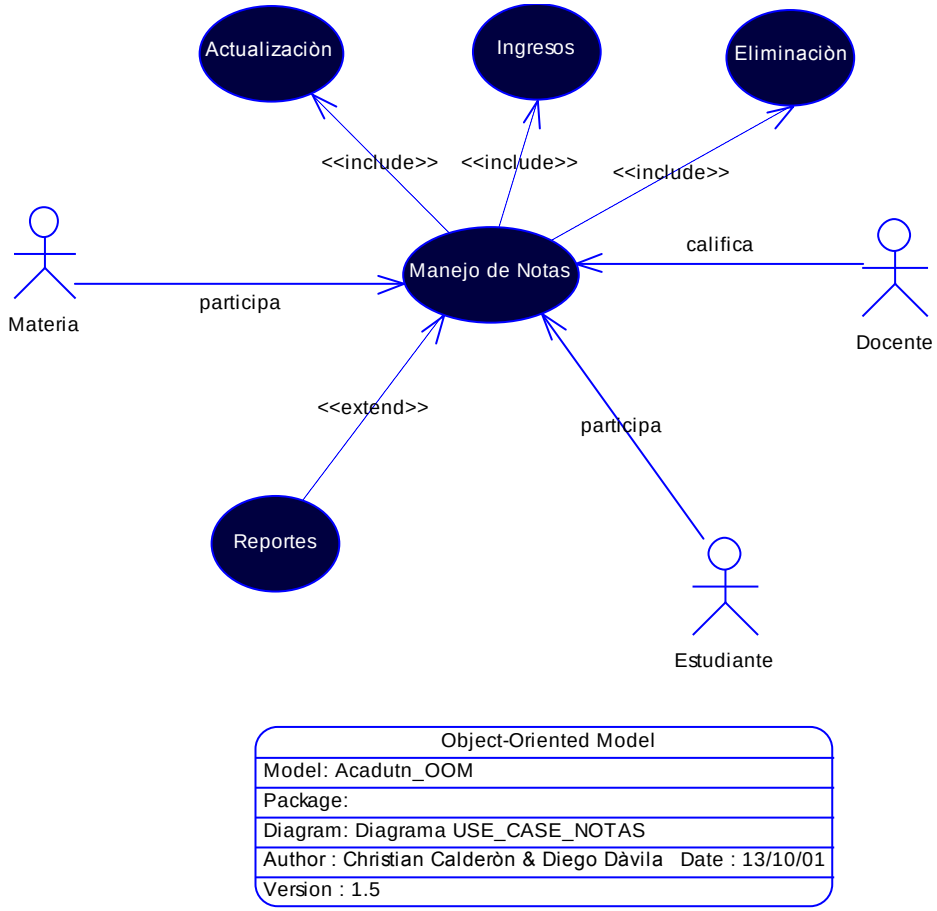


Figura 15: Diagrama de Casos de Uso para el proceso de Control de Notas

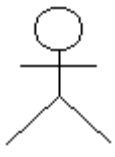
a) Caso de uso

Se representa en el diagrama por una elipse, denota un requerimiento solucionado por el sistema. Cada caso de uso es una operación completa desarrollada por los actores y por el sistema en un diálogo. El conjunto de casos de uso representa la totalidad de operaciones desarrolladas por el sistema. Va a acompañado de un nombre significativo. En el caso de ACADUTN se creo dos diagramas de casos de uso como subprocesos globales de análisis, el primer diagrama especifica el proceso de matriculación con todos sus procedimientos, este esquema ayudo a lograr especificar

los requerimientos funcionales de la aplicación. El segundo diagrama especifica otro de los grandes subprocesos de ACADUTN como fue el de Control de Notas.

En los dos gráficos anteriores se tiene como casos de uso a los siguientes procesos: Matriculación, Materias, Prerequisitos, Cierre de periodo, Pensum y Verificaciòn de Materias para el proceso de crear matrìcula; en tanto que para control de notas los casos de uso son: Actualizaciòn, Ingresos, Eliminaciòn, Manejo de Notas y Reportes.

b) Actor

Es un usuario del sistema, que necesita o usa algunos de los casos de uso. Se representa mediante un  , acompañado de un nombre significativo, si es necesario. Ejemplo: estudiantes, secretaria, docentes.

4.3.1.1 Relaciones en un diagrama de casos de uso

Entre los elementos de un diagrama de Casos de uso se pueden presentar tres tipos de relaciones, representadas por líneas dirigidas entre ellos (del elemento dependiente al independiente)

Comunica (communicates). Relación entre un actor y un caso de uso, denota la participación del actor en el caso de uso determinado. En el diagrama de ejemplo todas las líneas que salen del actor denotan este tipo de relación.

Usa (uses/include). Relación entre dos casos de uso, denota la inclusión del comportamiento de un escenario en otro. En el caso del diagrama de Control de Notas : Manejo de notas incluye en su comportamiento Actualizaciòn, Ingresos, Eliminaciòn como subprocesos dentro de ese proceso.

Extiende (extends). Relación entre dos casos de uso, denota cuando un caso de uso es una especialización de otro. Por ejemplo reportes es extendido implícitamente por manejo de notas.

4.3.2 Elementos básicos en un diagrama de estructura estática (Diagrama de Clases)

Un diagrama de estructura estática muestra el conjunto de clases y objetos importantes que hacen parte de un sistema, junto con las relaciones existentes entre estas clases y objetos. Muestra de una manera estática la estructura de información del sistema y la visibilidad que tiene cada una de las clases, dada por sus relaciones con las demás en el modelo.

A continuación se presenta un fragmento del Diagrama de Clases o Estructura Estática para ACADUTN:

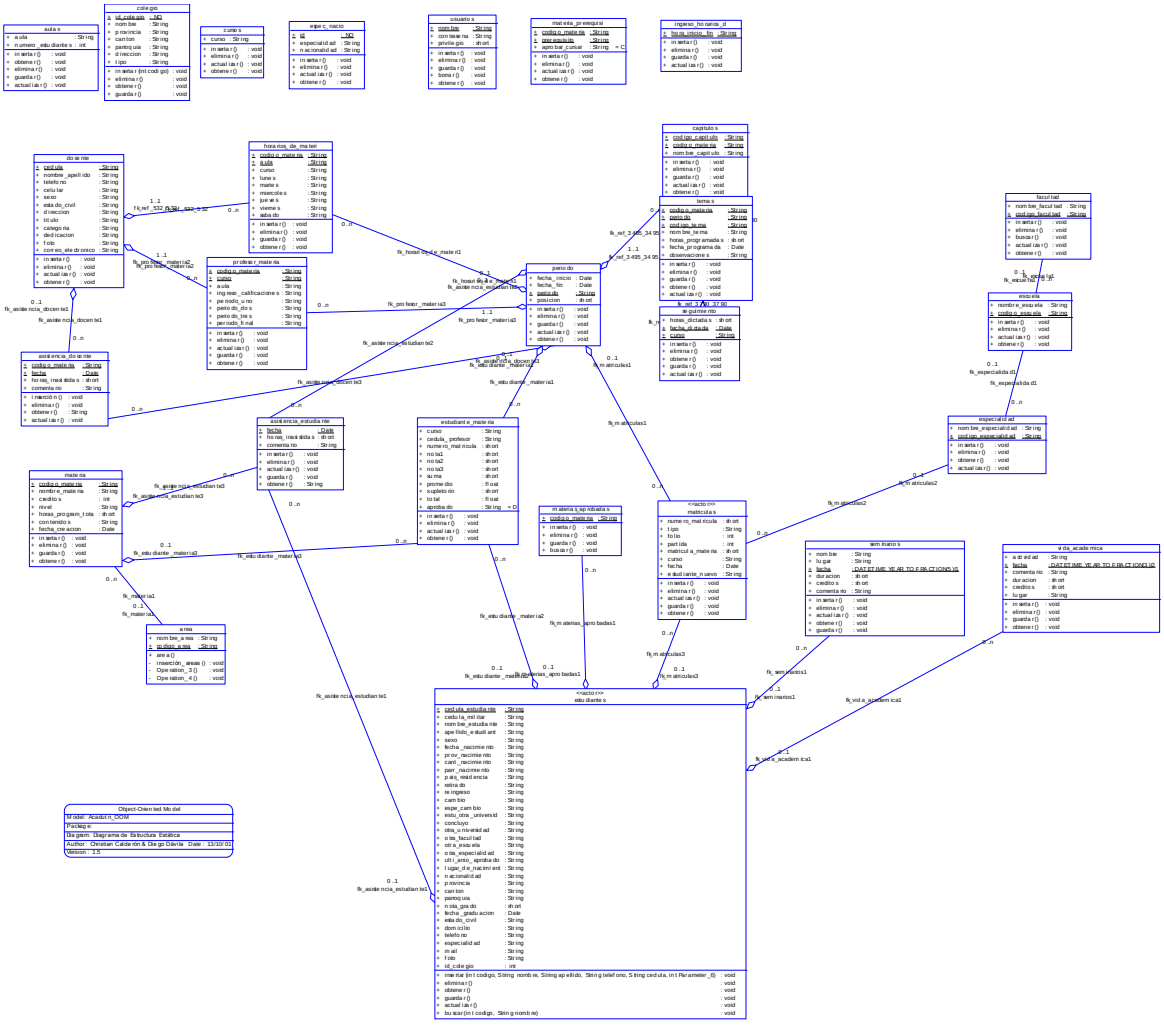


Figura 13: Diagrama de Clases o Estructura Estática

a) Clase

Representada por un rectángulo con tres divisiones internas, son los elementos fundamentales del diagrama. Una clase describe un conjunto de objetos con características y comportamiento idéntico. En el diagrama se destacan como clases principales Estudiante, Profesor, Materia y periodo y sobre estas se establece la mayoría de relaciones. Ustedes pueden notar que estas clases mencionadas eran los actores en el diagrama de casos de uso, esto sucede con mucha frecuencia, pero no es una regla general.

Los tres compartimientos estándares alojan el nombre de la clase, sus atributos y sus mensajes, respectivamente.

b) Atributo

Identifican las características propias de cada clase. Generalmente son de tipos simples, ya que los atributos de tipos compuestos se representan mediante asociaciones de composición con otras clases. La sintaxis de un atributo es

visibility name : type-expression = initial-value { property-string }

Donde *visibility* es uno de los siguientes:

\+ *public visibility*

\# *protected visibility*

\- *private visibility*

type-expression es el tipo del atributo con nombre *name*. Puede especificarse como se ve un valor inicial y un conjunto de propiedades del atributo.

En el caso del ejemplo, la clase Estudiante tiene dos atributos: uno denominado nombre de tipo string y el atributo sexo igual de string. En este caso, la herramienta utilizada ha cambiado la representación de la visibilidad, utilizando el símbolo  para indicar visibilidad privada, el símbolo  para visibilidad protegida y el símbolo  para indicar visibilidad pública.

c) Operación(método)

El conjunto de operaciones describen el comportamiento de los objetos de una clase. La síntaxis de una operación en UML es

visibility name (parameter-list) : return-type-expression { property-string }

Cada uno de los parámetros en *parameter-list* se denota igual que un atributo. Los demás elementos son los mismos encontrados en la notación de un atributo.

Como operaciones dentro de la clase estudiante tenemos: inserción, eliminación, actualización, guardar y buscar_datos. Al utilizar power designer como CASE ustedes pueden generar procedimientos almacenados de base de datos a partir de la definición de operaciones. Adicionalmente estos métodos automáticamente generan código power builder, lenguaje principal de front-end utilizado en el desarrollo del sistema.

d) Asociación (rol, multiplicidad, cualificador)

Una asociación en general es una línea que une dos o más símbolos. Pueden tener varios tipos de adornos, que definen su semántica y características. Los tipos de asociaciones entre clases presentes en un diagrama estático son:

1. Asociación binaria
2. Asociación n-aria
3. Composición
4. Generalización
5. Refinamiento

Cada asociación puede presentar algunos elementos adicionales que dan detalle a la relación, como son:

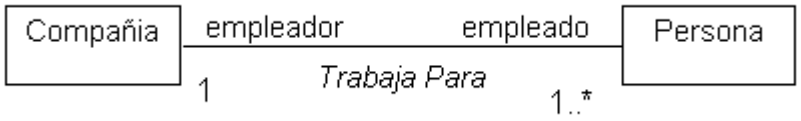
Rol: Identificado como un nombres al los finales de la línea, describe la semántica de la relación en el sentido indicado. Por ejemplo, la asociación de composición entre Maquina e Ingrediente recibe el nombre de *existencias*, como rol en ese sentido

Multiplicidad: Describe la cardinalidad de la relación. En el ejemplo anterior se utilizan **1**, **1 ..***, **5**, ***, como indicadores de multiplicidad.

e) Asociación binaria

Se identifica como una línea sólida que une dos clases. Representa una relación de algún tipo entre las dos clases, no muy fuerte (es decir, no se exige dependencia existencial ni encapsulamiento).

Un ejemplo de Asociación es la relación entre compañía y su persona.



En este caso la relación recibe el nombre genérico *Trabaja Para*, la compañía tiene uno o más instancias de la clase Persona denominadas *empleado* y cada empleado conoce su *empleador* (en este caso único).

f) Composición

Es una asociación fuerte, que implica tres cosas

- Dependencia existencial. El elemento dependiente desaparece al destruirse el que lo contiene y, si es de cardinalidad 1, es creado al mismo tiempo.
- Hay una pertenencia fuerte. Se puede decir que el objeto contenido es parte constitutiva y vital del que lo contiene

- Los objetos contenidos no son compartidos, esto es, no hacen parte del estado de otro objeto.

Se denota dibujando un rombo relleno del lado de la clase que contiene a la otra en la relación.

Existe también una relación de composición menos fuerte (no se exige dependencia existencial, por ejemplo) que es denotada por una un rombo sin rellenar en uno de los extremos. Un ejemplo puede encontrarse entre la típica relación materia-area donde una materia depende estrictamente de esa área.

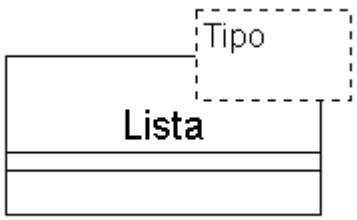
g) Generalización

La relación de generalización denota una relación de herencia entre clases. Se representa dibujando un triángulo sin rellenar en el lado de la superclase. La subclase hereda todos los atributos y mensajes descritos en la superclase.

h) Clase paramétrica

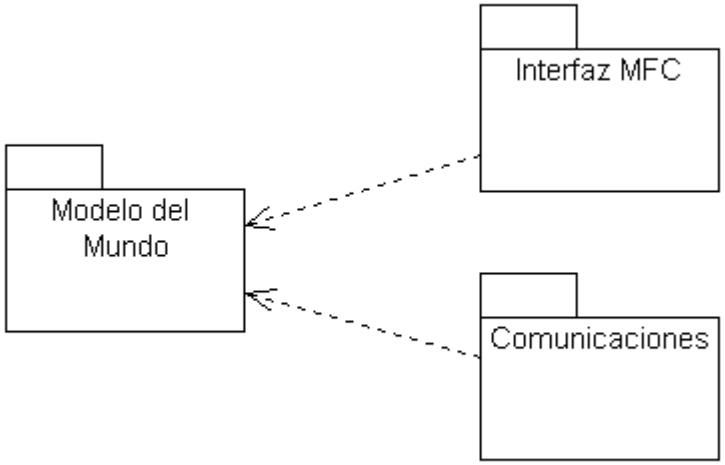
Una clase paramétrica representa el concepto de clase genérica en los conceptos básicos OO o de template en C++. Se dibuja como una clase acompañada de un rectángulo en la esquina superior derecha, con los parámetros del caso.

Por ejemplo, la clase Lista que utiliza un parámetro formal *Tipo* se vería de la siguiente manera



i) Paquete

Un paquete es una forma de agrupar clases (u otros elementos en otro tipo de diagramas) en modelos grandes. Pueden tener asociaciones de dependencia o de generalización entre ellos. Un ejemplo puede ser el siguiente



En este caso existen tres paquetes (que se muestran vacíos en este caso, con su contenido encapsulado), con dos de ellos dependiendo del Modelo del Mundo

j) Dependencia

Denota una relación semántica entre dos elementos (clases o paquetes, por el momento) del modelo. Indica que cambiar el elemento independiente puede requerir cambios en los dependientes. Se muestra como una línea punteada direccional, indicando el sentido de la dependencia. Puede tener por medio de estereotipos una explicación del tipo de dependencia presentada.

En el ejemplo anterior pueden verse dos relaciones de dependencia hacia el paquete Modelo del Mundo.

k) Nota

Es un comentario dentro de un diagrama. Puede estar relacionado con uno o más elementos en el diagrama mediante líneas punteadas. Pueden representar aclaraciones al diagrama o restricciones sobre los elementos relacionados (cuando el texto se encuentra entre '['y ']'). Se representa mediante un rectángulo con su borde superior derecho doblado.

4.3.3 Conceptos de un Diagrama de Secuencia

Un diagrama de secuencia muestra la interacción de un conjunto de objetos en una aplicación a través del tiempo. Esta descripción es importante porque puede dar detalle a los casos de uso, aclarándolos al nivel de mensajes de los objetos existentes, como también muestra el uso de los mensajes de las clases diseñadas en el contexto de una operación.

A continuación se muestra el diagrama de secuencia para el proceso de matriculación, que da detalle al caso de uso Matriculación:

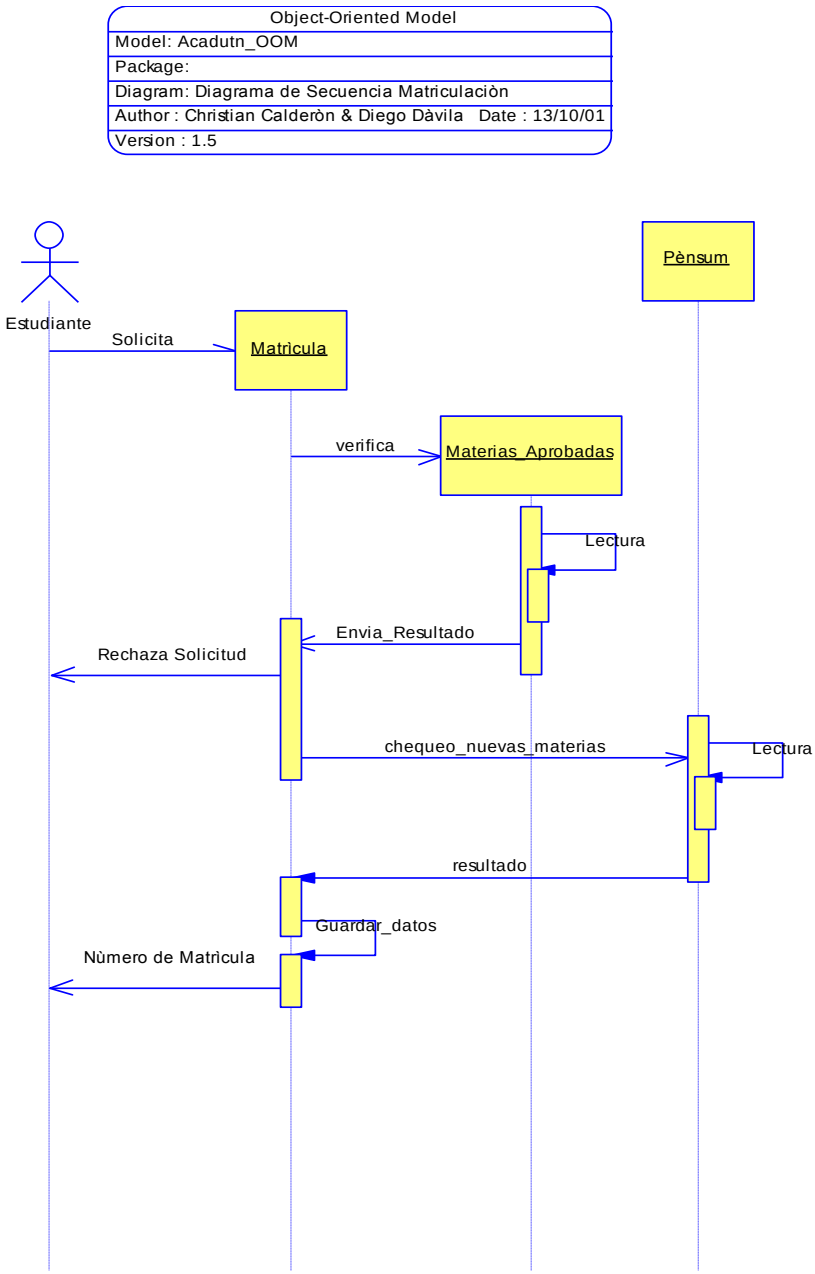


Figura 16: Diagrama de Secuencia

a) Línea de vida de un objeto

Un objeto se representa como una línea vertical punteada con un rectángulo de encabezado y con rectángulos a través de la línea principal que denotan la ejecución de métodos (véase activación). El rectángulo de encabezado contiene el nombre del objeto y el de su clase, en un formato *nombreObjeto: nombreClase*. Por ejemplo, el actor estudiante envía un mensaje de petición al objeto matrícula , luego este envía un mensaje al objeto materias _ aprobadas que a su vez en un ciclo de tiempo realiza un chequeo de las materias aprobadas del estudiante y envía un mensaje con el resultado de la misma; si es negativo el resultado se le rechaza la solicitud de matrícula , de los contrario se hace un chequeo de las nuevas materias a matricularse en el pensum y finalmente se guardan los resultados en el objeto matrícula.

b) Activación

Muestra el periodo de tiempo en el cual el objeto se encuentra desarrollando alguna operación, bien sea por sí mismo o por medio de delegación a alguno de sus atributos. Se denota como un rectángulo delgado sobre la línea de vida del objeto. En el ejemplo anterior el objeto Materias_aprobadas se encuentra activado mientras ejecuta el método correspondiente al mensaje lectura; el objeto **pensum** se encuentra activo mientras se ejecuta su método lectura (que ejecuta chequear_materias).

c) Mensaje

El envío de mensajes entre objetos se denota mediante una línea sólida dirigida, desde el objeto que emite el mensaje hacia el objeto que lo ejecuta.

En el ejemplo anterior el objeto matrícula envía el mensaje verifica al objeto materias_aprobadas y un poco más adelante en el tiempo el objeto materias_aprobadas se envía a sí mismo el mensaje lectura.

4.3.4 Etapas y actividades en el desarrollo OO basado en UML

Se pueden tener en cuenta las siguientes etapas

- Análisis de Requerimientos
- Diseño del sistema
- Diseño detallado
- Implementación y pruebas

A continuación en cada etapa se resaltan las actividades que le dan cuerpo y los documentos que se esperan al final de cada una de ellas.

4.3.4.1 Análisis de Requerimientos

Descripción

En esta etapa se logra claridad sobre lo que desea el usuario y la forma en la cual se le va a presentar la solución que está buscando.

1.- Actividades Técnicas

a) Identificar Casos de Uso del sistema

Esta información se representa en un diagrama de casos de uso.

Cómo encontrar un actor?

- Identifique los usuarios del sistema

- Porqué se diseña el sistema?
- Cuáles son los actores que el sistema va a beneficiar?
- Qué actores van a interactuar directamente con el sistema? (actores primarios)
- Qué actores van a supervisar, mantener, recibir información del sistema? (actores secundarios)

- Identifique los roles que juegan esos usuarios desde el punto de vista del sistema
- Identifique otros sistemas con los cuales exista comunicación

Cómo encontrar un caso de uso?

Identifique las operaciones importantes del sistema a construir

Cuáles son las principales tareas de un actor?

Qué información tiene el actor que consultar, actualizar, modificar? Cómo?

Qué cambios del exterior debe informar el actor al sistema?

Qué información debe informársele al actor, con respecto a los cambios del sistema?

Cómo encontrar relaciones entre actores y casos de uso?

Identifique los casos de uso en los cuales se ve implicado un actor

Busque relaciones *extends* entre casos de uso

Qué casos de uso son similares, diferenciándose en la forma en la cual hacen algunas operaciones?

Qué caso de uso redefine la forma en la cual se realiza una transacción dentro de otro caso de uso?

Busque relaciones *uses* entre casos de uso

Que casos de uso son usados como transacciones de otros?

b) Dar detalle a los casos de uso descritos

Describir la información de entrada y salida de cada caso de uso

Descripción detallada del caso de uso

- Descripción textual de su objetivo
- Variantes posibles para realizar este caso de uso. Diagramas de interacción de detalle (de secuencia o colaboración)
- Errores y excepciones posibles en el caso de uso

Relacionar el caso de uso con la interfaz a usuario que lo representa

Especificar el diálogo que da solución al caso de uso.

c) Definir una interfaz inicial del sistema (si es aplicable)

Dibujar las pantallas de interacción para los distintos actores-usuarios

Copiar el modelo mental del usuario

Revisar los elementos del modelo del mundo interesantes para el actor-usuario

Visualización típica de los elementos del modelo del mundo

Información relevante para el actor

Metáforas de interacción válidas

Especificar el diálogo que da solución a cada caso de uso que se soluciona con la interacción con esta interfaz. Puede especificarse este diálogo de varias maneras, dependiendo de la complejidad de la interfaz definida (en esta etapa se sugiere escoger el mínimo nivel de detalle posible, para dar más libertad de diseño en las etapas posteriores):

1. Por medio de una descripción textual de su funcionamiento
2. Por medio de diagramas de interacción que muestren la secuencia de operaciones entre los objetos de interfaz y los actores involucrados

3. Por medio de diagramas de estados, donde se muestre claramente los estados de la interfaz

   Por medio de un prototipo funcional, en términos de la interacción con el usuario

Definir restricciones para la comunicación con actores y sistemas

Describir en el detalle del actor o de la relación con el caso de uso particular.

d) Desarrollar el modelo del mundo

Esta información se representa en un diagrama de estructura estática de clases

I. Identificar Clases

- Elementos físicos y lógicos dentro del sistema a modelar
- Top-down: Comenzar por la clase del objeto más general (el mundo). Encontrar sus componentes hasta llegar a clases de tipos básicos
- Identificar los sustantivos del enunciado del problema y determinar si son clases del modelo del mundo
- Identificar clases desde el punto de vista de la información
    - Identifique los elementos del espacio del problema
    - Identifique otros sistemas relacionados como objetos externos
    - Identifique dispositivos relacionados
    - Identifique los eventos que el sistema debe recordar y manipular
    - Identifique los roles de los elementos del mundo
    - Identifique sitios
    - Identifique unidades organizacionales importantes en el problema

- Identificar clases desde el punto de vista funcional (casos de uso)
    - Identifique los objetos que participan en un caso de uso particular
    - Continue con los mensajes de cada objeto, dejando para el final los atributos.
- Identificar clases desde el punto de vista de sus estados
    - En qué estados está en sistema? Cuáles objetos determinan estos estados?
    - Cómo es el ciclo de vida de estos objetos?

Posibles errores:

- Una clase HACE en vez de una clase ES
- Solo se requiere un objeto de la clase
- Dificultad para encontrar atributos
- Objetos con iniciativa propia
- Es un objeto y un usuario a la vez
- Solo se encuentra un servicio
- Varias clases tienen los mismos atributos o servicios
- Solo tienen información o mensajes no relevantes para el problema
- Vista Funcional: Dividir un sistema de la manera clásica

II. Identificar atributos y asociaciones.

- Cuáles son las características determinantes del objeto en el dominio del problema?
- Con qué objetos esta relacionado?

- Con qué objetos debe estar relacionado para realizar sus mensajes?
- Identificar el nombre, los roles y cardinalidad de las asociaciones
- Qué asociaciones hay de tipo partes y un todo (composición)?
- Qué información se requiere en una clase para realizar su comportamiento?

Posibles errores

- Identificar atributos o relaciones no relevantes a los casos de uso identificados
- Las relaciones no reflejan directamente la realidad

III. Identificar mensajes

- Punto de vista funcional
    - Qué mensajes debe tener un objeto para colaborar en un caso de uso?
- Punto de vista de comportamiento
    - Qué comportamiento se espera de un objeto dado en el modelo del mundo?
    - Qué mensajes se requieren para manipular la información que contienen?
    - Qué mensajes requieren para manipular las relaciones que tiene?
    - Qué mensajes hacen que el objeto cambie de un estado a otro?

Posibles errores

- Identificar servicios no relevantes a los casos de uso identificados
- Identificar servicios que no puede realizar la clase por falta de información

IV. Identificar relaciones de herencia

- Qué clases son abstracciones naturales de clases ya existentes?
- Qué clases comparten atributos o servicios?
- Qué clases extienden atributos o servicios de otras?

Posibles Errores

- No tener una relación *Es Un* entre las clases

V. Identificar restricciones del modelo

- Identificar valores posibles y no posibles de los atributos. Describirlos como restricciones de las clases
- Identificar valores permitidos para las asociaciones. Describirlos como restricciones de la asociación
- Identificar restricciones que relaciones dos o más atributos o relaciones. Describirlas dentro de la clase correspondiente

Posibles errores

- Hay estados en el modelo imposibles en el mundo real
- Hay estados en el mundo real no considerados en el modelo

e) Validar los modelos

Validar las restricciones descritas para las clases

- Para cada clase evaluar la completitud de las restricciones

- Desarrollar objetos ejemplo que cumplan con las restricciones y que no sean válidos en el mundo real

Validar atributos y mensajes

- La clase tiene toda la información necesaria para desarrollar la tarea?
- La clase tiene las relaciones necesarias para propagar el mensaje y cumplir con la tarea?
- Los mensajes si son utilizados dentro del contexto del problema?
- Los mensajes obligan la conservación de las restricciones del modelo?

Desarrollar diagramas de interacción (diagramas de secuencia o de colaboración) para la variante por defecto de cada caso de uso, usando los objetos del modelo del mundo encontrados y sus mensajes.

- Escoger la opción por defecto de cada caso de uso
- Identificar los objetos involucrados
- Desarrollar el diagrama de secuencia o el de colaboración para la interacción

Validar con un usuario representativo de cada actor

- Validar la funcionalidad esperada para el actor en particular: completitud, relevancia
- Validar los diagramas de interacción descritos como detalle de los casos de uso del actor
- Validar la interfaz diseñada y el diálogo descrito

Documentos Entregables en esta Etapa

* Casos de Uso iniciales:

Requerimientos más importantes del sistema

Usuarios y sistemas externos en comunicación

Especificación de requerimientos

* Borradores de Interfaz :

Presentaciones iniciales para los distintos usuarios de la forma de solucionar sus requerimientos.

* Modelo del mundo inicial. Versión de requerimientos:

Clases, relaciones entre clases y especificación

4.3.4.2 Diseño del Sistema

En esta etapa se define una subdivisión en aplicaciones del sistema (si es lo suficientemente grande) y la forma de comunicación con los sistemas ya existentes con los cuales debe interactuar.

1.- Actividades Técnicas

a) Identificar la arquitectura del sistema

Definir componentes del sistema, las aplicaciones y su ubicación. Representarlos por medio de nodos, componentes y objetos activos (representando las aplicaciones) dentro de los nodos.

Definir mecanismos de comunicación. Expresarlos por medio de asociaciones de dependencia entre los nodos, componentes o aplicaciones y, si es conocido, agregar un estereotipo para definir el protocolo de comunicación requerido. Agregar notas con restricciones, rendimiento esperado y demás detalles de las conexiones.

Particularizar los casos de uso a la arquitectura planteada. Refinar los casos de uso ya existentes de la etapa anterior para adecuarse a la arquitectura planteada.

Validar arquitectura. Comprobar la validez técnica, económica y organizacional de la propuesta.

Documentos Entregables

Diagramas de Ejecución, versión inicial

4.3.4.2.1 Diseño detallado del Sistema

Descripción

En esta etapa se adecua el análisis a las características específicas del ambiente de implementación y se completan las distintas aplicaciones del sistema con los modelos de control, interfaz o comunicaciones, según sea el caso.

1.- Actividades Técnicas

a) Detalles de implementación del modelo del mundo

Completar el detalle de las clases:

Tipos de los atributos

Atributos y métodos de clase

Diseño de asociaciones

Completar los métodos

Enriquecer el modelo con el framework de base en el ambiente de implementación escogido.

Incorporar patrones de diseño.

Subdividir en paquetes .

Definir excepciones .

Completar comportamiento de las clases: Constructores, destructores, modificadores, consultores.

Adecuar el modelo a las características del lenguaje de programación. Evaluar eficiencia .

Validar el Sistema.

b) Desarrollar el modelo de interfaz

Conocer el framework[14] de base.

Enlazar las clases de interfaz con las clases del modelo del mundo.

c)Desarrollar los modelos de control, persistencia y comunicaciones

Conocer los frameworks de base.

Enlazar las clases del framework con las demás clases del sistema.

Documentos Entregables

Diagramas de clases y paquetes, con el detalle de la implementación

Diagramas de interacción con el detalle de las operaciones más importantes del sistema

Diagramas de estados y/o actividades para las clases concurrentes o complejas

4.3.4.3 Implementación y pruebas

Descripción

Se desarrolla el código de una manera certificada.

[14] Marco base sobre el cual se va a implementar una aplicación.

1.- Actividades Técnicas

a) Definir estándares de programación

Asimilar los idiomas aplicables al lenguaje

Conocer y adecuar estándares de programación al lenguaje

Definir estructura de directorios

Diseñar makefiles.

b) Codificación y pruebas unitarias

Revisiones de código.

Tratamiento de Trace y Log

c) Pruebas de módulos y de sistema

Casos de prueba

Procedimiento de instalación

Documentos Entregables

Código fuente

Soporte de pruebas unitarias

Documentación del código

4.4 Desarrollo de Prototipos en Espiral

Una vez que se ha tratado de describir brevemente la notación de UML el proceso de creación fue en base a prototipos en espiral tal como representa la figura:

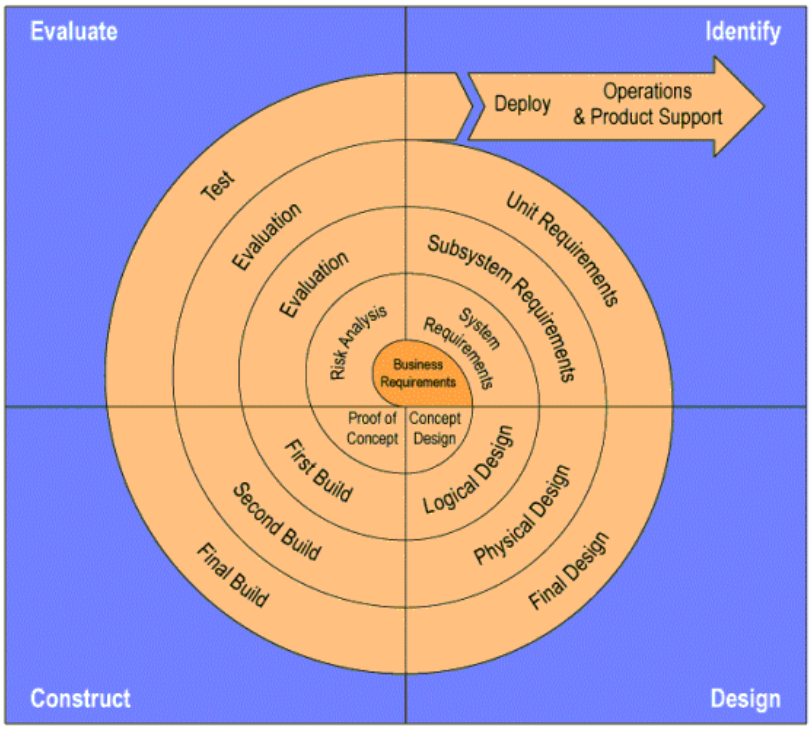


Figura 16: Metodologìa en Espiral

Fuente: Curso de Sybase Data Modeling using Power Designer 8

En la cual se fue tomando en cuenta ciertos requerimientos iniciales de la Administración Académica de la UTN, luego se realizó tareas de análisis para construir el primer prototipo que debido a las tres modal idees de régimen escolar (créditos, años y semestres) se decidió iniciar por la Modalidad de Créditos Semestrales en la Escuela de Ingeniería en Sistemas, una vez que se obtuvo el primer prototipo funcional se entró a la fase de pruebas para las especificación de nuevos requerimientos, con los resultados se regresó a la etapa de evaluación y análisis para desarrollar el segundo prototipo y así sucesivamente hasta completar los objetivos iniciales de implementación.

En base a este Modelo en Espiral se siguieron implementando prototipos funcionales para el resto de modalidades académicas.

Es muy importante destacar que para las personas que utilicen este tipo de modelamiento es importante realizar un documento de especificaciones funcionales para cada módulo de un sistema determinado, ya que sino se lo hace se corre el riesgo de no saber cuando terminar un prototipo final.

4.5 Definición del Sistema

En una arquitectura cliente/servidor, es de vital importancia tener una fuerte base en la que se desarrollarán e implementarán las aplicaciones, es decir, se requiere de un software base fuerte, que gestione de una manera óptima todas las tareas subyacentes de la red y de recursos que necesiten las aplicaciones, además que me brinde servicios de alta disponibilidad. Es por eso que, antes del desarrollo de una aplicación, es menester que se analice el software base, en función de sus pros y contras, y de esta manera elegir el mejor.

4.5.1 Selección del Sistema Operativo

En estos días de globalización de la economía, las cifras presentadas por compañías de renombre encargadas del análisis, investigación, evaluación y presentación de datos, correspondientes al comportamiento de diversos segmentos del mercado, proporcionan una herramienta indispensable en las decisiones que las empresas toman, al momento de realizar una inversión de cualquier índole. Para decidir cuál de ellos es el que más conviene a las necesidades presentes y futuras de la empresa, se debe considerar su participación actual en números, así como un pronóstico de las tendencias en los años por venir. Estas fuentes son por ejemplo dos de las compañías más prestigiadas en el campo del análisis de datos para computación e informática:

International Data Corporation (IDC) y Computer Intelligence, quienes continuamente realizan sondeos en todo el mundo, para conocer el comportamiento de los productos líderes que marcan la pauta en el desarrollo tecnológico de nuestra sociedad.

El modelo cliente/servidor se ha convertido en un estándar en las instituciones que desean un sistema robusto y seguro, por lo que los sondeos de mercado para encontrar el sistema operativo de red más requerido con estas características, se han incrementado en los últimos años.

Microsoft tuvo un inicio lento con su sistema Windows NT, pero recientemente ha tenido un gran despunte que lo ha llevado a estar en segundo sitio detrás del gigante Novell.

Con menos fuerza se reparten lo que resta del mercado el sistema OS/2 de IBM, la solución para redes cliente/servidor AppleShare de Apple

Adicional un gran espacio en el mercado es utilizado por las diferentes variantes de UNIX como Solaris, UnixWare y el clon Linux.

Para el presente análisis, se han utilizado tres factores sobre los cuales se ha producido el análisis:

1. Características
2. Costos
3. Requerimientos de hardware.
4. Familiaridad de uso.

1. CARACTERISTICAS:

A continuación se presenta en forma breve las principales características (ventajas/desventajas) de algunos NOS y de algunos OS clientes.

a) Novell Netware

Ventajas:

- NDS (Servicio de Directorios de Red) ofrece un directorio global y escalable, que puede diseñarse para gestión centralizada o descentralizada.
- Excelente administración de redes en gran escala.
- Es un sistema operativo de red independiente del hardware.
- Ofrece el mejor sistema de impresión y archivos.
- Excelente nivel de seguridad.
- Soporta aplicaciones a través de Módulos cargables de NetWare (NLM).
- La gran infraestructura de Novell es capaz de dar soporte técnico y asistencia por mucho tiempo.

- Cuando se descubre un error en la versión reciente de NetWare, Novell hace públicas las posibles soluciones para usuarios nuevos y antiguos.

Desventajas:

- NDS es bastante complejo de instalar y administrar.
- NetWare está perdiendo mercado por la complejidad de NetWare 4.1 y NDS.
- Servicios como FTP o HTTP requieren comprar software adicional de Novell.

- La actualización de una versión a otra es lenta y compleja.

- Puede ser caro para redes pequeñas.

b) WINDOWS NT SERVER-WINDOWS 2000 ADVANCED SERVER

Ventajas:

- Proporciona una plataforma de propósito general superior.

- Soporta múltiples procesadores.

- Excelente seguridad.

- Existe una gran variedad de aplicaciones diseñadas exclusivamente para NT, incluyendo freeware y shareware.

- Es fácil de instalar y manejar.

- Tiene una interfaz de usuario muy amigable.

- NT es IGU (Interfaz Gráfica de Usuario) y SO (Sistema Operativo) a la vez.

- NT tiene el respaldo de Microsoft, la compañía más poderosa en software del mundo.

- Tiene buen soporte técnico.

- NT es económico para entornos medianos.

Desventajas:

- Es un poco lento como servidor de archivos e impresión.

- Cuando se descubre un error en la versión reciente del sistema, Microsoft se espera al lanzamiento de la siguiente versión para solucionarlo.

- Presenta serias dificultades en entornos muy grandes.

- Necesita muchos recursos de cómputo para funcionar correctamente.

c) UNIX

Ventajas:

- Sistema multiusuario real, puede correr cualquier aplicación en el servidor.
- Es escalable, con soporte para arquitectura de 64 bits.
- El costo de las diferentes variantes de Unix es muy reducido y algunas son gratis, como Linux.
- Se pueden activar y desactivar drivers o dispositivos sin necesidad de reiniciar el sistema.
- UNIX puede trabajar con CLI (Command Line Interface).
- Los kernels de Unix se confeccionan según las necesidades.
- Es la mejor solución para enormes bases de datos.

Desventajas:

- La interfaz de usuario no es muy amistosa en algunas versiones.
- Requiere capacitación, ya que debido a su complejidad, no cualquiera puede usarlo.
- Padece de la falta de aplicaciones comerciales con nombres importantes.
- La efectividad como servidor de archivos e impresión no es tan eficiente como en otros NOS.

d) COSTOS

A la hora de iniciar un proyecto, el factor económico juega una pieza clave en la decisión a tomar, y la selección de un NOS no es la excepción. El costo varía entre cada NOS, partiendo desde precios bastante altos, hasta sistemas de distribución gratuita (Linux).

El pagar más por un NOS no significa que éste vaya a resultar más productivo para la organización que uno de bajo costo, por lo que se debe buscar aquél que cumpla con las expectativas de la empresa, tratando, claro, que el desembolso sea siempre el menor posible. Enseguida se presenta información reciente sobre los costos de varios NOS.

3. REQUERIMIENTOS DE HARDWARE:

Cada sistema operativo de red tiene diferentes requerimientos de hardware para funcionar correctamente, si éstos no son satisfechos, el sistema puede no operar o trabajar en un nivel muy por debajo del esperado, ocasionando serios problemas en la red. Es conveniente entonces, conocer los requerimientos de cada NOS para ver si el equipo actual los satisface o si es necesario invertir en nuevo hardware.

Windows NT 4.0- Win2000 AS:

- Procesador Pentium II 350 MHZ, para sistemas Intel y compatibles; procesador RISC compatible con Windows NT Server 4.0 para sistemas basados en RISC.
- 128 MB de memoria.
- Mínimo 250 MB de espacio en disco duro para sistemas Intel y compatibles; 160 MB para sistemas basados en RISC.
- Unidad de CD-ROM.
- Adaptador gráfico VGA, SVGA o compatible con Windows NT Server 4.0.

NETWARE 5:

Servidor:

- PC con procesador Pentium o superior.

- 64 MB de RAM.

- 1 GB en disco duro.

- Al menos un adaptador de red.

- Cable de red.

- Una unidad de CD-ROM para instalación.

Estaciones de trabajo:

- Para cada estación de trabajo se debe tener un adaptador de red y una computadora corriendo el sistema operativo cliente requerido.

UNIX Solaris 8

- Plataforma SPARC o Intel 486 (400 MHz) al Pentium.

- De 600 MB a 1 GB en disco duro.

- Mínimo 32 MB de ram.

Linux

- Procesador Intel 386 y posteriores, SPARC, Alpha, PowerPC, etc.

- Mínimo 4 MB de memoria.

- De 150 a 200 MB en disco duro.

Windows 95

- PC con procesador 486 a 25 MHz o superior.
- Mínimo 8 MB de memoria.
- 40 a 45 MB de espacio en disco duro.
- Pantalla VGA o de resolución superior.

4. FAMILIARIDAD DE USO:

Al momento de seleccionar el sistema operativo de red para una LAN, la organización necesita tener información sobre ciertos tópicos, que pueden influir de manera directa o indirecta en el éxito de la implementación del NOS en la red. Algunos de estos requerimientos son: profesionalización del personal, conocimiento de costos de cada NOS, requerimientos de hardware e interoperabilidad de los sistemas a instalar con otros similares, dentro y fuera de la organización.

Una vez hecho este análisis la recomendación ideal es que al menos la base de datos de un sistema transaccional se encuentre sobre un sistema operativo Unix o en su defecto en un Linux, por que motivo, estos sistemas son robustos y tolerables a fallas. En el caso de ACADUTN en un principio su base distribuida está instalada sobre tres sistemas linux y el resto sobre NT, no sé utilizó Unix por costos.

4.5.2 Selección del Front - End

Para realizar un análisis de herramientas para desarrollo se debe tomar en cuenta ciertos parámetros, como por ejemplo:

1. El diseño RAD.
2. Soporte Multiplataforma
3. Tipo de Conexión hacia la base de datos
4. Portabilidad al Internet
5. Costos

Se evaluaron herramientas como Microsoft Visual Studio, Borland Delphi y Sybase Power Builder 7, esta última sin lugar a dudas es la mejor del mercado y no tiene comparación al resto del software evaluado, es por eso que para la realización de ACADUTN se la utilizó.

A continuación se describe brevemente las características del producto.

- Soporte a componentes nativos PowerBuilder para ser ejecutados en EAServer de Sybase.
- Soporte par la construcción de componentes COM para ser desplegados en Microsoft Transaction Server (MTS).
- Validación de código para EAServer o componentes COM/MTS.
- Control sobre componentes transaccionales.
- Soporte a instanciación y "pools" de componentes.
- Soporte a tipos "ResultSet" y "ADOResult" para el retorno de resultados de base de datos desde componentes.
- Bitácora de errores para EAServer o MTS.
- Asistentes para la creación y despliegue de componentes PowerBuilder en EAServer.
- Asistentes para la creación de clientes PowerBuilder para EAServer.
- Soporte al protocolo de comunicaciones IIOP.

- Generación automática de IDL para componentes desarrollados.
- Depuración remota de componentes PowerBuilder que se ejecuten nativamente.
- Edición en vivo para prueba de componentes.
- Asistentes para la creación/edición de componentes COM.
- Despliegue automático hacia MTS desde el ambiente de desarrollo.
- NVOs de PowerBuilder encadenados en un sólo DLL o servidor PB COM.
- Soporte para interfaces personalizadas (a través de OLE automation data types).
- Propagación de errores de componentes hacia MTS.

4.5.3 Selección del Back - End

Las fuentes operacionales son muy importantes en la creación de aplicaciones cliente-servidor. Es por eso que escoger el motor fue una de las situaciones de mayor cuidado y tiempo. Básicamente ACADUTN empezó con datos cargados sobre Microsoft SQL Server 6.5 , luego debido a la arquitectura multiplataforma que se empleo es decir servidores unix, linux trabajando con windows nt server, este motor fue desechado.

Entonces se empezó a realizar evaluaciones con motores como Sybase e Informix, decidiéndose finalmente por Informix Dynamic Server 2000. Este es un motor relacional de alto rendimiento y escalabilidad, con la capacidad de manejar tipos de datos nativos html, excel, etc.

Otra de las características principales es el manejo de replicación a través de log de transacciones, bloqueos a nivel de fila-columna y una funcionalidad de administración al Web.

En resumen la implementación de ACADUTN empezó utilizando para su análisis y diseño Orientación a Objetos en base a UML, es decir a través de la Herramienta CASE Power Designer 7 se generaron los diagramas de Casos de Uso(toma de requisitos), Secuencia(Comportamiento de los Objetos del Negocio), Clases(Modelamiento de la Aplicación), Modelo Conceptual(a partir del diagrama de clases se generaron las estructuras para la base de datos), Modelo Físico(una vez escogido el motor de base de datos en este caso Informix se implemento tanto las tablas, disparadores y demás estructuras relacionales); con esto se empezaron a crear cada uno de los prototipos funcionales empezando por el Sistema de Créditos para la Facultad de Ingeniería en Ciencias Aplicadas, estos prototipos fueron creados utilizando Power Builder 6.5 y luego Power Builder 7.0 como front-end.

Una vez obtenido la mayoría de prototipos se empezó con las pruebas para el internet utilizando tecnología Informix, para luego finalmente implementar la replicación al repositorio central.

CAPITULO V

5.1 Verificaciòn de la Hipótesis

"La aplicación de la Tecnología Cliente/Servidor con un esquema de Replicación en Bases de Datos e Internet asistidos por CASE, permitirá definir un Modelo Informático para la Administración eficiente de los Datos Estudiantiles de la Universidad Técnica del Norte."

La hipótesis planteada en el Anteproyecto de Tesis y que se describe en el párrafo anterior si se cumplió por cuanto:

1.- Se diseño un esquema de base de datos capaz de soportar sin ninguna modificación las tres modalidades académicas existentes en la Universidad Técnica del Norte, como son Créditos, Semestres y Años, este esquema se implementó sobre arquitectura Cliente/Servidor.

2.- El modelo de replicación utilizado en ACADUTN (Modelo de Consolidación) permite mantener un repositorio histórico de datos del estudiante, de tal forma que la información se consolida en un solo sitio brindando la capacidad de emitir una infinidad de reportes académicos.

3.- Al utilizar una correcta metodología orientada a objetos y documentada sobre un CASE se logró mejorar los tiempos y resultados de las diferentes etapas de la construcción de ACADUTN.

4.- Finalmente con el uso del Internet dentro de ACADUTN, se mejoró notablemente los procesos de consultas de notas de los estudiantes, ya que ahora lo pueden hacer desde cualquier sitio y de manera eficaz.

Por todas las consideraciones anteriores la validez de la hipótesis planteada inicialmente fue la correcta y se cumplió a cabalidad.

5.2 Conclusiones

5.2.1 Investigaciòn

- Las aplicaciones Cliente/Servidor pueden adoptar muchas formas. Existen casos en que no puede ser factible debido a equipos, requisitos de usuario o disponibilidad del software. Cliente/Servidor es aconsejable implementar cuando no se piensa en reutilizar lógica de negocio entre aplicaciones y cuando no se tiene pensado migrar de plataforma back-end. Caso contrario se debe implementar arquitecturas distribuidas basadas en n-capas.

- Como resultado de la investigación en arquitecturas cliente/servidor se debe en lo posible programar la lógica del negocio del lado del back-end para optimizar el tiempo de respuesta y tráfico en la red.

- El reto más importante para crear un ambiente distribuido es la fase de diseño. Es extremadamente importante que él administrador o responsable de la base de datos defina: qué datos necesita replicar, qué tan frecuente se quiere propagar cambios entre las replicaciones y principalmente cómo evitar o resolver conflictos de actualización en múltiples sitios de replicación.

- Es muy importante en la etapa de análisis y diseño de sistemas distribuidos identificar si se necesita Distribución de Datos o Replicación de Datos.

- La Replicación de datos es importante debido a que facilita a las organizaciones a proveer a sus usuarios acceso a información actualizada donde y cuando ellos lo necesiten.

- La replicación de datos puede proveer un amplio espectro de beneficios que incluyen mejoras en rendimiento cuando los recursos centralizados lleguen a saturarse,

incremento de la disponibilidad de datos, capacidad, y soporte para datawarehousing, además de facilidad para soporte de decisiones.

- La técnica de Replicación Asincrónica maximiza la disponibilidad y rendimiento de tiempo de respuesta, aunque ésta requiere de planificación y diseño para asegurar integridad de datos y fundamentalmente resolver conflictos de actualización.

- A la hora de replicación de datos, el aspecto más importante a considerar es el Control de concurrencia. Dicho factor incide en el rendimiento y confiabilidad del proceso de replicación.

- Es importante antes de iniciar un proyecto de software conocer bien las metodologías que se pueden utilizar para construir software y además saber identificar cual se acopla a sus necesidades

- De manera ineludible la Metodología UML (Unified Modeling Languaje) se constituye hoy por hoy en la metodología vanguardista para documentar un proyecto de software, ya que cubre todas las fases del diseño de software.

- Para obtener un resultado de calidad en una investigación es vital utilizar técnicas que permitan fomentar la participación en la autoenseñanza personal juntamente con el desarrollo en equipo.

5.2.2 Aplicativo

- La topología de replicación de “Consolidación”, es el tipo óptimo de replicación para el proyecto AcadUtn, dado por el flujo de información que no requiere retroalimentación.

- El proyecto ACADUTN implementado en la Universidad Técnica del Norte para gestionar la información concerniente al Sistema Educacional, contempla de manera efectiva las tres modalidades que actualmente están vigentes en la Universidad, a saber: Modalidad Anual, Modalidad Créditos y Modalidad Semestres, mejorando de esta manera los procesos y el control que se tiene sobre los mismos, al automatizarlos mediante un software.

- ACADUTN tiene un óptimo tiempo de respuesta al haber utilizado un producto front-end como Sybase Power Builder que ofrece conexiones nativas hacia la base de datos y manejo de datos en memoria en las máquinas cliente.

- El módulo diseñado para Internet, es totalmente aplicable y manejable para la intranet local, es decir para cada unidad académica, con el beneficio de que la información sustancial esté "pública" y al momento para cada estudiante.

- El proyecto AcadUtn está construido sobre una sólida infraestructura de red, como es la "Red Universitaria", la misma que emplea un cableado estructurado, que brinda una base fuerte para la implementación de proyectos modelo Cliente/Servidor propiamente dichos.

5.3 Recomendaciones

- Fomentar en la Universidad la autoenseñanza y la investigación de nuevas tecnologías en los estudiantes para lograr objetivos de calidad.

- Es importante poner énfasis en que para la implementación de un proyecto informático, se requiere de dar soluciones de acuerdo al entorno en el que se lo desarrollará. Este concepto se maneja en diferentes ámbitos a saber: humano, económico, infraestructura, disponibilidad, acceso a datos es decir, se debe tomar en cuenta:

  Comunicaciones existentes(red, modem, radio).

  Desarrollo Modelo Cliente/Servidor, Web, n capas, etc.

  Financiamiento.

  Concurrencia y disponibilidad de datos

- Es imprescindible conocer a profundidad el "Sistema" en análisis, conocimiento en cuanto a procesos y a políticas internas y otras variantes que se pueden dar, con la finalidad de entregar un producto personalizado para las necesidades del usuario final. Para esto es necesario realizar un levantamiento de datos, formal o informalmente, con personas funcionales, es decir, con las personas involucradas directamente con los procesos del día a día del "Sistema" en cuestión.

- El documento inicial o de definición del proyecto para construir un prototipo funcional o aplicativo de una tesis de investigación debe elaborarse con el mayor análisis y documentación en las especificaciones funcionales, con la finalidad de cumplir con profesionalismo en los tiempos planteados.

- Para implementar de una manera óptima un sistema informático, es necesario manejar las mejores herramientas (de acuerdo al entono en el que se va a desarrollar, lo que obliga al Ingeniero a ser lo suficientemente versátil para desarrollar el proyecto en una vasta cantidad de herramientas) y mantenerse a la vanguardia de las nuevas tendencias tecnológicas, ya que en informática y computación nada perdura, sino el cambio.

- Desarrollar una cultura de "respaldos", esto es, redundar en los llamados "backups"en cuanto se refiere a datos críticos del negocio, códigos fuentes, documentos, en localizaciones diferentes a la máquina servidor.

- Las organizaciones deben tratar de automatizar los procesos, ya que esto genera control sobre los mismos y mejora la administración.

- Las empresas en general deberían generar proyectos en los que se involucre la informática, los cuales van en mejora y crecimiento de la empresa, a mediano plazo, deben verlo como una inversión y no como un gasto.

- Las instituciones deben dar apertura a los nuevos profesionales, que vienen con ideas frescas y con conocimiento de las nuevas tendencias tecnológicas, de esta manera actualizar la tecnología que manejan.