UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA EN CIENCIAS Y SISTEMAS

REGLAS DEL NEGOCIO EN ARQUITECTURA TRES CAPAS

BENITO MANOLO COTI COLOP
ASESORADO POR: LIC. SERGIO MEJÍA SÁNCHEZ

GUATEMALA, NOVIEMBRE DE 2003

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

REGLAS DEL NEGOCIO EN ARQUITECTURA TRES CAPAS

TRABAJO DE GRADUACIÓN

PRESENTADO A JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

BENITO MANOLO COTI COLOP

ASESORADO POR: LIC. SERGIO MEJÍA SÁNCHEZ

AL CONFERÍRSELE EL TÍTULO DE
INGENIERO EN CIENCIAS Y SISTEMAS

GUATEMALA, NOVIEMBRE DE 2003

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INEGENIERIA

NÓMINA DE LA JUNTA DIRECTIVA

DECANO:	Ing. Sydney Alexander Samuels Milson
VOCAL I	Ing. Murphy Olympo Paiz Recinos
VOCAL II	Lic. Amahán Sánchez Alvarez
VOCAL III	Ing. Julio David Galicia Celada
VOCAL IV	Br. Kenneth Issur Estrada Ruiz
VOCAL V	Br. Elisa Yazminda Videz Leiva.
SECRETARIO	Ing. Pedro Antonio Aguilar Polanco.

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENRAL PRIVADO

DECANO	Ing. Sydney Alexander Samuels Milson
EXAMINADORA	Inga. Ligia Maria Pimentel.
EXAMINADOR	Ing. Edgar Santos.
EVAMINADOR	Ing. Jorge Luis Álvarez Mejía
SECRETARIO	Ing. Pedro Antonio Aguilar Polanco.

HONORABLE TRIBUNAL EXAMINADOR

Cumpliendo con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a consideración mi trabajo de graduación titulado:

REGLAS DEL NEGOCIO EN ARQUITECTURA TRES CAPAS

Tema que me fuere asignado por la Dirección de la Escuela de Ingeniería en Ciencias y Sistemas con fecha agosto de 2001.

Benito Manolo Coti Colop

Guatemala, mayo del 2,003

Ingeniero:
Carlos Azurdia.
Coordinador comisión de tesis.

Estimado Ingeniero:

Por este medio me permito infórmale que he precedido a revisar el trabajo de Graduación titulado *"**Reglas del negocio en arquitectura tres capas**"*, elaborado por el estudiante Benito Manolo Coti Colop, el cual cumple con los objetivos propuestos para su desarrollo.

Sin otro más particular, atentamente.

Sergio Mejía Sánchez
Lic. en Admin. De sistemas de información
Asesor

Guatemala, 07 Noviembre del 2003

Ingeniero
Sydney Alexander Samuels Milson
Decano, Facultad de Ingeniería.

Señor Decano:

De manera atenta me dirijo a usted, para informarle que después de conocer el dictamen del asesor del Trabajo de Graduación Titulado: **REGLAS DEL NEGOCIO EN ARQUITECTURA TRES CAPAS**, elaborado por el estudiante **Benito Manolo Coti Colop**, procedo a la autorización del mismo.

Sin otro particular, me suscribo a usted.

"ID Y ENSEÑAD A TODOS"

Ing. Luis Alberto Vettorazzi España
DIRECTOR ING. EN CIENCIAS Y SISTEMAS

AGRADECIMIENTOS:

A Dios

"Gracias a Dios, que en Cristo siempre nos lleva en triunfo, y que por medio de nosotros manifiesta en todo lugar la fragancia de su conocimiento" Cor 2:14. Gracias señor por darme el privilegió de ser tu hijo, y por enseñarme el camino de la gloria.

A mi madre

Matilde Colop por su amor, por su ayuda, y por haberme mostrado el camino correcto.

A mi esposa

Carmen Santos por haberme motivado a dar los últimos pasos de esta meta alcanzada.

A mis hermanos

Arturo, por enseñarme a sonreír en los momentos difíciles y a ser humilde en los momentos de éxito. Alfredo, por enseñarme a madurar y a ser perseverante para alcanzar el éxito. Mario, por enseñarme a dar en los momentos de fruto y ayudar en los momentos difíciles. Arnoldo, por enseñarme a ser constante y aguerrido para alcanzar mis ideales. Carlos para que le sirva de ejemplo.

A mis amigos

Por el apoyo, orientación y la sabiduría que ellos me proporcionan

Y a la Universidad de San Carlos de Guatemala.

DEDICATORIA:

A mis padres

Marto Coti (QEPD), Matilde Colop, por enseñarme a ser, lo que hoy soy.

A mi esposa

Por su amor incomparable, por su amistad, por ser la otra parte, de mi ser. Mi media naranja no lo es, porque es mi naranjal.

A mis hijos

Axelito y Jenny por ser la razón de mí existir.

A mis hermanos

Por su amistad, su apoyo incondicional en todo momento, y por formar parte de mi vida.

A toda mi familia

Por su apoyo incondicional.

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES

FIGURAS

RESUMEN

En el siguiente trabajo se hace énfasis en el desarrollo de software utilizando reglas del negocio implementadas en arquitectura 3 capas. Se implementan 6 capítulos, los cuales están clasificados de la siguiente manera:

En el capitulo uno hace referencia al modelo clásico de cliente servidor, teniendo así definiciones de la arquitectura de dos capas y la arquitectura de 3 capas la cual se le da mayor realce en este documento para la definición de las reglas del negocio.

El dos, tendrá mayor énfasis en el desarrollo de arquitectura de tres capas, por lo tanto se tiene un capitulo completo sobre esta arquitectura donde se tocaran los puntos conceptuales e importantes de cada una de las capas que la forman.

El tercero, es un capitulo completo donde se implementa todo lo relacionado conceptualmente a las reglas del negocio, desde los conceptos básico, hasta la formulación de las mismas.

Con lo expuesto en los capítulos anteriores, y luego de tener conceptos básicos de la arquitectura tres capas y las reglas del negocio, en el capitulo

cuatro haremos referencia a algunas herramientas de desarrollo, que nos ofrecen en la actualidad la capacidad de soportar esta arquitectura y la metodología de las reglas del negocio, además, se hará énfasis en la implementación de soluciones basadas en DNA que involucra crear aplicaciones divididas en capas funcionales que se comunican entre sí. Este concepto por si mismo no significa nada nuevo, pero DNA provee protocolos estándares e interfaces pre-implementadas que permiten al desarrollador concentrarse en construir la lógica del sistema, sin preocuparse por como las partes se intercomunican.

Luego de finalizar conceptualmente y de conocer una herramienta de desarrollo se utilizará una metodología de desarrollo, para poder cumplir este objetivo se usará una herramienta de modelado UML (*Unified Modeling Language*), concentrándonos en el diseño conceptual, diseño lógico y en el diseño físico. Para diseñar y estructurar los componentes y elementos necesarios que nos permitan dar solución a problemas reales.

Y para finalizar, se tendrá un capitulo en donde se recomienda en que casos utilizar la arquitectura tres capas para el desarrollo de aplicaciones, viendo así los beneficios administrativos, operativos y de desarrollo que esta metodología ofrece.

OBJETIVOS

General

1. Hacer ver a las personas involucradas en el desarrollo de sistema informático, las ventajas administrativas, operativas y de desarrollo que ofrece la definición determinación e implementación de las reglas del negocio en la vida del desarrollo de aplicaciones con arquitectura tres capas.

Específicos

1. Dar a conocer las ventajas que ofrecen las aplicaciones desarrolladas con arquitectura tres capas.

2. Utilizar las ventajas de comunicación que ofrece Internet en aplicaciones desarrolladas en tres capas.

3. Conocer la importancia de definir las reglas del negocio, con todos los individuos que se involucran de una u otra manera en el ciclo de vida de un sistema.

4. Definir e implementar, en forma ordenada las reglas del negocio, en aplicaciones cliente servidor de 3 capas.

INTRODUCCIÓN

La empresa como modelo operativo experimenta permanentes cambios. Actualmente las tendencias reflejan una visión menos operacional y más orgánica del concepto de empresa. La empresa se percibe no como una organización con una cierta estructura de producción sino como un organismo adaptable a toda innovación dentro del ambiente de producción.

Esto requiere una aceleración en los ritmos de estabilización ante la eventual integración de nuevas soluciones o la articulación de políticas alternativas. Esta aceleración exige no sólo un sistema informático altamente integrado al esquema troncal de la empresa, sino también una arquitectura fundada en capas, totalmente funcional que soporte dichos cambios.

Desde un clásico sistema hasta una integración completa de procesos del negocio en un Sistema de Flujo de trabajo, las decisiones sobre adoptar una u otra herramienta como estrategia operativa sólo se descartan a la hora de delimitar las fronteras del negocio. Toda aplicación trata de reflejar parte del funcionamiento del mundo real de la empresa, para ello, es necesario que cada aplicación refleje las restricciones que existen en el negocio dado, de modo que nunca sea posible llevar a cabo acciones no válidas, a las reglas que debe seguir la aplicación, para garantizar esto se introduce una metodología llamada *reglas de negocio*, o *business rules*.

Asimismo el rápido crecimiento de Internet y la evolución de tecnologías para adaptarse a este crecimiento han hecho que los desarrolladores de sistemas cambien el enfoque que tenían de construir y diseñar aplicaciones utilizando la arquitectura 3 capas. Un sistema desarrollado con arquitectura 3 capas ofrece Interoperatividad, Modalidad, Reutilización, Escalabilidad y el uso de Internet, además utiliza la metodología de reglas del negocio la cual es una metodología fundamental que a existido siempre en el desarrollo de un sistema pero que no se le había dado mayor énfasis.

Las reglas del negocio siempre han existido en los sistemas, la diferencia es que no se le había dado mayor énfasis en el desarrollo de una aplicación, hoy en día se toma como una metodología que existe dentro de los sistemas desde los inicios de la aplicación hasta el final de la misma, esta reglas son definidas por todas las personas que forman parte de la vida de un sistema, esto hace que exista mayor comunicación y con mayor frecuencia para así poder definir estas reglas.

La mayoría de empresas, tratan de actualizar el movimiento informático para tener mayores mejoras y gracias al desarrollo de la tecnología y a las nuevas metodologías de desarrollo de aplicaciones informáticos e creído conveniente realizar mi trabajo de tesis titulado: **REGLAS DEL NEGOCIO EN ARQUITECTURA TRES CAPAS**, ya que es un tema muy importante y nuevo en estos días, en donde los sistemas de información han ganado mayor campo y crecimiento acelerado en el desarrollo de nuestras empresas. Ya que la información es lo más valioso hoy en día, en cada una de las empresas que colaboran de una u otra manera en el desarrollo de nuestro país, y del mundo entero.

1. ARQUTECTURA CLIENTE SERVIDOR

1.1 Un poco de historia

Para entender la tendencia descripta sería conveniente explorar muy brevemente el desarrollo de la informatización corporativa. Los procesos contables sirvieron de punto de entrada a los primeros proyectos informáticos. En consecuencia, los departamentos contables sea por hecho o por derecho, se transformaron en reductos de los primeros centros de procesamiento.

Paulatinamente, los sistemas asimilaron una mayor cantidad de responsabilidades convirtiéndose así en grandes aplicaciones monolíticas, bloques de software cimentados sobre otros bloques.

Las exigencias de automatización comenzaron a desbordar las fronteras departamentales en forma de requerimientos de informatización. Surge entonces la figura de Departamento de Tecnología y Sistemas, en un principio, íntimamente asociada a su sector de gestación.

El vertiginoso descenso de los costos de equipamiento, por un lado, y la saturación de requerimientos y diferencias de criterio sobre la prioridad de los

proyectos, sumado a la interposición de trabas burocráticas, provocaron la proliferación de soluciones informáticas departamentales nativas.

Los centros de procesamiento ceden a las exigencias del *downsizing* y la descentralización. Se experimenta un primer grado de distribución de recursos gracias a la instalación de LANs corporativas, la arquitectura de los nuevos desarrollos por antonomasia es el modelo Cliente / servidor. La metáfora que marca el paradigma es la PC-Escritorio: potenciar a la PC cliente con software pesado y por lo tanto con hardware que lo sustente.

Subsiste aún un alto grado de aislamiento entre los sistemas, imposibilitando un aprovechamiento eficiente de los recursos y provocando efectos aún más perniciosos, como la desintegración conceptual en todos sus matices: redefiniciones de términos, aliasing conceptual, ambigüedades, contradicciones dentro del dominio del negocio, superposición de roles y responsabilidades.

En resumen, todos aquellos obstáculos que se ponen de manifiesto cada vez que se decide encarar un relevamiento y que en un importante porcentaje de los casos terminan por boicotear el proyecto. El advenimiento de la integración de procesos de negocios exige una nueva generación de aplicaciones.

Las hipótesis son un paquete de conceptos innovadores: arquitecturas multicapa, núcleos fundacionales, sistemas *Peer-to-Peer* cooperativos, un modelo de interacción homogéneo basado en la metáfora de la Navegación.

Las consecuencias de este enfoque son más que alentadoras: escalabilidad absoluta, procesamiento distribuido, lógica distribuida, información omnipresente, eficiente utilización de los recursos computacionales y dispositivos caros, un punto de control uniforme para la inserción de estrategias operativas.

En definitiva, una plataforma sólida que soporte nuevos desarrollos verticales y que posibilite ciclos de desarrollo evolutivos como los que son de rigor en estos nuevos tiempos.

1.2 Conceptos básicos

1.2.1 Arquitectura cliente servidor

1.2.1.1 Antecedentes

Existen diversos puntos de vista sobre la manera en que debería efectuarse el procesamiento de datos, aunque la mayoría que opina, coincide en que nos encontramos en medio de un proceso de evolución que se

prolongará todavía por algunos años y que cambiará la forma en que obtenemos y utilizamos la información almacenada electrónicamente.

El principal motivo detrás de esta evolución es la necesidad que tienen las organizaciones (empresas o instituciones públicas o privadas), de realizar sus operaciones más eficientemente, debido a la creciente presión competitiva a la que están sometidas, lo cual se traduce en la necesidad de que su personal sea mas productivo, que se reduzcan los costos y gastos de operación, al mismo tiempo que se generan productos y servicios más rápidamente y con mejor calidad.

En este contexto, es necesario establecer una infraestructura de procesamiento de información, que cuente con los elementos requeridos para proveer información adecuada, exacta y oportuna en la toma de decisiones y para proporcionar un mejor servicio a los clientes y ciudadanos.

El modelo Cliente/Servidor reúne las características necesarias para proveer esta infraestructura, independientemente del tamaño y complejidad de las operaciones de las organizaciones públicas o privadas y, consecuentemente desempeña un papel importante en este proceso de evolución.

1.2.1.2 Definición cliente servidor

Cliente: Es el que inicia un requerimiento de servicio. El requerimiento inicial puede convertirse en múltiples requerimientos de trabajo a través de redes LAN o WAN. La ubicación de los datos o de las aplicaciones es totalmente transparente para el cliente.

Servidor: Es cualquier recurso de cómputo dedicado a responder a los requerimientos del cliente. Los servidores pueden estar conectados a los clientes a través de redes LANs o WANs, para proveer de múltiples servicios a los clientes y ciudadanos tales como impresión, acceso a bases de datos, fax, procesamiento de imágenes, etc. Entre las principales definiciones se tiene:

a. Desde un punto de vista conceptual

Es un modelo para construir sistemas de información, que se sustenta en la idea de repartir el tratamiento de la información y los datos por todo el sistema informático, permitiendo mejorar el rendimiento del sistema global de información

b. En términos de arquitectura

Los distintos aspectos que caracterizan a una aplicación (proceso, almacenamiento, control y operaciones de entrada y salida de datos) en el sentido más amplio, están situados en más de un computador, los cuales se encuentran interconectados mediante una red de comunicaciones.

c. IBM define al modelo cliente/servidor

Es la tecnología que proporciona al usuario final el acceso transparente a las aplicaciones, datos, servicios de cómputo o cualquier otro recurso del grupo de trabajo y/o, a través de la organización, en múltiples plataformas. El modelo soporta un medio ambiente distribuido en el cual los requerimientos de servicio hechos por estaciones de trabajo inteligentes o "clientes'', resultan en un trabajo realizado por otros computadores llamados servidores".

El modelo de desarrollo Cliente/Servidor se basa en la aplicación de dos conceptos simples, relacionados con la vida cotidiana. En cualquier tipo de práctica comercial, existen dos figuras relevantes, cada una de las cuales desempeña su propio papel.

Una de estas figuras, El Cliente, se dedica a una determinada actividad comercial, ofrece su propio abanico de productos y desempeña, por tanto, una determinada serie de funciones. Pero esta figura no puede dar servicio a todas

sus funciones sin la existencia de otra, El Servidor, quien proporciona al Cliente las materias primas necesarias para el cumplimiento de sus funciones.

Además, se ha de garantizar que este flujo de peticiones y de respuestas de peticiones se realiza en tiempo real, es decir, que las dos figuras mencionadas han de actuar simultáneamente, de forma que la actividad de uno no se vea perjudicada por la del otro.

El Cliente puede realizar peticiones al Servidor para que le proporcione el material necesario sin interrumpir su actividad.

Si trasladamos el ejemplo anterior, al mundo informático, tenemos que ciertas aplicaciones (Clientes), necesitan de otras aplicaciones (Servidores), para poder llevar a cabo la mayor parte de sus funciones.

Las aplicaciones Clientes realizan peticiones a una o varias aplicaciones Servidoras, que deben encontrarse en ejecución para atender dichas demandas.

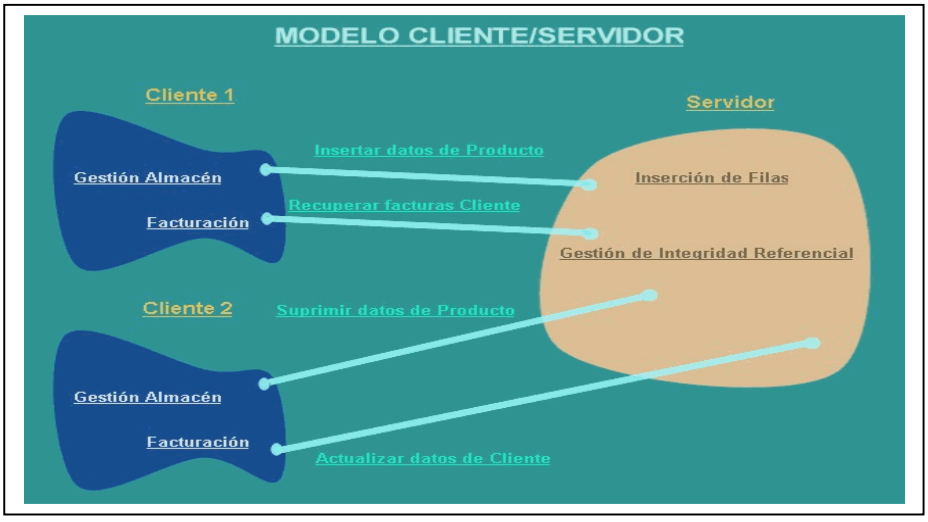
El modelo Cliente/Servidor permite diversificar el trabajo que realiza cada aplicación, de forma que los Clientes no se sobrecarguen, cosa que ocurriría si ellos mismos desempeñaran las funciones que ahora le son proporcionadas de forma directa y transparente.

Por otro lado, con este modelo, todas las tareas que son comunes a distintas aplicaciones Clientes pueden ser implementadas por una única aplicación servidora, que proporcionará los servicios requeridos por todos los clientes.

Llegados a este punto, es necesario aclarar que, si bien el concepto de Cliente y Servidor más ampliamente difundido en el entorno informático alude principalmente al papel desempeñado por las máquinas, en este modelo, tanto el Cliente como el Servidor son entidades abstractas que pueden residir en la misma máquina o en máquinas diferentes.

Recuerde que cuando hablamos del Modelo Cliente/Servidor, nos referimos siempre a un modelo de desarrollo de software. En el siguiente gráfico se muestra de forma clara y sencilla lo expuesto hasta ahora.

Figura 1. Modelo cliente servidor



1.2.2 Cliente servidor de 2 capas

Uno de los objetivos de las aplicaciones de 2 capas es separar la lógica de acceso a los datos de lo que es la interfaz de usuario y trasladarla al servidor. Habitualmente, se implementan servicios como procedimientos almacenados en el sistema gestor de datos; con esto se pretende reducir la carga de los clientes y centralizar las operaciones comunes de acceso a los datos. El Sistema Gestor de Datos también suele incorporar la funcionalidad necesaria para trabajar en entornos multiusuarios. En este modelo intervienen únicamente dos entidades: El Cliente y El Servidor.

El papel de Cliente lo desempeña la aplicación final del usuario, que implementará todas las funciones correspondientes a la lógica de presentación, más algunas de las funciones relacionadas con la lógica del negocio, como pueden ser determinadas validaciones de datos y condiciones de recuperación.

El papel de Servidor lo desempeña el propio SGBD, sobre el cual revertirán todas las funciones correspondientes a la lógica de datos, más las restantes funciones correspondientes a la lógica del negocio, mediante la codificación de Procedimientos Almacenados.

Este es el modelo Cliente Servidor más sencillo y más utilizado habitualmente. En la mayor parte de los casos, el desarrollador de una aplicación de este tipo, desarrolla únicamente la aplicación Cliente y utiliza al

propio motor de Base de datos. Como aplicación servidora, tal manera que no se codifica la aplicación Servidora propiamente dicha.

El mantenimiento de las aplicaciones Cliente que utilizan en este modelo exige un esfuerzo considerable, dado que las reglas del negocio que son implementadas por sí mismas, provocarán la modificación del código de la aplicación en el caso en que éstas varíen.

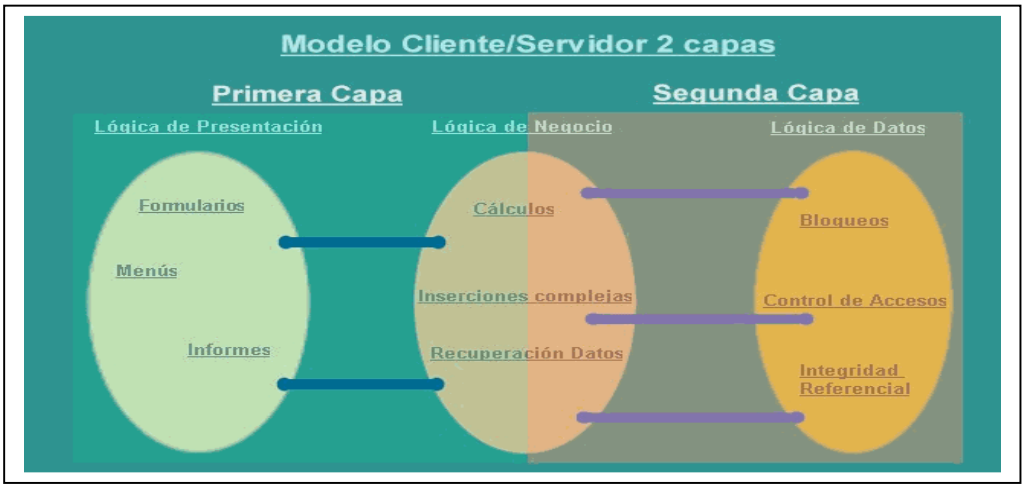
Como hemos dicho anteriormente, el resto de las reglas del negocio se implementan en el servidor de datos, que dispone de un recurso propio, denominado Procedimiento Almacenado. Un Procedimiento Almacenado es una porción de código que reside en la Base de Datos, se compila una sola vez y es compartido por todos aquellos usuarios que gocen de los permisos establecidos con relación a su acceso.

El hecho de codificar ciertas reglas del negocio mediante Procedimientos Almacenados, minimiza el impacto que supone la alteración de dichos procedimientos, ya que no siempre es necesaria la recompilación del código de la aplicación.

Otra consecuencia derivada de dicho modelo es la dependencia de la aplicación final, respecto de la organización de los datos. La aplicación necesita conocer el modelo de datos para poder acceder a él.

Este modelo tiene la desventaja de no ser escalable, pues cada cliente está consumiendo, como mínimo, una conexión con el servidor de datos y, dado que éstas son limitadas, se está restringiendo el número de clientes que pueden coexistir. En el siguiente gráfico se muestra el modelo en cuestión.

Figura 2. Modelo cliente servidor, 2 capas



1.2.2.1 Elementos principales de la arquitectura

Los elementos principales de la arquitectura cliente servidor son justamente el elemento llamado cliente y el otro elemento llamado servidor. Por ejemplo dentro de un ambiente multimedia, el elemento cliente seria el dispositivo que puede observar el vídeo, cuadros y texto, o reproduce el audio distribuido por el elemento servidor. Por otro lado el cliente también puede ser una computadora personal o una televisión inteligente que posea la capacidad de entender datos digitales. Dentro de este caso el elemento servidor es el

depositario del vídeo digital, audio, fotografías digitales y texto y los distribuye bajo demanda de ser una maquina que cuenta con la capacidad de almacenar los datos y ejecutar todo el software que brinda éstos al cliente.

1.2.2.2 Características del modelo

En el modelo Cliente Servidor podemos encontrar las siguientes características: El Cliente y el Servidor pueden actuar como una sola entidad y también pueden actuar como entidades separadas, realizando actividades o tareas independientes. Las funciones de Cliente y Servidor pueden estar en plataformas separadas, o en la misma plataforma.

Un servidor da servicio a múltiples clientes en forma concurrente. Cada plataforma puede ser escalable independientemente. Los cambios realizados en las plataformas de los Clientes o de los Servidores, ya sean por actualización o por reemplazo tecnológico, se realizan de una manera transparente para el usuario final.

La interrelación entre el hardware y el software están basados en una infraestructura poderosa, de tal forma que el acceso a los recursos de la red no muestra la complejidad de los diferentes tipos de formatos de datos y de los protocolos.

- Un sistema de servidores realiza múltiples funciones al mismo tiempo que presenta una imagen de un solo sistema a las estaciones clientes**.** Esto se logra combinando los recursos de cómputo que se encuentran físicamente separados en un solo sistema lógico, proporcionando de esta manera el servicio más efectivo para el usuario final.

- También es importante hacer notar que las funciones Cliente/Servidor pueden ser dinámicas. Ejemplo, un servidor puede convertirse en cliente cuando realiza la solicitud de servicios a otras plataformas dentro de la red.

- Su capacidad para permitir integrar los equipos ya existentes en una organización, dentro de una arquitectura informática descentralizada y heterogénea

Además se constituye como el nexo de unión mas adecuado para reconciliar los sistemas de información basados en mainframes o mini computadores, con aquellos otros sustentados en entornos informáticos pequeños y estaciones de trabajo.

Designa un modelo de construcción de sistemas informáticos de carácter distribuido. Su representación típica es un centro de trabajo (PC), en donde el usuario dispone de sus propias aplicaciones de oficina y sus propias bases de datos, sin dependencia directa del sistema central de información de la

organización, al tiempo que puede acceder a los recursos de este host central y otros sistemas de la organización ponen a su servicio.

En consecuencia, parte del control de las aplicaciones se transfieren del computador central (servidor) a los PCs o estaciones de trabajo (clientes), adquiriendo estas plataformas, entonces, un papel protagonista en conjunto del sistema de información.

1.2.3 Cliente servidor de 3 capas

Este modelo aporta una flexibilidad adicional en la construcción de aplicaciones cuando éstas aumentan su complejidad. Influye tanto en el modelo de aplicación (lógicas de presentación, del negocio y de datos) como en la distribución de los servicios. El modelo conceptual de una aplicación establece sus definiciones, reglas y relaciones así, como su estructura. Hay partes de la *lógica* que residen en el ordenador cliente, normalmente las que se refieren a la interfaz de usuario, mientras que las del negocio y de datos suelen residir en los ordenadores servidores, que proporcionan los mecanismos necesarios para el trabajo en entornos multiusuarios.

En este tipo modelo se aplica íntegramente el modelo de servicios ya que, cada una de las capas se corresponde con cada una de las lógicas descritas.

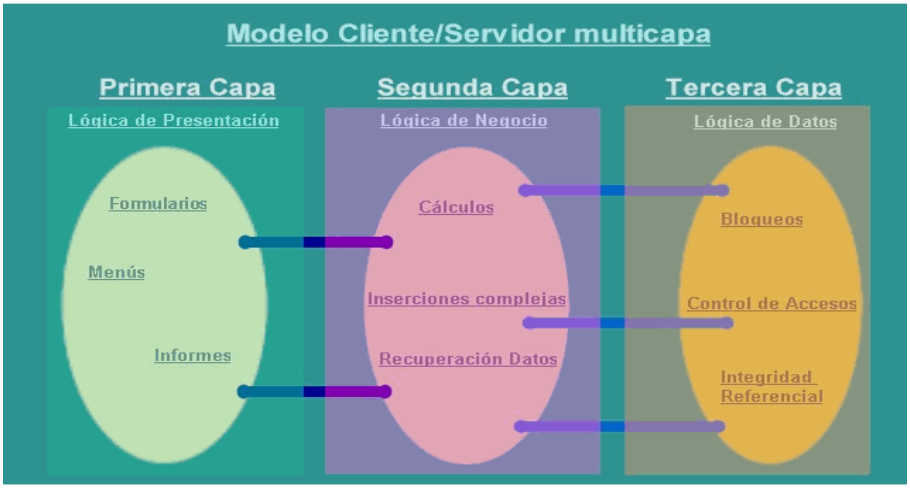
Para llevar a cabo la implementación de un modelo como éste, se hace uso de los mismos recursos descritos en el modelo de dos capas pero se introduce un nuevo tipo de tecnología que permite la construcción de componentes especializados.

Una de las características principales de este modelo reside en la desconexión total entre la lógica de presentación y la lógica de los datos. Las conexiones que se producen, se dan entre las lógicas de presentación y del negocio, y las lógicas del negocio y la de datos.

Este modelo hace que la aplicación final sea completamente independiente del origen de los datos que procesa, tarea que pasa a ser competencia directa del componente especializado.

A pesar de lo comentado anteriormente, no es necesario que las distintas lógicas residan en máquinas diferentes; en la mayoría de los casos, es perfectamente compatible su implementación en la misma máquina, si bien este diseño no es el más habitual. En el siguiente gráfico se muestra el esquema correspondiente al modelo multicapa.

Figura 3. Modelo cliente servidor, 3 Capas



1.2.3.1 Lógica de presentación (*front end*)

Se encarga del GUI que es presentado al usuario. El GUI es la interfase grafica, es decir todas las formas, paginas Web o menús de dialogo, con los cuales interactúa el usuario

Los servicios de presentación proporcionan la interfaz necesaria para presentar información y reunir datos. También aseguran los servicios de negocios necesarios para ofrecer las capacidades de transacciones requeridas e integrar al usuario con la aplicación para ejecutar un proceso de negocios.

Los servicios de presentación generalmente son identificados con la interfaz de usuario, y normalmente residen en un programa ejecutable

localizado en la estación de trabajo del usuario final. Aún así, existen oportunidades para identificar servicios que residen en componentes separados.

El cliente proporciona el contexto de presentación, generalmente un *browser* como *Microsoft Internet Explorer o Netscape Navigator*, que permite ver los datos remotos a través de una capa de presentación HTML, O también una aplicación WIN32 como ser los formularios de *Visual Basic*.

Mediante el uso de componentes, se separa la programación que da acceso a los datos en las bases de datos y aplicaciones desde el diseño y otros contenidos de la página Web.

Esto ayuda a asegurar que los desarrolladores estén libres para enfocarse en escribir su lógica de negocios en componentes sin preocuparse acerca de cómo se muestra la salida. Recíprocamente, esto da libertad a los diseñadores de usar herramientas familiares para modificar la interfaz.

La capa de servicios de presentación es responsable de:

- Obtener información de usuario.
- Enviar la información del usuario a los servicios de negocios para su procesamiento

- Recibir los resultados del procesamiento de los servicios de negocios

- Presentar estos resultados al usuario

a. Presentación distribuida

Se distribuye la interfaz entre el cliente y la plataforma servidora.
La aplicación y los datos están ambos en el servidor.
Similar a la arquitectura tradicional de un Host y Terminales.
El PC se aprovecha solo para mejorar la interfaz gráfica del usuario.

Ventajas
Revitaliza los sistemas antiguos.
Bajo costo de desarrollo.
No hay cambios en los sistemas existentes.

Desventajas
El sistema sigue en el Host.
No se aprovecha la GUI y/o LAN.
La interfaz del usuario se mantiene en muchas plataformas.

b. Presentación remota

La interfaz para el usuario esta completamente en el cliente.
La aplicación y los datos están en el servidor.

Ventajas
La interfaz del usuario aprovecha bien la GUI y la LAN.
La aplicación aprovecha el Host.
Adecuado para algunos tipos de aplicaciones de apoyo a la toma de decisiones.

Desventajas
La aplicaciones pueden ser complejas de desarrollar.
Los programas de la aplicación siguen en el Host.
El alto volumen de tráfico en la red puede hacer difícil la operación de aplicaciones muy pesadas.

1.2.3.2 Capa de negocios: *Business Tier*

Esta capa es la encargada de procesar y validar las reglas del negocio, actúa como un servidor para la capa de presentación y traslada las solicitudes de esta a la capa de datos actuando como cliente de la misma. Así mismo también funciona de la misma manera de la forma inversa.

Los servicios de negocios son el “puente” entre un usuario y los servicios de datos. Responden a peticiones del usuario (u otros servicios de negocios) para ejecutar una tarea de este tipo. Cumplen con esto aplicando procedimientos formales y reglas de negocio a los datos relevantes. Cuando los datos necesarios residen en un servidor de bases de datos, garantizan los servicios de datos indispensables para cumplir con la tarea de negocios o aplicar su regla. Esto aísla al usuario de la interacción directa con la base de datos.

Una tarea de negocios es una operación definida por los requerimientos de la aplicación, como introducir una orden de compra o imprimir una lista de clientes. Las reglas de negocio (b*usiness rules*) son políticas que controlan el flujo de las tareas.

Como las reglas de negocio tienden a cambiar más frecuentemente que las tareas específicas de negocios a las que dan soporte, son candidatos ideales para encapsularlas en componentes que están lógicamente separados de la lógica de la aplicación en sí.

Para ayudar a los desarrolladores a construir la lógica de negocio basado en componentes Windows DNA incluye un conjunto muy poderoso de servicios que se encargan de la comunicación en una aplicación de tres capas. Estos servicios están altamente integrados unos con otros, bajo un sistema operativo, y expuestos de forma única a través de COM.

El nivel de servicios de negocios es responsable de:

- Recibir la entrada del nivel de presentación

- Interactuar con los servicios de datos para ejecutar las operaciones de negocios para los que la aplicación fue diseñada a automatizar (por ejemplo, la preparación de impuestos por ingresos, el procesamiento de ordenes y así sucesivamente).

- Enviar el resultado procesado al nivel de presentación. Algunos de los servicios DNA para la capa de negocios son los siguientes:

- Servicios Web a través de *Microsoft Internet Information Server* (IIS)

- Transacciones y Servicios de Componentes, *Microsoft Transaction Server* (MTS)

- Servicios Asíncronos, *Microsoft Message Queue Server* (MSMQ).

- *Server-side Scripting, via Active Server Pages* (ASP).

a. Lógica distribuida

La interfaz esta en el cliente.
La base de datos esta en el servidor.

La lógica de la aplicación esta distribuida entre el cliente y el servidor.

Ventajas
Arquitectura más simple que puede manejar todo tipo de aplicaciones.
Los programas del sistema pueden distribuirse al nodo mas apropiado.
Pueden utilizarse con sistemas existentes.

Desventajas
Es difícil de diseñar.
Difícil prueba y mantenimiento si los programas del cliente y el servidor están hechos en
distintos lenguajes de programación.
No son manejados por la GUI 4GL.

1.2.3.3 Capa de datos: *Back end*

Esta capa incluye bases de datos y programas que maneja la lectura y escritura sobre las mismas. En esta capa es donde se deben definir los métodos de acceso y políticas de seguridad utilizadas sobre los datos.

El nivel de servicios de datos es responsable de:

- Almacenar los datos

- Recuperar los datos

- Mantener los datos

- La integridad de los datos

Los servicios de datos tienen una variedad de formas y tamaños, incluyendo los sistemas de administración de bases de datos relacionales (SABDs), servidores de correo electrónico y sistemas de archivos.

a. Administración de datos remota

En el cliente residen tanto la interfaz como los procesos de la aplicación.
Las bases de datos están en el servidor.
Es lo que comúnmente imaginamos como aplicación cliente servidor

Ventajas
Configuración típica de la herramienta GUI 4GL.
Muy adecuada para las aplicaciones de apoyo a las decisiones del usuario final.
Fácil de desarrollar ya que los programas de aplicación no están distribuidos.
Se descargan los programas del Host.

Desventajas
No maneja aplicaciones pesadas eficientemente.
La totalidad de los datos viaja por la red, ya que no hay procesamiento que realice el Host.

b. Base de datos distribuida

La interfaz, los procesos de la aplicación, y parte de los datos de la base de datos están
en el cliente.
El resto de los datos están en el servidor.

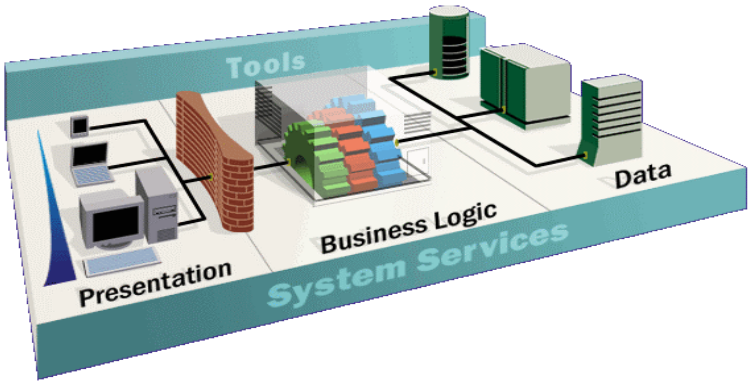
Ventajas
Configuración soportada por herramientas GUI 4GL.
Adecuada para las aplicaciones de apoyo al usuario final.
Apoya acceso a datos almacenados en ambientes heterogéneos.
Ubicación de los datos es transparente para la aplicación.

Desventajas
No maneja aplicaciones grandes eficientemente.
El acceso a la base de datos distribuida es dependiente del proveedor del software
administrador de bases de datos.

1.2.4 Arquitectura:

En la siguiente figura se muestra la arquitectura cliente servidor de 3 capas

Figura 4. Arquitectura cliente servidor, 3 capas



1.2.5 Ventajas y desventajas

a. Ventajas

Uno de los aspectos que más ha promovido el uso de sistemas Cliente/Servidor de n capas, es la existencia de plataformas de hardware cada vez más baratas. Esta constituye a su vez una de las más palpables ventajas de este esquema, la posibilidad de utilizar máquinas considerablemente más baratas que las requeridas por una solución centralizada, basada en sistemas grandes. Además, se pueden utilizar componentes, tanto de hardware como de

software, de varios fabricantes, lo cual contribuye considerablemente a la reducción de costos y favorece la flexibilidad en la implantación y actualización de soluciones.

El esquema Cliente/Servidor de n capas facilita la integración entre sistemas diferentes y comparte información permitiendo, por ejemplo que las máquinas ya existentes puedan ser utilizadas pero utilizando interfaces mas amigables al usuario. De esta manera, podemos integrar PCs con sistemas medianos y grandes, sin necesidad de que todos tengan que utilizar el mismo sistema operacional. Facilita el mantenimiento de las aplicaciones.

Al favorecer el uso de interfaces gráficas interactivas, los sistemas construidos bajo este esquema tienen mayor interacción más intuitiva con el usuario. El uso de interfaces gráficas para el usuario, el esquema Cliente/Servidor de n capas presenta la ventaja, con respecto a uno centralizado, de que no es siempre necesario transmitir información gráfica por la red pues esta puede residir en el cliente, lo cual permite aprovechar mejor el ancho de banda de la red.

Una ventaja adicional del uso del esquema Cliente/Servidor de n capas es que es más rápido el mantenimiento y el desarrollo de aplicaciones, pues se pueden emplear las herramientas existentes (por ejemplo los servidores de SQL o las herramientas de más bajo nivel como los sockets o el RPC). Además se tiene la reutilización de código el cual facilita la creación y el mantenimiento de las aplicaciones.

La estructura inherentemente modular facilita además la integración de nuevas tecnologías y el crecimiento de la infraestructura computacional, favoreciendo así la escalabilidad de las soluciones.

El esquema Cliente/Servidor de n capas contribuye además, a proporcionar, a los diferentes departamentos de una organización, soluciones locales, pero permitiendo la integración de la información relevante a nivel global.

Reparto de la Carga Los componentes desarrollados pueden repartirse en varios servidores, mejorando así el rendimiento de la red.

Accesos mas eficientes a los datos El problema de la limitación de conexiones que admite la base de datos (o para las que se tiene licencia) ahora sólo afecta a los componentes y no a todos los clientes. Además, no es necesario instalar los drivers que se precisan para establecer la conexión con las fuentes de datos, en todos los clientes, sólo se instalarán en aquellos en los que se sitúen los componentes especializados

Mejora en la Seguridad Los componentes de la capa intermedia pueden compartir una seguridad centralizada basada en perfiles de usuarios. Es posible asignar o denegar permisos componente a componente o por paquetes. Esto último simplifica su administración.

Otras ventajas:

Tolerancia de Fallos, Utilización de Internet

b. Desventajas

El esquema Cliente/Servidor de n capas, tiene algunos inconvenientes que se mencionan a continuación:

Además de lo anterior, se cuenta con muy escasas herramientas para la administración y ajuste del desempeño de los sistemas.

En el desarrollo de aplicaciones Cliente/Servidor se deben tener en cuenta diferentes aspectos, que se mencionan a continuación.

Es importante que los clientes y los servidores utilicen el mismo mecanismo (por ejemplo sockets o RPC), lo cual implica que se deben tener mecanismos generales que existan en diferentes plataformas.

Además, hay que tener estrategias pare el manejo de errores y para mantener la consistencia de los datos. La seguridad de un esquema Cliente/Servidor es otra preocupación importante. Por ejemplo, se deben hacer

verificaciones en el cliente y en el servidor. También se puede recurrir a otras técnicas como el encriptamiento.

El desempeño es otro de los aspectos que se deben tener en cuenta en el esquema Cliente/Servidor. Problemas de este estilo pueden presentarse por congestión en la red, dificultad de tráfico de datos, etc.

Un aspecto directamente relacionado con lo anterior es el de cómo distribuir los datos en la red. En el caso de una organización, por ejemplo, éste puede ser hecho por departamentos, geográficamente, o de otras maneras. Hay que tener en cuenta que en algunos casos, por razones de confiabilidad o eficiencia, se pueden tener datos replicados, y que puede haber actualizaciones simultáneas.

A otro nivel, una de las decisiones que deben tomar las organizaciones es la de si comprar o desarrollar los diferentes componentes.

2. REGLAS DEL NEGOCIO (*business tier*)

Definición

Las siguientes definiciones resumen los conceptos relacionados con las reglas del negocio.

2.1.1 Reglas del negocio

Existen dos enfoques entorno a esta expresión; uno de ellos se refiere a la capa lógica en un modelo Cliente servidor de n capas; el otro hace referencia al conjunto de prácticas y/o políticas que, ya sea explícita o implícitamente, definen las tácticas de negocio de la empresa.

Un argumento que define algunos aspectos del negocio. Este debe ser un termino o hecho (descritos bajo una aserción estructural) un constraint (descrito como una acción de aserción), o una derivación. Esto es atómico en que no puede estropearse o descomponerse más allá de las reglas del negocio detalladas.

2.1.2 Argumento regla del negocio

Un argumento declarativo de estructura o constraint el cual las reglas han sido puestas así mismas o en él.

2.1.3 Tipo de expresión formal

Una de las gramáticas formales para representar las reglas del negocio

Argumento regla formal:

Una expresión de una regla del negocio en una gramática formal especifica.

2.1.4 Política

Un argumento general de la dirección de una empresa

Categorías de las reglas del negocio

Una afirmación de una regla del negocio cae dentro de una de estas cuatro categorías:

Definición de los términos del negocio

El elemento básico de una regla del negocio es el lenguaje usado para expresarlo. La misma definición de un término se es una regla comercial que describe cómo las personas piensan y hablan sobre las cosas. Así, definiendo un término está estableciendo una categoría de regla comercial.

Hechos relacionados con términos para nosotros

La naturaleza o estructura operacional de una organización puede ser descrita en términos de los factores el cual relaciona los términos para nosotros. Para decir que un cliente puede colocar una orden es una regla del negocio. Las circunstancias pueden ser documentadas como lenguaje natural o como relaciones, atributos, y estructuras generalizadas en un modelo grafico.

Constraints (Aquí llamados *action assertions*)

Toda empresa reprime el comportamiento de alguna manera, y esta se relaciona estrechamente a constreñimiento en que datos pueda o no puede ponerse al día. Para prevenir un registro de ser este hecho, en cualquier caso.

Derivaciones

Las reglas del negocio (incluyendo las leyes de la naturaleza) se definen cómo el conocimiento en una forma que puede ser transformada en otro conocimiento, posiblemente en una forma diferente.

Formalización de las reglas del negocio

Las reglas del negocio pueden aparecer en cualquier manera. Estas pueden ser descritas en una forma diferente, así como formal e informal. Este es el propósito del Proyecto de las Reglas del negocio para mantener una base declarando las reglas de negocio de una organización formalmente y rigurosamente. Con esta perspectiva, estando debajo de los principios aplican:

Expresión explícita

La declaración de necesidades de las reglas del negocio en una expresión explícita, o gráficamente o como un lenguaje formal (basado en la lógica). Actualmente, las anotaciones modeladas están disponibles para expresar algunos de los tipos de reglas del negocio. Por ejemplo, las reglas estructurales pueden ser representadas por cualquiera de varios diagramas de entidad / relación (o clases de objeto). La replica de elementos puede ser mostrado en forma esencial vía diagramas de flujo o como la vida histórica de los diagramas, hay menos anotaciones disponible, sin embargo, para describir los action assertions. Lo mas notable de estos es el Papel del Objeto que Planea derivado Método de Análisis de Información

Representación coherente

Conceptualmente, la representación de constraints puede ser pensado como una extensión, por ejemplo, el diagrama E-R representa cosas significantes para una empresa, y esto es razonable para los constraints para ser descritos en términos de estas cosas. Todavía, mientras la integración de conceptos es un objeto deseable, logrando una representación integrada cosa que no era el objetivo del proyecto de las reglas del negocio. Se piensa que este documento describe la naturaleza de reglas del negocio, sin tener en cuenta cómo ellos podrían ser representados.

Naturaleza declarativa

Note que una regla del negocio es declarativa, no procesal. Esto describe un estado deseable, posible que esta en cualquier sugerencia, requerido o prohibido. Este puede ser condicional, eso es, si alguno es el caso, algo mas debe o no ser el caso. Esto no es, sin embargo, descrito por los pasos a ser tomados para lograr la transición de un estado a otro, o los pasos a ser tomados para prohibir una transición.

Modelo conceptual de las reglas del negocio

Para expresar los conceptos y estructura de una regla del negocio, este proyecto describe la estructura de una regla del negocio así mismo como un modelo conceptual, en la forma de un diagrama E-R. Esto es, el modelo presentado en este documento define que una regla del negocio es y que tipos de reglas hay.

El modelo de las reglas del negocio esta organizado alrededor de los siguientes temas:

- Los orígenes atómicos de las reglas del negocio.
- Aserciones estructurales (términos y hechos)
- Action Assertions (constraints)
- Derivaciones

Formulando las reglas del negocio

El proceso de identificar las reglas del negocio es a menudo iterativo y heurístico, donde las reglas generales empiezan a declaraciones generales de la política. Incluso si la política es formal y especifica, esta es típicamente descrita en un modelo general e informal, y a menudo permanece para los empleados traducirlo en declaraciones específicas significantes de qué hacer. Todavía, incluso éstas declaraciones más específicas están inmóviles, a

menudo en la naturaleza de business ramblings, sin disciplina. De hecho, estas declaraciones sólo pueden a veces originarse en la política. Más a menudo, ellos se levantan del funcionamiento diario de la organización. Esto implica que estas frases son a veces claras, a veces (quizás deliberadamente) ambiguas, y la mayoría de veces, contienen más de una idea. A pesar de estas limitaciones, los business ramblings son normalmente un análisis de punto de comienzo para derivar declaraciones más formales de reglas del negocio.

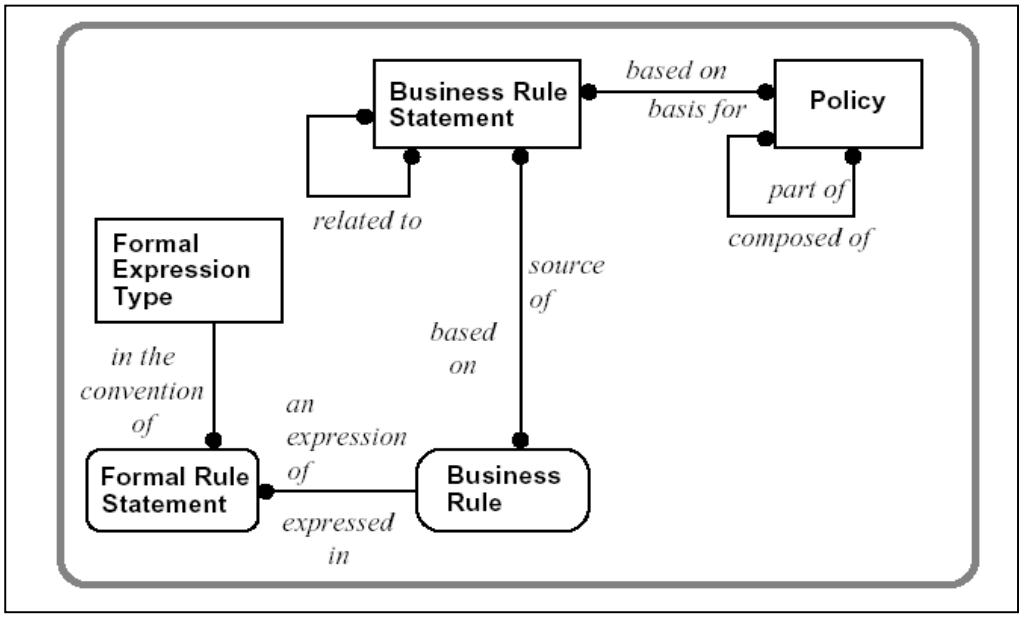
Inicialmente, la asignación del análisis es descomponer estos ramblings compuestos dentro de las reglas del negocio, donde cada regla del negocio es una declaración específica, formal de un solo término, el hecho, derivación, o constraints en el negocio. Entre otras cosas, el analista debe evaluar la estabilidad de la regla. Es él un aspecto fundamental del negocio, o probablemente cambiar en lo ¿mas cerca de (o distante) el futuro? Pueden verse aspectos fundamentales en la infraestructura de la compañía, mientras se manifiestan a menudo reglas de negocio más transeúntes en práctica de un trabajo.

Luego, la tarea es identificar la declaración atómica como la definición de un término, hecho, constraints, o derivación. Términos, hechos, y algunos de los constraints pueden representarse directamente en modelos gráficos. Los constraints restantes y derivaciones deben traducirse en algún otro formalismo. Esto puede ser tan simple como las declaraciones del idioma naturales o puede tener un poco más de expresión formal, como una especificación de idioma de lógica o una anotación gráfica. Cualquiera de la forma, finalmente el diseñador se cobrará con identificar una tecnología apropiada por llevar a cabo las reglas del negocio en un sistema de información.

Orígenes

La figura muestra la primera parte del modelo conceptual de las reglas del negocio. En el texto citado, términos como constraint, regla y regla del negocio tienen que ser usadas en sus términos convencionales del ingles. En las descripciones siguientes, cada uno de estos términos y otros se dan definiciones muy precisas que son críticos al significado del modelo en conjunto.

Figura 5. El origen de las reglas del negocio.



En el modelo anterior, podemos observar una política, un enunciado general de dirección para una empresa. Cada política puede ser compuesta de

políticas más detalladas, es decir, una política detallada puede ser parte de uno o más políticas generales.

Ejemplo:

Una política para nuestro negocio de renta de autos puede ser:
Nosotros solo rentamos carros legales, un camino seguro (que seria condición para nuestros clientes).

Una política puede ser la base de una o más argumentos de las reglas del negocio (business ramblings) así como un argumento de la regla del negocio, puede ser basado en una o más políticas. Un argumento de la regla del negocio es una declarativa argumentada de estructuras o constraint una vez puestos en los negocios en si mismo o se ha puesto en él. Cada regla del negocio puede relacionarse a uno o más reglas del negocio por ejemplo, cada uno de los siguientes podría ser un argumento de reglas del negocio:

- Los carros deben ser chequeados cuando sean regresados después de la renta, y transferidos al dominio.

- Si cualquier luz no esta trabajando, las bombillas deben ser cambiadas. Si las llantas están gastadas, estas también deben de ser cambiadas.

- Bajo cualquiera de las siguientes condiciones el carro debe ser programado para darle servicio o reparación:

    - la distancia en millas acumuladas desde el último servicio sea mayor que 5000.

    - los frenos no funcionen.

    - el tubo de escape este ruidoso o emita gases.

    - tenga cualquier daño el carro (aparte de las mellas superficiales y arañazos), luces o vidrio.

    - hay cualquier gotera fluido significante.

Un argumento de regla del negocio, en torno, puede ser la fuente de uno o más (Atómico) reglas del negocio. Como un argumento de la regla del negocio, una regla del negocio es una declaración donde se define o reprime algún aspecto del negocio, pero (en contraste con un argumento de regla del negocio) no puede estropearse o descomponerse más allá en más reglas del negocio detallados. Cada regla del negocio puede ser basada en uno o más reglas del negocio. Por ejemplo: La distancia en millas acumuladas desde el último servicio sea mayor que 5000.

Es importante notar que las reglas del negocio aplicadas en una empresa, sin tener en cuenta la forma de expréselo. Las reglas del negocio han estado en lugares y compañías que han respondiendo en la vida por largo tiempo antes

de que cualquiera soñara con formalizarlos o dibujarlos. Las reglas del negocio son una realidad subyacente en una organización independiente del esfuerzo de un analista en estructurarlos y describirlos.

Note también que cada regla del negocio puede expresarse en uno o más reglas formales, aunque cada regla formal simplemente debe ser una expresión de una (Atómico) regla del negocio. Una regla formal es una expresión de una regla del negocio en una gramática formal específica. Una regla formal debe ser en la convención de una forma particular, es decir una de las gramáticas formales por representar reglas del negocio.

Tipos de reglas del negocio

Cada regla del negocio debe ser uno de los siguientes:

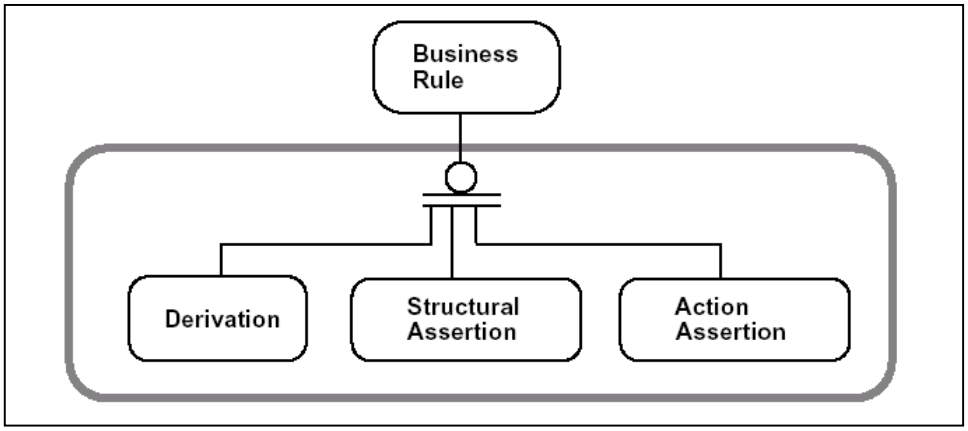
Un Structural Assertion; Una definición conceptual o un argumento de un hecho que exprese algunos aspectos de la estructura de una empresa. Este abarca ambos términos y los hechos configurados de estos términos.

Un Action Assertion: Un argumento de un constraint o condición que limite o controle las acciones de la empresa.

Un Derivation: Un argumento de conocimiento que es derivado en otro conocimiento en el negocio.

La siguiente figura muestra las partes del modelo de las reglas del negocio reflejando estas ideas.

Figura 6. Tipos de reglas del negocio



Antes de seguir adelante con el estudio de la problemática que presenta la implementación de las reglas de negocio, vamos a establecer una clasificación de los tipos en varios grupos.

El primer grupo de reglas de negocio engloba todas aquellas reglas que se encargan de controlar que la información básica almacenada para cada atributo o propiedad de una entidad u objeto es válida: no hay precios de artículos negativos, el sexo de una persona solo puede ser masculino o femenino, una

fecha siempre debe ser una fecha válida (no existe el 30 de Febrero, ¿cierto?), etc. A estas reglas las llamaremos reglas del modelo de datos.

Otro grupo importante de reglas incluye todas aquellas reglas que controlan las relaciones entre los datos. Estas reglas especifican, por ejemplo, que todo pedido debe ser realizado por un cliente, y que el mismo debe estar dado de alta en nuestro sistema: además, una vez que un cliente haya hecho algún pedido, se deberá garantizar que no es posible eliminarlo, a menos que previamente se eliminen todos sus pedidos. Estas reglas constituyen las reglas de relación.

Es frecuente que a partir de cierta información se pueda derivar otra: por ejemplo, el total de un pedido se puede calcular a partir de las distintas líneas que lo componen, mientras que el total de cada línea se puede calcular a partir del número de unidades vendidas y el precio por unidad. Al conjunto de reglas que especifican y controlan la obtención de información que se puede calcular a partir de la ya existente se las llama reglas de derivación.

Otro grupo de reglas de negocio es el compuesto por las reglas de restricción, que restringen los datos que el sistema puede contener. Nótese que este grupo de reglas se solapa en cierto modo con las reglas del modelo de datos, dado que aquellas también impiden la introducción de datos erróneos, como se vio anteriormente. La diferencia estriba en que las reglas de restricción restringen el valor de los atributos o propiedades de una entidad más allá de las restricciones básicas que sobre las mismas existen: por ejemplo, para un saldo existe una regla básica (regla del modelo de datos) que indica que éste debe

ser un número (¡no por obvia es menos regla!), pero además puede haber una regla que indique que el saldo nunca puede ser menor que cierta cantidad tope establecida para cierto tipo de clientes. Esta sería lo que aquí denominamos una regla de restricción, y la diferencia fundamental estriba en el hecho de que este tipo de reglas requiere para su verificación del acceso a otros fragmentos de información, algo que no sucede con las reglas del modelo de datos.

El último grupo de reglas de negocio incluye aquellas reglas que determinan y limitan cómo fluye la información a través de un sistema. Por ejemplo, un cliente puede hacer una petición de análisis a un laboratorio, que anota un encargado: hecho esto, se genera un parte para uno o más analistas, estos realizan las mediciones correspondientes y devuelven los partes con la información pertinente, a partir de la cuál se genera un informe de análisis, que será un análisis válido solo cuando sea firmado por los responsables de garantizar su corrección. A las reglas que indican qué camino recorre la información y obligan a que se sigan solo los caminos válidos se las llama reglas de flujo.

3. REGLAS DEL NEGOCIO EN ARQUITECTURA CLIENTE SERVIDOR

3.1 Modelo de servicios

Un modelo es una vista abstracta que establece las definiciones, reglas y relaciones entre las estructuras relacionadas con la aplicación. Sirve de base para el intercambio de ideas durante el desarrollo lógico de la aplicación y determina cómo será la aplicación resultante.

Cuando alguien habla de una casa, automáticamente asumimos que ésta tendrá un salón, habitaciones, baños, cocina, etc., sin que se nos diga nada más. Aunque la casa no tuviera salón, el modelo nos serviría como punto de partida para “entender” el concepto “casa” y empezar a discutir sobre ella. De igual manera, el modelo de una aplicación nos indica lo que hace una aplicación o. más exactamente, lo que uno cree que debe hacer la aplicación.

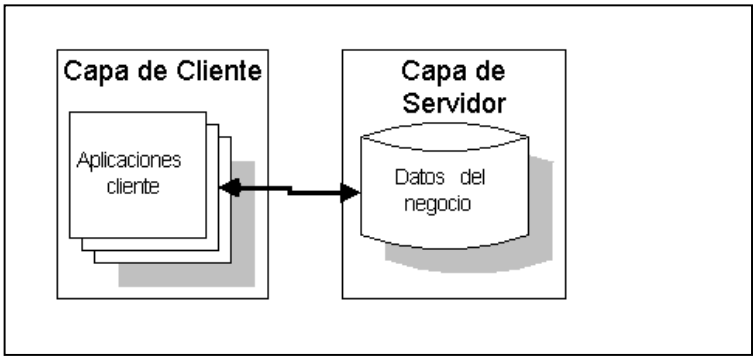
3.2 Capas en un sistema cliente servidor

En un esquema Cliente/Servidor clásico (Figura 3.1) existen dos capas, el cliente y el servidor: éste está ubicado normalmente en otra máquina, y suele ser un gestor de base de datos, como DB2, SQL Server, Oracle, aunque

también puede ser una base de datos más pequeña, como Paradox, dBase, etc., que accedemos directamente desde nuestra aplicación cliente.

Los mejores gestores de base de datos relacionales proporcionan soporte para implementar en ellos bastantes reglas de negocio, mediante el uso de claves primarias, integridad referencial, triggers, etc., mientras que sistemas como dBase y otros apenas proporcionan soporte para reglas de negocio.

Figura 7. Esquema de arquitectura cliente servidor 2 capas



Suponiendo que tengamos la información en un gestor de bases de datos potente, podremos despreocuparnos de llevar a cabo la codificación de numerosas validaciones en nuestras aplicaciones: así, si en la base de datos creamos una regla de integridad referencial que indica que todo pedido pertenece a un cliente, el gestor de base de datos rechazará cualquier intento de almacenar un pedido en el que se nos haya olvidado indicar el mismo.

Cualquier aplicación que acceda a esta base de datos se beneficiará de esta y otras validaciones automáticamente, sin tener que añadir ni una línea de código.

Si estamos utilizando una base de datos menos potente, como dBase, no estamos de suerte. Casi todas las reglas de negocio deberán implementarse dentro de los programas que accedan a la base de datos. Si los programas que acceden a la base de datos son varios, garantizar que en todos ellos se respetan todas las reglas puede llegar a ser muy difícil y engorroso, especialmente si se desarrollan con distintas herramientas.

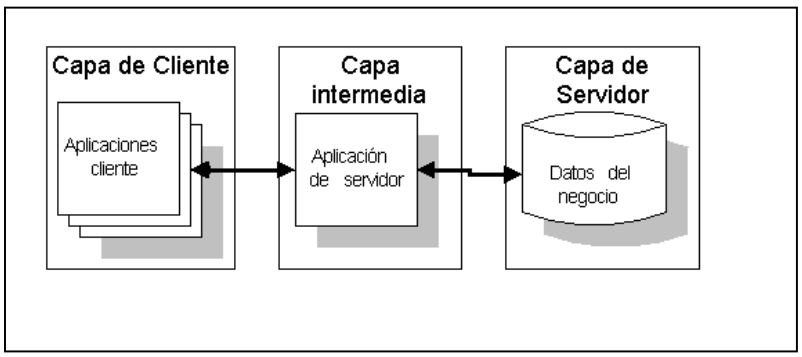
La necesidad de implementar reglas de negocio dentro de las aplicaciones cliente puede surgir también utilizando gestores de bases de datos más potentes. En primer lugar, las bases de datos relacionales son cada vez más potentes, pero no todas las reglas de negocio pueden reflejarse en ellas: por ejemplo, las reglas de flujo son bastante difíciles de implementar dentro de la base de datos, y suelen ser las aplicaciones cliente las que controlan que la información sigue una ruta válida a través del sistema. Sin embargo, muchas de las reglas de negocio pueden reflejarse adecuadamente a nivel de la base de datos con estos gestores.

El problema se agrava cuando la información del negocio se encuentra en distintas bases de datos, gestionadas por distintos gestores, digamos DB2 y Oracle: si, por ejemplo, en DB2 almacenamos la información sobre facturas, etc., y en la base de datos Oracle almacenamos información técnica, como reparaciones llevadas a cabo en la maquinaria, ¿cómo reflejamos que ciertas

facturas corresponden a una reparación, y garantizamos que no podamos relacionar una reparación con una factura inexistente? Evidentemente, aquí no hay manera de establecer una regla de integridad referencial entre tablas almacenadas en dos bases de datos distintas y correspondientes a distintos gestores de base de datos. De nuevo, la solución al problema es implementar el chequeo en cada aplicación cliente, comprobando que exista la factura en la tabla DB2 antes de referenciarla en la tabla de reparaciones Oracle.

Ya que parece que de cualquier modo seremos nosotros mismos los encargados de obligar a que se cumplan algunas reglas de negocio, puede ser conveniente encontrar la manera de centralizar la gestión de estas reglas en un único lugar, de modo que todo el código necesario no se haya de duplicar en cada una de las aplicaciones. La solución puede ser crear una aplicación que se encargue de llevar a cabo estas tareas, de modo que todos los clientes pidan o envíen información a la misma, no al gestor de base de datos en el servidor: a éste solo accederá la nueva aplicación, que conforma una nueva capa dentro de un sistema Cliente/Servidor, la capa intermedia o middle-tier, con lo que nuestro sistema ha pasado de ser un sistema Cliente/Servidor convencional a ser un sistema con tres capas (three-tiered). Conviente apuntar que pueden haber varias de estas aplicaciones, que llamaremos servidores de aplicación, lo que permite distribuir la carga de trabajo.

Figura 8. Arquitectura cliente servidor en tres capas (*three tier*)



Hemos hablado hasta ahora de la capa intermedia como si se tratara de una aplicación cualquiera, pero esto no es así. En los esquemas Cliente/Servidor tradicionales, de dos capas, suele ser el gestor de bases de datos el que proporciona la conectividad, así como capacidades tan fundamentales como el soporte de transacciones.

La introducción de una capa intermedia puede romper con esto, al ser necesario un modo de comunicar las aplicaciones cliente con la aplicación que lleva a cabo las labores de la capa intermedia, siendo ahora ésta la que se aprovecha de las capacidades de conectividad, el soporte de transacciones y las distintas capacidades que proporciona el gestor de base de datos. Solucionar todos los problemas de conectividad, etc., no es tarea fácil, y lógicamente debería utilizarse uno o más productos que solucionen algunos de estos problemas, basados en DCOM, CORBA y tecnologías similares: es el caso de *MIDAS/OleEnterprise*, de *Borland*, que proporciona protección contra la caída de servidores, conectividad e importación de algunas reglas del negocio a los clientes (aunque no soporta transacciones distribuidas), y de *Microsoft*

Transaction Server (que proporciona conectividad y transacciones distribuidas, pero no protección contra caídas de los servidores).

3.3 Ubicación de las reglas del negocio

La decisión de dónde ubicar una determinada regla de negocio dentro de una arquitectura Cliente Servidor de tres capas puede simplificarse mucho si se atiende al tipo de regla de que se trata, utilizando la clasificación introducida más arriba.

Las *reglas del modelo de datos* especifican los valores válidos de cada atributo de las diversas entidades que se almacenan, lo que simplificando es lo mismo que decir los valores válidos para cada campo de cada tabla. Estas reglas deben, a ser posible, reforzarse en el servidor: esto se hace en primer lugar escogiendo correctamente el tipo de los campos de cada tabla (no almacenar una fecha en un campo de tipo cadena si nuestra base de datos dispone de campos del tipo fecha), y donde el servidor lo soporte, mediante restricciones (por ejemplo, en *Interbase* es posible especificar mediante *CHECK* diversas condiciones que debe verificar un dato para poder almacenarse en un campo), etc. El hacer esto así proporcionará mayor robustez a la base de datos.

Como complemento de esto, sin embargo, se debe implementar estas validaciones también a nivel de cliente, por una simple razón: evitar trabajo y esperas innecesarias a los usuarios. Todos hemos rellenado algún formulario HTML varias veces hasta que por fin los datos introducidos son válidos: esto es

así porque en un formulario HTML raramente se lleva a cabo la validación de los datos en el cliente, por lo que la información no se chequea hasta que llega al servidor, debiendo el usuario en el cliente esperar a cualquier mensaje de éxito o error. Lo mejor sería no tener que llevar a cabo ningún tipo de programación, y que las reglas del modelo de datos se pudieran importar del servidor al cliente, haciéndose en el mismo gran parte de los controles necesarios para garantizar la validez de la información introducida por el usuario: esto no es muy difícil, dado que son chequeos relativamente sencillos, al afectar solo a un campo, y de hecho algunos productos, como *OleEnterprise* de *Borland*, incorporan la posibilidad de importar estas reglas del servidor.

Por lo que respecta a las reglas de relación, el lugar más adecuado para implementarlas es, sin lugar a dudas, el servidor. La mayor parte de los gestores de base de datos proporcionan integridad referencial y los mecanismos necesarios para implementar fácilmente estas reglas, y esto proporciona una robustez enorme a la base de datos. Además, el hecho de que los datos necesarios para verificar si se respetan las relaciones residan en la misma base de datos/máquina hace que estas verificaciones sean muy rápidas, mientras que si el chequeo se hace en el cliente se incrementará el tráfico de red. El problema lo encontraremos si el negocio está distribuido en varias bases de datos: si ese es el caso, será necesario implementar alguna de estas reglas en la capa intermedia, y que ésta resida lo más cerca posible de las bases de datos, a ser posible en la misma máquina o red local.

Las reglas de derivación pueden variar mucho en complejidad: la información derivada más simple, como calcular el total de una línea de pedido, puede calcularse a partir de otros campos del mismo registro. Dado que no se

requiere información adicional, y en general toda la información de un registro viaja a la vez al cliente, calcular esta información en el mismo es trivial, y no resulta gravoso. Ahora bien, si lo que se desea es calcular la suma de todos los pedidos de un cliente desde 1960, el volumen de información necesaria para calcular el total es enorme, y hacerla viajar a través de la red no es recomendable. Lo ideal será implementar el cálculo de esta información de modo que se ejecute en el servidor, posiblemente mediante un procedimiento almacenado, o al menos mediante una sentencia SQL de agregación, con lo que la única información que viajará hasta el cliente será el total, no todos los registros necesarios para calcularlo.

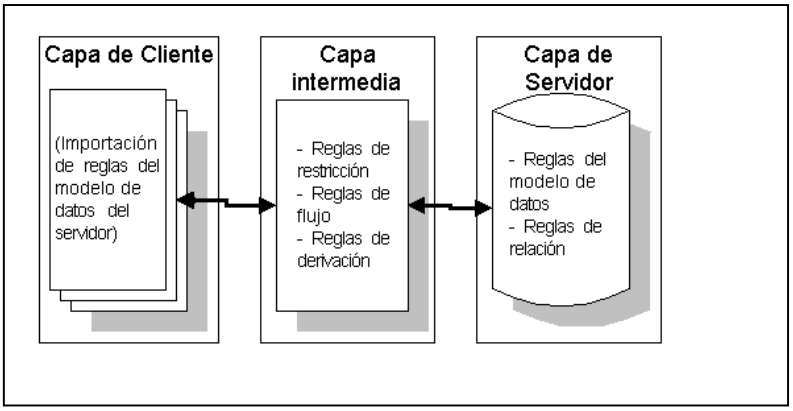
Sin embargo, SQL, o las extensiones de SQL que proporcionan los distintos gestores de base de datos, no es un lenguaje de propósito general, por lo que puede ser difícil o imposible implementar ciertas reglas de derivación. La solución ideal puede pasar por implementar las reglas de derivación en la capa intermedia: si ésta reside en la misma máquina que el servidor, la información no deberá viajar por la red, y aunque el cálculo no será tan rápido como si lo hace el gestor, puede ser suficiente. Si tenemos los datos distribuidos en diversas bases de datos, ésta será la única solución para implementar muchas de las reglas de derivación.

Por otro lado, las reglas más sencillas pueden también implementarse en la capa intermedia, de modo que tengamos las reglas centralizadas en una única aplicación, no desperdigadas por diversas aplicaciones, no importa lo sencillas que sean.

Las reglas de restricción deben implementarse en el servidor, o en la capa intermedia. Dado que estas reglas contemplan restricciones en los datos que dependen casi siempre de información presente en varias tablas, llevar a cabo el control en el cliente puede implicar cierto tráfico de red, por lo que es más conveniente situar la implementación de la regla más cerca de los datos. El lugar ideal podría ser el servidor, pero aquí nos encontramos con las limitaciones del SQL de los gestores de base de datos, y con el posible problema del acceso a diversas bases de datos, por lo que ubicar estas reglas en la capa intermedia puede ser de nuevo una buena solución.

Por último, quedan las reglas de flujo: estas reglas son excelentes candidatas a ser implementadas en la capa intermedia, dado que su complejidad suele ser bastante grande, lo que las hace inmanejables por un gestor de bases de datos.

Figura 9. Ubicación de las reglas de negocio dentro del esquema de tres capas



3.3.1 Lógica de presentación (capa cliente)

Esta lógica es la responsable del control de todos los aspectos relacionados con la interacción entre el usuario y la aplicación. Para llevar a cabo esta tarea de control, es necesario conocer qué tipos de usuarios utilizarán la aplicación, qué actividades tiene que realizar y, teniendo en cuenta estos datos, cuáles son los mejores estilos para que esos usuarios realicen sus tareas.

En esta lógica se engloban todas las tareas que deben ser realizadas por la parte Cliente del modelo general.

Con el fin de independizar en la medida de lo posible la interfaz de usuario de las características propias de los procesos, debemos tener presente que la codificación de las tareas asociadas a esta lógica consiste, principalmente, en la llamada a procesos independientes situados en las otras lógicas, cuya ejecución es totalmente transparente.

Si en la capa que implementará la lógica de presentación no incluimos lógica del negocio ni accesos directos a datos, conseguiremos que esta capa sea inmune a los cambios introducidos en los procedimientos de la empresa, así como a los cambios de los sistemas de gestión de datos utilizados.

3.3.2 Lógica de negocio (capa intermedia)

Es la lógica de la aplicación que controla la secuencia de acciones y fuerza el cumplimiento de las reglas del negocio propias de cada empresa; además, asegura la integridad de las transacciones de las operaciones necesarias que haya que realizar para que se cumplan dichas reglas. La lógica del negocio también transforma una fila de datos en información útil para el usuario mediante la aplicación de las reglas apropiadas.

El objetivo que debe cumplir esta lógica (si está bien diseñada) es el de aislar las reglas del negocio, así como las transformaciones de datos de los consumidores (usuarios y otros componentes de esta misma capa) y de los sistemas de gestión de datos. Este aislamiento tiene las siguientes ventajas:

- Flexibilidad a la hora de decidir cómo y dónde situar el código de esta lógica: en componentes dentro de una aplicación servidora; en procedimientos almacenados, dentro del sistema gestor de datos; o incluso en el cliente.

- La habilidad de colocar distintas interfaces de usuario para un mismo un conjunto estándar de reglas de negocio. Por ejemplo, el conjunto de reglas que define las operaciones realizables con los clientes puede implementarse como un sólo componente que se ejecuta en un servidor. Los servicios que ofrece este componente pueden utilizarse desde una macro que se ejecute dentro de Microsoft Office, desde

una aplicación desarrollada con Visual Basic o desde páginas HTML vistas desde Internet Explorer.

- Facilita el mantenimiento de las reglas del negocio y de su lógica, aislando los cambios de las interfaces de los usuarios y de los datos.

- La habilidad para sustituir el código de estas reglas, de forma que, aunque el conjunto de reglas que se encuentra dentro de un conjunto de servicios del negocio varíe de un país a otro, las interfaces de esos servicios pueden permanecer constantes.

Estos procesos, dada su naturaleza, pueden ser cambiantes en cuanto a su construcción, pero no en cuanto a su funcionalidad. Al regirse por directrices empresariales, éstas podrían cambiar atendiendo a razones internas, sin variar necesariamente la funcionalidad que proporcionan. Tomemos como ejemplo tomemos el cálculo de una nómina. Dicho cálculo se basa en algunos preceptos oficiales que deben ser respetados, aunque la compañía también tiene potestad para introducir cambios que influirán en dicho cálculo. El proceso que realiza el cálculo en cuestión proporciona una funcionalidad concreta y específica: el cálculo de la nómina, pero su codificación dependerá de las directrices de la compañía. Es posible que la compañía establezca un porcentaje de productividad dependiendo de la categoría del empleado que influirá en el importe total de la nómina, pero también es perfectamente posible que, en un futuro, introduzca nuevos procedimientos para incentivar; esto supondría un cambio interno en el proceso de cálculo, pero no en su funcionalidad.

Por cierto, pueden existir distintas aplicaciones de usuario que soporten el cálculo de la nómina como una más de sus funcionalidades, en la mayoría de los casos, esto supone un gran ahorro de tiempo de mantenimiento de procesos, ya que éstos son independientes, de forma que toda aplicación que haga uso de esos procesos, disfrutará de las respectivas actualizaciones sin necesidad de alterar ni su código ni su fichero ejecutable.

3.3.3 Lógica de datos (capa de servidor)

En este conjunto entran los procesos encargados de la gestión de los datos propiamente dicha, es decir, los procesos encargados del mantenimiento de los datos, de garantizar las reglas de integridad referencial establecida, así como de la gestión de las transacciones. Estas tareas son realizadas, generalmente, por un Sistema de Gestión de Bases de Datos Relaciónales.

No hay una única posibilidad a la hora de distribuir las reglas de negocio dentro de un esquema Cliente/Servidor. Sin embargo, sí hay ciertas pautas que se pueden tener en cuenta a la hora de tomar una decisión, basadas en una clasificación de las reglas de negocio aquí expuestas: en general, lo más recomendable suele ser implementar todas las reglas de negocio relativas al modelo de datos y las relaciones en el servidor, dado que los modernos servidores suelen llevar a cabo estas tareas muy eficazmente.

Por lo que respecta al resto de las reglas, la mejor solución suele ser implementarlas en la capa intermedia: si tuviésemos que acceder a varias

bases de datos, habríamos de migrar aquí algunas de las reglas que de otro modo irían al servidor, especialmente las de relación. Si bien el tráfico en la red se incrementará al utilizar una capa intermedia, este puede quedar aliviado haciendo que ésta resida en la misma máquina que el servidor de datos, o al menos dentro de la misma red local. La figura 3.3 muestra gráficamente la ubicación recomendable de las reglas de negocio dependiendo de su tipo (para acceso a una única base de datos).

Por último, vale la pena resaltar la conveniencia de implementar las reglas del modelo de datos también en el cliente, para hacer fluida la interacción con el usuario, siendo lo ideal importarlas dinámicamente del servidor de base de datos. Por lo demás, la implementación de reglas en las aplicaciones cliente puede dar lugar a muchos problemas, tanto de velocidad como de portabilidad y fiabilidad, al tener que reflejar una y otra vez las mismas reglas en distintas aplicaciones, quizá desarrolladas con distintos lenguajes y ejecutándose bajo distintos sistemas operativos.

3.4 Interfase de conexión

Dentro de las interfases de conexión para el desarrollo de aplicaciones con arquitectura cliente servidor tenemos:

Sockets o RPC de OSF

DCOM de Microsoft

CORBA de OMA

Soluciones especificas que proporcionan algún tipo de transformación de datos

3.5 Selección de una interfase de conexión

Para poder seleccionar una interfase de conexión dentro de nuestra aplicación desarrolla con arquitectura cliente servidor debemos de tomar en cuenta:

La disponibilidad para nuestro lenguaje

Seguridad

Ancho de Banda consumido

Rendimiento de la lógica

3.6 Infraestructura de la aplicación

El servidor proporciona una infraestructura o funcionalidad centralizada para el trabajo en entornos multiusuarios. Dicha infraestructura constituye el

software que soporta accesos concurrentes a servicios compartidos, normalmente a la lógica del negocio y a la de datos. Por ejemplo, el acceso a los procedimientos almacenados y a los datos de un servidor de datos son controlados por un Sistema de Gestión de Bases de Datos Relaciónales (SGBDR). Dicho software trabaja en conjunción con el S.O. para proporcionar la funcionalidad necesaria en el acceso concurrente a los datos.

Microsoft Transaction Server (MTS) es un software que trabaja en conjunción con el S.O. para aportar los mecanismos necesarios para el trabajo concurrente a los servicios de la capa del negocio (que se implementan como componentes). Estudios recientes indican que entre el 30 y el 40% del presupuesto de los departamentos de informática se destinan al desarrollo de infraestructuras como éstas. MTS reduce este coste ya que no hay que desarrollarla.

3.7 Ventajas

La metodología de las reglas del negocio en arquitectura cliente servidor nos ofrece las mismas ventajas que ofrecen las aplicaciones cliente servidor clásicas, pero además nos ofrecen:

Soporte Multilenguaje Los componentes pueden desarrollarse utilizando distintos lenguajes de programación

Centralización de Componentes Los componentes pueden agruparse y situarse de una manera centralizada, lo que facilita su desarrollo y posterior distribución

Reparto de la Carga Los componentes desarrollados pueden repartirse en varios servidores, mejorando así el rendimiento de la red.

Accesos mas eficientes a los datos El problema de la limitación de conexiones que admite la base de datos (o para las que se tiene licencia) ahora sólo afecta a los componentes y no a todos los clientes. Además, no es necesario instalar los drivers que se precisan para establecer la conexión con las fuentes de datos, en todos los clientes, sólo se instalarán en aquellos en los que se sitúen los componentes especializados

Mejora en la Seguridad Los componentes de la capa intermedia pueden compartir una seguridad centralizada basada en perfiles de usuarios. Es posible asignar o denegar permisos componente a componente o por paquetes. Esto último simplifica su administración.

Otras ventajas

Modularidad / Interoperatividad, Facilidad de Mantenimiento, Reutilización de código, Procesamiento Distribuido, Tolerancia de Fallos, Utilización de Internet.

4. HERRAMIENTAS COMERCIALES, PARA EL DESARROLLO DE APLICACIONES CLIENTE SERVIDOR

La evolución de la arquitectura Cliente/Servidor, el crecimiento de Internet y la continua innovación en el hardware y el software han proporcionado potentes y sofisticadas aplicaciones para los usuarios. Con todos estos adelantos, los problemas han recaído en los desarrolladores, los vendedores de software y los usuarios:

Aumenta la complejidad y tamaño de las aplicaciones, aumenta el tiempo de desarrollo, se hace más difícil y costoso el mantenimiento; los riesgos se disparan cuando se pretende añadir funcionalidad a ese software.

Las aplicaciones vienen prefabricadas con muchas características, la mayoría de las cuales no pueden ni suprimirse, ni actualizarse independientemente, ni sustituirse por alternativas.

Estas aplicaciones son difíciles de integrar porque los datos y la funcionalidad de una aplicación no están disponibles para otras, aunque las aplicaciones se hubiesen desarrollado con el mismo lenguaje de programación y se ejecutasen en el mismo ordenador.

La solución es un sistema en el que los desarrolladores creen componentes de software reutilizable. Un componente es un trozo de código reutilizable que puede integrarse con otros componentes de otros vendedores o desarrolladores con relativa facilidad. Así, un componente puede tratar una petición de un cliente (buscar un registro específico en una base de datos, por ejemplo) y devolver el resultado al cliente.

A continuación se listan, unos de los elementos más importantes del conjunto de herramientas disponibles en el mercado y en la actualidad, para el desarrollo de aplicaciones cliente servidor basadas en las reglas del negocio.

4.1 Herramientas de desarrollo

4.1.1 Herramientas para la capa de presentación

Internet Explorer
Visual Basic .NET
Visual Basic 7
Delphi3
Delphi4
Louis

4.1.2 Herramientas para la capa de negocios

Visual Basic .NET
Visual Basic 7
Visual InterDev
Visual C++
Biztalk 2000
Windows 2000 Server Family
Delphi3
Delphi4
Louis

4.1.3 Herramientas para la capa de Datos

SQL Server
Oracle
Dbasic
Informix
Etc.

4.1.4 Descripciones a groso modo de algunas herramientas

Delphi 3

- Basado en DCOM de Microsoft
- Reconvertir aplicaciones Delphi exige un gran esfuerzo
- El reaprovechamiento de las aplicaciones antiguas depende de que exista un interfaz DCOM con las mismas

Delphi 4

- Reconvertir aplicaciones Delphi exige un enorme esfuerzo
- El reaprovechamiento de las aplicaciones antiguas depende de que exista un interfaz CORBA, DCE o DCOM con las mismas
- Interprise proporciona puentes
  DCE ←→ CORBA
  DCOM ←→ CORBA

Louis

- Basada en TCP/IP
- Disponible para: Unix, w3.11, w95, NT, AS/400
- El interfaz puede estar escrito en: Java / HTML, VB, Delphi, Cobol

Otras herramientas

GeneXus
Prolifics
GemStone
Javaa

- Suelen centrarse en Java, RMI, CORBA

- Suelen olvidar COBOL, RPG

Etc.

4.2 Determinación de lenguaje de programación apropiado

A la hora de evaluar el lenguaje a utilizar en la etapa de desarrollo, se consideraron como prioritarios ciertos requisitos de calidad, a saber:

- Simplicidad del lenguaje.
- Naturalidad para reflejar un diseño.
- Flexibilidad ante eventuales cambios.
- Soporte a desarrollo en grupo.
- Eficiencia.
- Robustez.

Actualmente es impensable el desarrollo de un modelo complejo sin la utilización de la tecnología de objetos.

Por otro lado era necesario encontrar un lenguaje que brindara una preforma adecuada a los exigentes requerimientos a los que sería sometido, así como también disponer de una escalabilidad muy grande.

También debe tenerse en cuenta la facilidad y potencia del lenguaje, ya que cuando se trata de Java BA de un framework, es fundamental brindar una herramienta de extensión fácil y potente, que asegure a su vez la disminución de los costos necesarios para lograr ese objetivo.

Una de las herramientas de desarrollo, que esta ganando gran parte de terreno en el desarrollo de aplicaciones cliente servidor es el lenguaje Java. La potencia del lenguaje Java podría condensarse en seis argumentos:

- Lenguaje Orientado a Objetos razonablemente puro.
- Portabilidad. (Virtual Machine).
- Gestión de Memoria Transparente.
- Soporte a Programación Concurrente (Multithreading.)
- Soporte de Networking desde el lenguaje.
- Performance (JITa - Hot Spota).

4.3 DNA (Microsoft)

La arquitectura Windows para aplicaciones distribuidas sobre Internet (Windows DNA) es un marco de trabajo para construir una nueva generación de

soluciones de cómputo que incluyan los mundos de la computación personal e Internet. Windows DNA es la primera arquitectura de aplicación que contiene e integra totalmente tanto los modelos Web de desarrollo de aplicaciones para cliente como para servidor.

Al utilizar el modelo Windows DNA, se construyen aplicaciones de negocios modernas, escalables, multi-capas, para ser ejecutadas sobre cualquier tipo de red. Las aplicaciones Windows DNA mejoran el flujo de información dentro y fuera de la organización, son dinámicas y flexibles al cambio en la medida en que cambian las necesidades del negocio, se integran fácilmente con los sistemas y datos existentes. Como las aplicaciones Windows DNA impulsan servicios de plataforma Windows profundamente integrados que trabajan juntos, las organizaciones pueden enfocarse a entregar soluciones de negocios en vez de ser integradoras de sistemas.

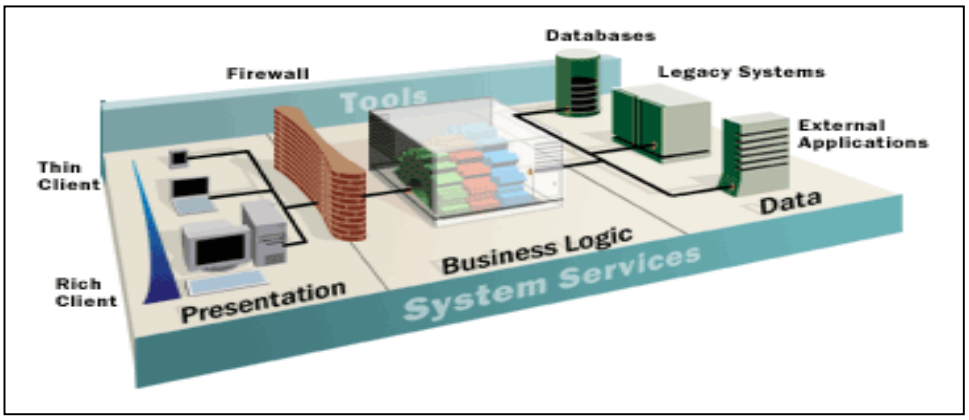
Windows DNA incluye productos y servicios para ayudar a los desarrolladores a implementar los servicios de las aplicaciones de tres capas basadas en componentes. La arquitectura de tres capas es recomendada para construir aplicaciones distribuidas escalables. Los componentes son recomendados como una vía para construir soluciones flexibles y de fácil mantenimiento.

Una de las principales ventajas de Windows DNA es Internet, que ha cambiado dramáticamente el panorama de la computación. Siete años atrás, el proceso de desarrollo de aplicaciones ejecutado por una persona en una computadora era relativamente informal. En contraste, algunas de las

aplicaciones más poderosas de nuestros días soportan miles de usuarios simultáneos, necesitan estar corriendo las 24 horas del día y deben ser accesibles desde una amplia variedad de dispositivos, desde computadoras portátiles hasta estaciones de trabajo de alto desempeño.

Para satisfacer estos requerimientos imperativos, los desarrolladores de aplicaciones necesitan adecuar herramientas de planificación y guías de cómo incorporar las tecnologías apropiadas.

Figura 10. Arquitectura Windows DNA



Principios DNA

La arquitectura de Windows DNA está diseñada para maximizar en la aplicación lo siguiente:

- La autonomía.
- La confiabilidad.
- La disponibilidad.
- La escalabilidad.
- La interoperabilidad.

4.3.1 Autonomía

La autonomía de la aplicación se refiere a la habilidad de una aplicación para gobernar sus recursos críticos. Los recursos críticos son los recursos preciosos requeridos por una aplicación para funcionar confiablemente como una entidad independiente. Las conexiones a las SABDs, las conexiones a los mainframes y las transacciones son todas, ejemplo de recursos críticos. La autonomía de la aplicación es supuestamente uno de los aspectos más importantes del diseño de las aplicaciones para Windows DNA, y es también una de las diferencias más importantes entre los diseños cliente / servidor de dos y de tres niveles.

En un diseño cliente / servidor de dos niveles, los clientes tienen acceso directo a los recursos críticos de la aplicación y son libres de utilizar estos recursos cuando así lo deseen.

Debido a que los clientes tienen acceso directo a los recursos críticos de la aplicación hay poco o nada que la aplicación pueda hacer para protegerse a sí misma del comportamiento inesperado o malintencionado, lo que compromete la estabilidad general de la aplicación. Por ejemplo, un solo cliente que no se comporte adecuadamente puede intencionadamente acabar con los recursos críticos de una aplicación en un intento para prevenir a otros clientes de hacer su trabajo. Un ataque como éste puede convertir a las aplicaciones indefensas en inútiles.

Las aplicaciones de Windows DNA, por otro lado, nunca permiten a los clientes tener acceso directo a los recursos críticos. En lugar de ello, los clientes hacen peticiones de componentes especiales de su nivel de confianza llamados ejecutantes que realizan las operaciones de negocios para los que la aplicación fue diseñada.

Al forzar a los clientes a hacer peticiones a los ejecutantes para que hagan las operaciones de negocios, utilizando los recursos críticos en maneras bien definidas y confiables, las aplicaciones de Windows DNA se quedan en completo control de sus recursos críticos, lo cual finalmente incrementa la estabilidad general de la aplicación.

Debido a que los ejecutantes son componentes confiables tienen acceso directo a los recursos críticos de la aplicación, lo que significa que deben poner mucha atención a la manera en que los recursos críticos son utilizados. Antes de que un ejecutante haga alguna operación de negocios en favor de un cliente, debe autentificar la identidad del cliente que hace la petición, validar que el

cliente tiene la autorización apropiada para ejecutar la operación de negocios pedida e inspeccionar la petición del cliente para ver si tiene una apropiada sintaxis y para validar los datos.

4.3.2 Confiabilidad

La confiabilidad se refiere a la habilidad de una aplicación para proporcionar resultados exactos. Una aplicación no es confiable si regresa resultados inexactos. Sin embargo, asegurarse que los resultados sean exactos en un ambiente multiusuario es muy difícil. Para asegurar resultados exactos, se deben hacer las operaciones de negocios como parte de una transacción administrada por el Microsoft Transaction Server (MTS). Las transacciones aseguran que el estado de las transformaciones sean Atómicas, Consistentes, Aisladas y Durables (Atomic, Consistent, Isolated, Durable*;* ACID).

- Operaciones atómicas son operaciones que se completan en su totalidad o no se completan en absoluto. La transacción debe haber sido exitosa para que el estado de transformación sea exitoso, de otro modo el estado de la transformación falla, y el sistema es regresado a su último estado conocido.

- Transformaciones consistentes preservan la integridad interna de los recursos involucrados. Por ejemplo, el borrar registros de una tabla primaria viola la integridad referencial de la base de datos si hay registros relacionados que concuerden.

- Transformaciones aisladas parecen ocurrir serialmente, una detrás de otra, creando la ilusión de que ninguna transformación está siendo ejecutada al mismo tiempo.

- La durabilidad se refiere a la habilidad para almacenar los resultados de una transformación de estado, usualmente en un disco, de tal modo que los resultados de una transformación puedan ser recuperados en caso de una falla del sistema.

Debido a que las transacciones bloquean los registros para asegurar el comportamiento ACID, deben ser considerados recursos críticos, lo cual significa que los clientes nunca deben de tener acceso a ellos directamente. Así que para mantener la autonomía de la aplicación, las aplicaciones de Windows DNA deben tratar a las transacciones como recursos críticos.

4.3.3 Disponibilidad

La disponibilidad se refiere a la cantidad de tiempo que una aplicación es capaz de dar servicio confiablemente a las peticiones del cliente, y es importante debido a que una aplicación solamente es útil cuando está disponible para dar servicio a las peticiones de los clientes.

La disponibilidad de la aplicación es dependiente de muchas cosas que están más allá del control del desarrollador, cosas como disponibilidad de

hardware, (controlador de disco, tarjetas de red, controladores, y así sucesivamente), disponibilidad de software (RDBMSs, servidores de Web, sistemas de espera, y así sucesivamente) y disponibilidad de red.

Las aplicaciones de Windows DNA pueden simular una disponibilidad aumentada de la red al utilizar los servicios del Microsoft Message Queue (MSMQ). El MSMQ proporciona funcionalidad de almacenamiento y redirección que permita que los mensajes estén almacenados en la máquina local, siempre que la cola destino no pueda ser alcanzado debido a la falta de disponibilidad de la red. Una vez que un mensaje ha sido enviado al MSMQ para su envío, el MSMQ constante y repetidamente intentará redireccionar cada mensaje a su cola destino, hasta que el mensaje ya sea que llegue a su destino o que el tiempo de espera de entrega espire.

4.3.4 Escalabilidad

La escalabilidad es la meta utópica del crecimiento lineal del rendimiento al agregar recursos adicionales, y es lo que le permite a una aplicación soportar desde 10 usuarios, hasta decenas de miles de usuarios, simplemente agregando o quitando recursos como sea necesario para escalar la aplicación.

Para incrementar la escalabilidad, los desarrolladores de aplicaciones de Windows DNA deben concentrarse en mantener los tiempos de adquisición de recursos y de uso de recursos tan bajos como sean posibles. El Microsoft

Transaction Server permite compartir recursos entre usuarios reutilizando los ya existentes.

4.3.5 Interoperabilidad

La interoperabilidad se refiere a la habilidad de una aplicación para acceder a aplicaciones, los datos o los recursos en otras plataformas. Muchos ambientes empresariales soportan diferentes tipos de hardware y sistemas de software que deben trabajar juntos para que la empresa sea exitosa, por lo cual es importante que las aplicaciones de Windows DNA sean interoperables. Para maximizar la interoperabilidad de las aplicaciones, las aplicaciones de Windows DNA deben apoyarse en:

- Objetos de Datos ActiveX® de Microsoft (ADO) o accesos de datos universales con OLE DB.

- Extensible Markup Language (XML) para compartir datos con otras aplicaciones.

- DCOM para acceder aplicaciones en sistemas UNIX o en sistemas de almacenamiento en Virtual Múltiple (Multiple Virtual Storage, MVS).

- MSMQ para acceder a sistemas de cola de mensajes en otras plataformas.

- Y el integrador de otras acciones COM (COMTI) para ejecutar el Sistema de Control de Información de Clientes (Customer Information Control Systems, CICS), (LU 6.2) o el Sistema de Administración de Información de Transacciones (IMS) en sistemas MVS.

4.4 COM

Windows DNA está basada en un modelo de programación denominado COM (*Component Object Model*). El uso del modelo COM se ha difundido ampliamente desde su introducción por Microsoft y por ser parte integral de muchas aplicaciones y tecnologías, incluyendo en Internet Explorer y todo el suite de aplicaciones *Office.*

COM permite que los desarrolladores puedan crear complejas aplicaciones usando una serie de objetos de software para negocios. Así como los automóviles o las casas, son construidas con piezas estandarizadas, COM deja que los desarrolladores hagan porciones de sus aplicaciones usando componentes.

Esta estrategia acelera el proceso de desarrollo permitiendo que un grupo de muchos desarrolladores pueda trabajar en partes separadas al mismo tiempo. Los desarrolladores además pueden reutilizar componentes entre proyectos, y pueden fácilmente cambiar las versiones de estos sin necesidad de afectar a otras aplicaciones que los utilizan. COM además ofrece la ventaja de independencia del lenguaje de programación. Esto significa que los

desarrolladores pueden crear Componentes COM usando las herramientas y los lenguajes con los que estén más familiarizados, así como *Visual Basic .NET, Visual Basic, C++, Java, Borland Delphi,* etc.

La manera más fácil de visualizar las cosas es que COM sirve como pegamento entre las capas de la arquitectura, permitiendo que las aplicaciones Windows DNA se comuniquen en un ambiente altamente distribuido.

El Modelo de Objetos Componentes (COM) constituye un modelo de programación basado en objetos y diseñado para promover el trabajo conjunto. Permite que dos o más aplicaciones o componentes cooperen fácilmente, aunque hayan sido escritas por diferentes equipos en diferentes momentos y se ejecuten con diferentes S.O. Para soportar esta interoperabilidad, COM define e implementa los mecanismos que posibilitan la conexión de las aplicaciones como si de objetos se tratase.

4.4.1 Ventajas de COM

Las principales ventajas de COM son las siguientes:

Compatibilidad binaria

Cualquier lenguaje de programación que pueda crear estructuras con punteros y realizar, ya sea de forma explícita o implícita, llamadas a las funciones apuntadas por esos punteros, pueden crear y utilizar componentes COM. Estos componentes pueden implementarse con una gran variedad de lenguajes de programación y utilizarse posteriormente desde clientes escritos en otro lenguaje completamente diferente. COM asegura la compatibilidad binaria entre el cliente desarrollado en Visual Basic y los componentes desarrollados en otro lenguaje como Visual C++, Visual J++ o cualquier otra herramienta de terceros.

Compatibilidad en el trabajo con sistemas operativos distintos.

Los componentes COM deben desarrollarse para una plataforma específica, como Windows NT o UNIX ya que no pueden ejecutarse en una plataforma que no sea para la que fueron escritos; sin embargo, sí que puede comunicarse con otros objetos COM situados en otras plataformas. También es posible construir componentes COM para varias plataformas, incluyendo Windows 95, Windows NT, Windows XP, Macintosh y UNIX, etc.

Localización transparente

Permite usar un componente desde un cliente sin importar si ese componente pertenece al mismo proceso, a otro proceso en la misma máquina o a otro proceso en una máquina distinta.

Reutilización del código

Es la mayor ventaja de COM. Los componentes COM contienen clases que exponen grupos de métodos, conocidos como interfaces, a través de los cuales los clientes se comunican con objetos. Dado que estas interfaces están bien documentadas, el código que implementan puede reutilizarse con facilidad.

Control de versiones

Las interfaces implementadas en las clases son estáticas, una vez definidas, no varían. Por lo tanto, para añadir nuevas funcionalidades habrá que añadir nuevas interfaces a las clases. Se puede cambiar la implementación de las viejas interfaces siempre que se mantenga la compatibilidad con los clientes que ya las usaban. No se pierde funcionalidad cuando se actualiza o modifica un componente COM, siempre se mejora o se añaden nuevas.

4.4.2 Implementación de COM

El Modelo de Objetos Componentes (COM) es una especificación de cómo estos objetos interactúan con sus clientes. Como tal especificación, COM define una funcionalidad estándar, no define cómo debe implementarse sino qué es lo que debe tener. Entre lo que específica COM se incluye lo siguiente:

Cómo se instancian los objetos COM.
Cómo acceden los clientes a las características de estos objetos
La responsabilidad de su auto-destrucción cuando ya no quedan instancias suyas.
Además de lo que es meramente la especificación, existen bibliotecas COM que contienen:

Una reducida API para facilitar la creación de aplicaciones COM, tanto clientes como servidoras. Esta API ofrece, por un lado, las funciones para trabajar con clientes y, por otro, las que permiten a los servidores proporcionar objetos COM.

Servicios de localización, a través de los cuales se averigua en qué servidor se encuentra el objeto y la ubicación dentro de éste. Indirectamente esto incluye un soporte entre el identificador de objeto y el paquete en el que se encuentra su código, que normalmente se obtiene a través del Registro de Windows. De esta manera, los clientes quedan totalmente independientes de los posibles cambios de ubicación de los objetos COM, ya sean cambios de paquete o incluso cambios de máquina servidora.

Servicios para la transparencia en las llamadas a procedimientos remotos, es decir, cuando un objeto está ejecutándose en un proceso aparte o en una máquina distinta desde la que se usa.

5. DISEÑO DE APLICACIONES CLIENTE SERVIDOR DE N CAPAS

5.1 Tres capas utilizando reglas del negocio

Después de consultar con expertos y analizar bibliografía especializada en el tema, se ha encontrado que aún no se cuenta con una metodología dada para diseñar aplicaciones distribuidas en tres capas, por esto es que los expertos consultados sugirieron utilizar uno de los modelos o guías que utiliza Microsoft Co. para el diseño de este tipo de aplicaciones, este modelo se denomina Modelo de Diseño de Soluciones que es parte de una infraestructura más amplia de modelos denominada *Microsoft Solución Framework .*

Si bien no es una metodología que muestre paso a paso como llevar a cabo el diseño, como modelo presenta todas las pautas para poder diseñar una aplicación en tres capas, por otra parte el lenguaje gráfico que se utilizará combinado con este modelo es UML (*Unified Modeling Language*).

Gran parte del esfuerzo de este trabajo consiste idear la manera de combinar el modelo de soluciones (MDS) con el lenguaje de diagramas (UML). A continuación se hará una breve introducción del modelo propuesto y como fue combinado con UML en cada una de las etapas del diseño

5.2 Herramientas de desarrollo

5.2.1 MSF

5.2.1.1 *Microsoft solution framework* (MSF)

La manera más fácil de llegar al éxito es seguir los pasos exitosos de otros. Esto no significa que se debería esperar a ver que es lo que la competencia esta haciendo, y luego colocar esto en un producto duplicado o copiado. La idea es aprender a hacer las cosas de la misma manera que lo hacen los grandes siguiendo sus planes de acción. Esto es lo que hace MSF; toma las mejores prácticas de los expertos y las integra en distintos modelos, principios y guías.

MSF consiste en siete modelos, que pueden ser usados individualmente o combinados. Estos modelos son:

- Team Model (Modelo de Equipo)
- Process Model (Modelo de Procesos)
- Application Model (Modelo de Aplicación)
- Solution Design Model (Modelo de Diseño de Soluciones)

- Enterprise Architecture Model (Modelo de Arquitectura Empresarial)
- Infraestructure Model (Modelo de Infraestructura)
- Total Cost Ownership Model (Modelo de Costo Total de Propiedad)

5.2.1.2 El modelo de diseño de soluciones

El Modelo de Diseño de Soluciones ayuda al equipo del proyecto a anticiparse a las necesidades del usuario incluyéndolo en el problema. Vale destacar la diferencia entre cliente y usuario, cliente se considera a la persona que paga por el software y usuario es aquella persona que va a utilizar el software. Estos no son necesariamente la misma persona. Es importante conseguir los requerimientos de los usuarios si es que se quiere lograr que la solución este enfocada a la realidad del negocio.

En el Modelo de Diseño de Soluciones, los usuarios se ven involucrados en el proceso de diseño. Obteniendo de ellos información sobre ciertos detalles, como de funcionalidad y otros requerimientos, el equipo puede determinar como se va a usar la aplicación e incrementar su productividad.

Más allá de involucrar a los usuarios en el diseño, el Modelo de Diseño de Soluciones provee una estrategia para diseñar soluciones orientadas a negocios que deben ser creadas para satisfacer necesidades especificas. Este

modelo une el Modelo de Equipo, el Modelo de Aplicación y el Modelo de Procesos, de tal manera que los recursos pueden ser enfocados en las áreas donde tengan mayor rendimiento.

El Modelo de Diseño de Soluciones esta compuesto por diferentes perspectivas. Una perspectiva es una forma de ver algo, lo que en este caso es el proceso de diseño de la aplicación. Se utiliza para centrarse en el proceso mismo del diseño. Estas perspectivas son:

- Diseño conceptual
- Diseño lógico
- Diseño físico

Las perspectivas son usadas para identificar los requerimientos técnicos y de negocios para la aplicación. El resultado de utilizar este modelo es una mejor distribución de los recursos del proyecto, lo que puede facilitar mucho las cosas.

5.2.1.2.1 Diseño conceptual

Es donde se origina el concepto inicial de la solución. Es en este diseño donde el equipo de desarrollo trata de entender las necesidades de los usuarios de la solución. Escenarios y modelos son usados para suavizar este entendimiento de manera que cada una de las entidades involucradas (equipos

de desarrollo, clientes y usuarios) sepan que es lo que se necesita de la solución.

El proceso de Diseño Conceptual esta compuesto de las siguientes tareas para determinar y substanciar los requerimientos de la aplicación:

- Identificación de usuarios y sus roles
- Conseguir información de los usuarios
- Validación del diseño

5.2.1.2.1.1 Perfiles de usuario

Perfiles de usuario son documentos que describen con quien se esta lidiando, y proveen una descripción de la gente y los grupos que usan el sistema. Esta información es usada para organizar como la información será recolectada, e identificar quien dará dicha información para su recolección. Estos perfiles también pueden ser creados al tiempo que se generan los escenarios de uso.

5.2.1.2.1.2 Escenarios de uso

Los escenarios de uso describen los requerimientos del sistema en el contexto del usuario, mostrando como se efectúan los procesos de negocios, o como se deberían efectuar. Los escenarios de uso toman los datos que han sido recolectados, y los aplica en un documento donde paso a paso se describe que pasa primero, luego y después en la ejecución de una tarea específica. Esto transforma los requerimientos que se han recolectado en el contexto de cómo se usan los procesos, funciones y procedimientos.

Existen diferentes métodos para construir los escenarios de uso que son:

a. El modelo de proceso de flujo de trabajo

Es usado para crear escenarios de uso que muestran como trabajos específicos son ruteados a través de una organización.

Al usar este modelo es necesario definir pre y pos condiciones. Estas son las condiciones necesarias para que el trabajo sea ruteado de un área a otra, y que es necesario para que un paso particular pueda darse.

b. El modelo de secuencia de tareas

Es usado para crear escenarios de uso. Este modelo observa a las series de acciones o secuencias de tareas que un usuario efectúa para completar una actividad.

Es posible usar este modelo con texto estructurado o no estructurado. Dependiendo del que se use, se necesita identificar el rol del usuario, y escribir el escenario de uso para este. El rol del usuario debe estar identificado en el escenario de uso de manera que cualquiera que lo vea pueda saber quien efectúa que actividad.

c. El modelo de ambiente físico

Los escenarios de uso también son útiles para entender el ambiente físico en el que se desenvuelve la aplicación. Esto se debe a que el diseño puede ser afectado por el lugar donde la aplicación vaya a ser usada, además de cómo y por que.

Este modelo observa el ambiente en el que la aplicación va a ser usada. Al usar este modelo, se documenta como las actividades se relacionan con el ambiente físico de la empresa.

Esto permite determinar como los datos se mueven a determinadas localizaciones, como un proceso o una actividad de negocio se mueve de un departamento a otro, etc.

El paso final del Diseño Conceptual es validar el diseño. Esta es una presentación del entendimiento del equipo de los requerimientos del usuario. Se efectúa mostrando a usuarios finales y otras partes interesadas los escenarios de uso que se han creado, esto permite determinar si se tiene un entendimiento correcto de lo que se requiere de la aplicación.

Una vez que se ha llegado al final del proceso del Diseño Conceptual, se esta generalmente listo para aplicar los documentos obtenidos al diseño lógico. Si es que fuera necesario es posible volver al Diseño Conceptual para determinar necesidades y percepciones de otras características o funcionalidades del producto. Esto provee de gran flexibilidad al proceso de diseño de la solución de negocios

5.2.1.2.2 Diseño lógico

Este diseño toma la información brindada por el Diseño Conceptual y la aplica al conocimiento técnico. Mientras que los requerimientos y necesidades de los clientes y usuarios son identificados en la perspectiva de diseño previa, es en éste diseño que la estructura y comunicación de los elementos de la solución son establecidos. Los objetos y servicios, la interfaz de usuario y la

base de datos lógica son el conjunto de elementos identificados y diseñados en esta perspectiva.

En esta etapa no interesan los detalles de implementación física, tales como donde se van a alojar ciertos componentes o cuantos servidores están involucrados. El único interés es crear un modelo de abstracción de alto nivel, independiente de cualquier modelo físico.

Este alto nivel de abstracción permite distanciarse de muchos detalles recolectados en la fase conceptual y organizarlos sin tener que analizar los detalles particulares de cada uno de los requerimientos. Además hace posible centrarse en un requerimiento específico a la vez sin perder la visión de la aplicación como un todo.

El Diseño Lógico es el proceso de tomar los requerimientos de usuario obtenidos en el Diseño Conceptual y mapearlos a sus respectivos objetos de negocios y servicios.

5.2.1.2.2.1 Organización de las estructuras lógicas

Una vez que se han identificado los objetos, es necesario organizarlos según los servicios que proveen, y las relaciones que tienen unos con otros.

Existen muchas consideraciones que deben ser tomadas en cuenta al diseñar una aplicación en tres capas que proporciona ciertos beneficios como ser escalabilidad, disponibilidad y eficiencia. Cuando se diseñan los objetos se debe dejar que estos factores dirijan la manera de organizar las estructuras lógicas. A pesar de que estos conceptos también se aplican al Diseño Físico, tienen igualmente importancia en esta etapa de Diseño Lógico. Es bueno definir que tan granular, tienen que ser los componentes. Si estos mantendrán un estado o no con el fin de maximizar la escalabilidad, estos y otros elementos deben analizarse detenidamente para obtener un buen Diseño Lógico.

5.2.1.2.2.2 Del diseño conceptual al diseño lógico

Crear un Diseño Lógico consiste en mapear a objetos las reglas de negocios y los requerimientos de usuario identificados en el Diseño Conceptual. Estos objetos pueden ser más fácilmente identificados de los requerimientos de usuario por los nombres o sustantivos, los servicios que proveen estos objetos representan las reglas y requerimientos del dominio del negocio que se está modelando y son reconocidos por verbos, para reconocer las propiedades o atributos de un objeto se deben identificar los datos asociados al objeto.

Cuando se diseñan los objetos es importante que estos se centren en una sola cosa en lo posible, en otras palabras los objetos deberían solamente proveer servicios relacionados con un único propósito.

La funcionalidad de un objeto se llama granularidad. Mientras su granularidad es mas fuerte el objetos presta muchos servicios, mientras más débil es su granularidad menos servicios presta. Lo óptimo es que el objeto tenga granularidad débil.

5.2.1.2.3 Diseño físico

Es donde los requerimientos del diseño conceptual y lógico son puestos en una forma tangible. Es en este diseño que las restricciones de la tecnología son aplicadas al Diseño Lógico de la solución. El Diseño Físico define cómo los componentes de la solución, así como la interfaz de usuario y la base de datos física trabajan juntos. Desempeño, implementación, ancho de banda, escalabilidad, adaptabilidad y mantenibilidad son todos resueltos e implementados a través del Diseño Físico. Ya que esta perspectiva transforma los diseños previos en una forma concreta, es posible estimar qué recursos, costos o programación de tiempo serán necesarios para concretar el proyecto.

Al lidiar con estas tres perspectivas, es importante notar que éstas no son series de pasos con puntos de finalización claros. No es necesario alcanzar un punto específico en una de las perspectivas antes de continuar con la siguiente. De hecho, un área de diseño puede ser usada en combinación con otra de manera tal que mientras una parte de la solución es diseñada conceptual o lógicamente, otra esta siendo codificada o implementada en el producto final. Desde que no existen etapas con puntos definidos o límites, es posible regresar a las distintas perspectivas de diseño cuantas veces sea necesario. Esto permite afinar el diseño revisando y rediseñando la solución

5.2.2 UML

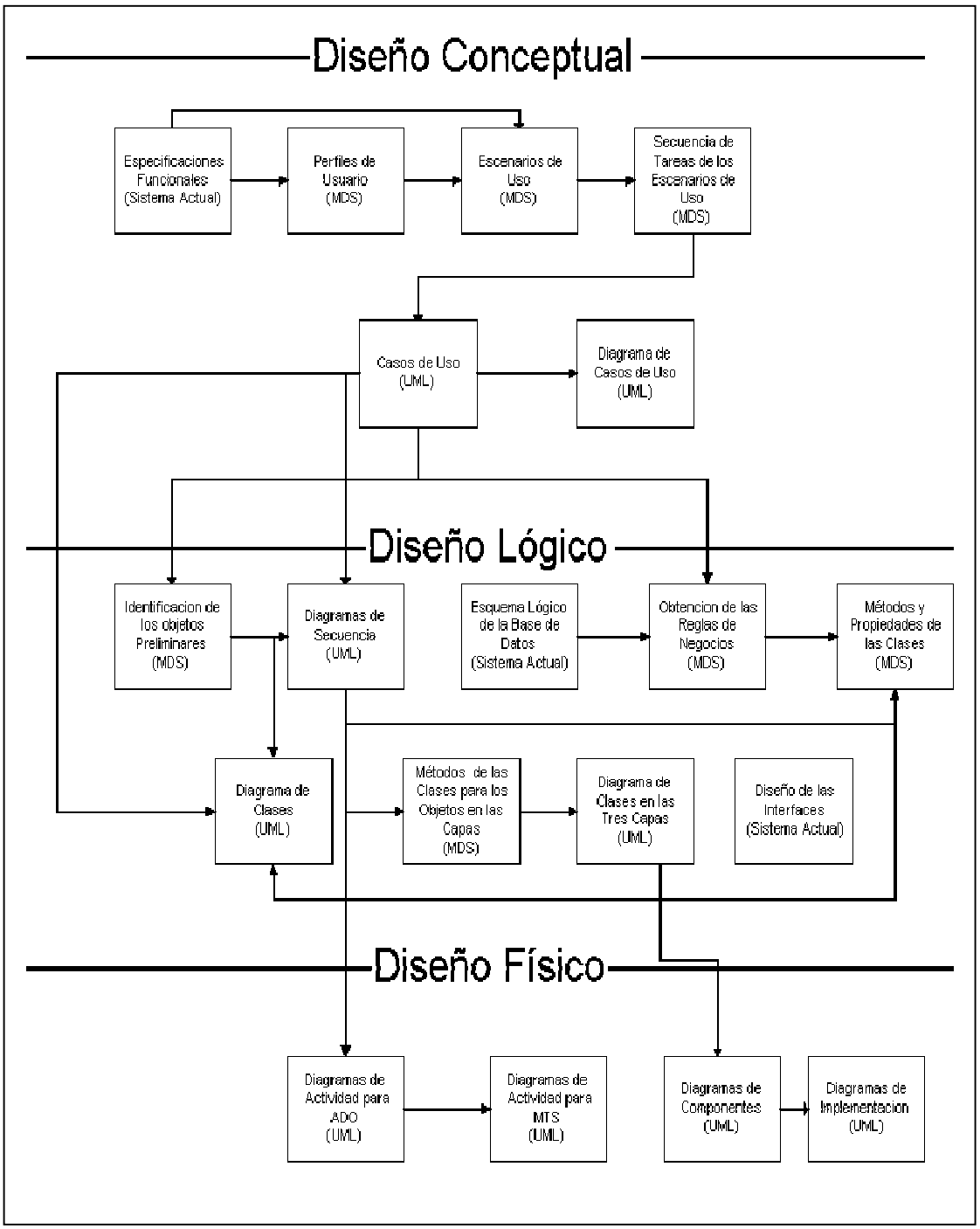
5.2.2.1 Modelo de diseño de soluciones con diagramas UML

Con el fin de llevar al cabo el diseño de la migración de la Aplicación X desde una infraestructura de dos capas a una infraestructura de tres capas se conjugó el Modelo de Diseño de Soluciones. Se utilizaron diagramas UML en cada una de las etapas del modelado para poder diseñar la migración en sí, se hicieron algunas adaptaciones al modelo ya que se debió tomar en cuenta ciertas características que varían desde el diseño de una migración al diseño de una aplicación desde cero.

A continuación se expondrán dichas adaptaciones en cada una de las fases del modelo y los diagramas que se utilizaron en cada una de estas fases.

En el siguiente Esquema se muestran las herramientas utilizadas en cada una de las perspectivas del diseño.

Figura 11. Modelo de diseño de soluciones



5.2.2.1.1 Diseño conceptual

5.2.2.1.1.1 Especificaciones funcionales

Durante el Diseño Conceptual como se dijo anteriormente se modelan los requerimientos del usuario, para adaptar el modelo a la migración en vez de usar los requerimientos de los usuarios, lo que se hizo fue tomar las especificaciones funcionales del sistema actual como base para el diseño.

5.2.2.1.1.2 Perfiles de usuario

Para identificar a los actores del sistema como son llamados en UML o los perfiles de usuario como son llamados en Diseño de Soluciones Microsoft, se hizo un análisis de las personas que utilizan en la actualidad el sistema y los roles o papeles que juegan en su interacción con el mismo así se llegó a la conclusión de que los principales actores eran el Administrador, el Cajero y el Vendedor que durante el desarrollo del diseño serán simplemente denominados Responsables.

Otro de los actores encontrados fue nombrado como otros módulos en representación de los otros módulos del sistema que en el caso de este diseño actúan como agentes externos al ambiente del módulo de ventas, pero que son necesarios como un actor por las interrelaciones que las ventas tienen con ellos.

5.2.2.1.1.3 Escenarios de uso y secuencia de tareas de los escenarios de uso

Los Escenarios de Uso describen los requerimientos del sistema en el contexto de las especificaciones funcionales mostrando como se efectúan los procesos de negocios y que actores o perfiles de usuario intervienen en estos a través de las secuencia de tareas descritas para cada uno de los Escenarios.

a. Casos de uso

Basándose en la Secuencia de Tareas de los Escenarios de Uso se crearon los Casos de Uso de manera que se pueda tener una idea clara de que es lo que se quiere funcionalmente del sistema y de la forma en la que se realizan los procesos.

b. Diagramas de casos de uso.

Para finalizar y poder hacer la validación del Modelo Conceptual se creó el Diagrama de Casos de Uso donde se conjugan todos los Casos de Uso en un único diagrama que se analizó para ver si cumplía con las especificaciones funcionales del sistema actual

5.2.2.1.2 Diseño lógico

5.2.2.1.2.1 Identificación de los objetos preliminares

En esta etapa de diseño se identificaron los objetos y sus métodos basándose en los Casos de Uso de la etapa anterior.

5.2.2.1.2.2 Diagramas de secuencia

Para poder representar las interacciones de objetos con otros objetos, encontrados en la etapa previa, y de objetos con los actores del sistema se utilizaron Diagramas de Secuencia, el motivo para haber escogido este tipo de diagrama es que es específico para diseñar interacciones y se deriva directamente de los Casos de Uso, otros diagramas que modelan interacciones son los Diagramas de Colaboración, la diferencia es que dichos diagramas se construyen basándose en una técnica completamente distinta de modelado. Ya que en el caso de la migración, las especificaciones funcionales del sistema actual vienen a ser los requerimientos de usuario; los Diagramas de Secuencia son los que más se adecuan al modelo que se está usando.

Los Diagramas de Secuencia utilizados ayudan a convertir los Casos de Uso que se exponen de manera verbal en imágenes que mapean todos los mensajes. Como se sabe una imagen siempre es más fácil de comprender.

Los Diagramas de Secuencia además muestran el orden de los eventos que un usuario maneja mientras interactúa con el sistema.

5.2.2.1.2.3 Esquema lógico de la base de datos

El esquema lógico de la Base de Datos se obtiene del sistema actual, como ya se dijo anteriormente para fines de este diseño, la base de datos no necesita modificaciones, sin embargo si es necesario conocerla para poder definir así las características de los objetos que van a lidiar con las tablas, relaciones y atributos de dicha base de datos, este es el motivo para presentar en esta etapa del diseño su esquema.

5.2.2.1.2.4 Reglas de negocios

En la primera parte de este Diseño se pudieron encontrar algunos objetos y sus métodos, sin embargo en esa etapa previa no es posible conocer las propiedades de éstos ni sus métodos con exactitud, para este efecto se utilizaron la Reglas de Negocio.

Para obtener las Reglas de Negocio de Definición se utilizó el esquema de la Base de Datos, ya que se asume que los objetos que van a manejar o manipular las tablas deben tener las mismas propiedades o atributos de éstas.

Las Reglas de Negocio de Restricción se obtuvieron de los Casos de Uso donde se describe claramente las condiciones asociadas a cada uno de los elementos.

5.2.2.1.2.5 Métodos y propiedades de las clases

Para encontrar los métodos y las propiedades de las clases se utilizaron tanto los diagramas de Secuencia como las Reglas de Negocio. De las Reglas de Negocio de Definición se obtuvieron las propiedades de cada uno de los objetos, y de las Reglas de Negocios de Restricción se obtuvieron los métodos. De los mensajes de los Diagramas de Secuencias se obtuvieron los otros métodos de las clases.

5.2.2.1.2.6 Diagramas de clases

Para poder analizar la relación entre Clases (Conjunto de Objetos) se utilizó el Diagrama de Clases, dicho diagrama se basa en los anteriormente mencionados Diagramas de Secuencia.

5.2.2.1.2.7 Métodos de las clases en cada una de las tres capas

Para obtener un buen diseño de una aplicación de tres capas es necesario poder colocar los métodos de las clases correctas en las capas correctas, de

esta manera es posible encontrar las características de escalabilidad y buen desempeño.

En las etapas anteriores se llegó a encontrar los métodos de las clases, pero en el caso de una aplicación de tres capas es necesario que los métodos se ubiquen según su funcionalidad en cada una de las capas

En la capa de servicios de presentación o interfaz, deben ir los métodos que no tienen nada que ver ni con manejo de datos, ni con reglas de negocios, simplemente llamadas a métodos de otras capas y procesos de exposición de información a través de la interfaz. Como por ejemplo Ir al Siguiente Registro, etc.

En la capa del medio o de servicios de negocios, deben ir todos los métodos que contienen las reglas de negocios propiamente dichas y las llamadas a los métodos de manipulación de datos de la capa tres.

En la capa tres o capa de servicio de datos deben ir todos los métodos de manipulación de datos que acceden al repositorio de datos directamente en el caso de este diseño la Base de Datos.

Para encontrar el lugar donde colocar cada uno de los métodos se debe utilizar el Diagrama de Secuencias y un análisis de la funcionalidad de cada uno de los métodos.

5.2.2.1.2.8 Diagrama de clases en las tres capas

El Diagrama de Clases en las Tres Capas muestra de manera gráfica los métodos que se encontraron en la etapa previa, en este diagrama no es necesario colocar las relaciones entre los objetos, ya que las relaciones son las mismas que las del anterior Diagrama de Clases.

5.2.2.1.2.8 Diseño de las interfaces de usuario

El diseño de las Interfaces de usuario se obtiene de la aplicación actual, no amerita cambios ya que los usuarios del sistema se han acostumbrado ya a su interfaz y tiene todas las características necesarias para adaptarse a la funcionalidad del sistema

5.2.2.1.3 Diseño físico

Hasta esta etapa del diseño lo que se ha hecho es modelar los requerimientos de la aplicación por medio de sus especificaciones funcionales hasta encontrar distintos aspectos de cómo trabaja la aplicación por fuera. Ahora es necesario considerar como la aplicación trabaja por dentro, o más precisamente como las tecnologías y arquitecturas que se han elegido van a impactar en el diseño.

5.2.2.1.3.1 Revisión de los requerimientos tecnológicos

Se debe hacer un análisis o revisión de las tecnologías que se escogieron para utilizar dentro de la aplicación, de esta manera es posible ya comenzar a diseñar los elementos tomando en cuenta estas restricciones utilizando diagramas UML.

5.2.2.1.3.2 *Distributed Internet Applications* (DNA)

En esta parte daremos una breve explicación de porque se escogió como requerimiento tecnológico que la aplicación se base en el modelo Windows DNA.

DNA es una arquitectura prácticamente nueva, no es realmente nada más que una arquitectura de tres capas, que algunas veces es llamada n capas, excepto porque hace énfasis en el tema de Internet.

Ya se explico anteriormente que una de las capas se coloca en el cliente, que en el caso de la aplicación X es la interfaz que manejaran los empleados encargados de realizar las ventas. La segunda capa se encuentra en el servidor que contiene la lógica de negocios, que es la que se encarga de lidiar con las transacciones (editar, actualizar, adicionar, recuperar registros) de la base de datos. La tercera capa en la que contiene la base de datos.

El esquema se vería de la siguiente manera.

A primera vista, no parece haberse ganado demasiado. Se ha removido la lógica de negocios que estaba en los procedimientos junto con la interfaz, y se la ha colocado dentro de componentes en la capa del medio. De todas maneras si se analiza con mayor profundidad se pueden encontrar muchos beneficios importantes de esta arquitectura.

a. La capa del medio

A diferencia de un procedimiento almacenado SQL, un componente Visual Basic en la capa del medio maneja las transacciones de la lógica de negocios. Tiene código que es fácil de leer, escribir y depurar, además de tener pocas limitaciones. Se puede además crear un componente que pueda trabajar con una multitud de bases de datos en diferentes servidores. Se pueden construir componentes muy poderosos que pueden realizar manipulaciones muy complejas de datos. Las actividades extremadamente complejas, que son realizadas por el cliente pueden ser ejecutadas en la capa del medio. Esto hace al cliente más pequeño, liviano y mejor para trabajar en Internet.

La capa del medio trabaja como un intermediario, tomando los requerimientos del cliente para un conjunto particular de información formateado de acuerdo a un conjunto de reglas de negocios. La capa del medio consigue los datos, los transforma de acuerdo a las reglas de negocios y los devuelve al cliente. El cliente no tiene porque preocuparse acerca de las

conexiones a la base de datos, tablas o transformación de los datos. El cliente simplemente solicita la información que necesita y la capa del medio se la devuelve.

b. Trabajar con componentes

Combinado la arquitectura DNA con Programación Orientada a Objetos, se pueden crear proyectos muy poderosos. Si se crean componentes que realizan ciertas tareas, se pueden luego construir aplicaciones basadas en dichos componentes. Usar estos componentes para construir nuestra aplicación X de tres capas permite colocar los componentes en las diferentes capas, y en distintas ubicaciones dentro de las capas, dependiendo de las necesidades particulares de la aplicación.

Usar DNA y OOP (*Object Oriented Programming*), ofrecen otra ventaja. Si por alguna razón la lógica de negocios cambia, seguramente uno o dos componentes tendrán que ser cambiados. Estos componentes deberían estar en la capa del medio, lo que significa que solamente habrá que actualizar el servidor. En el caso de una empresa que utilice la aplicación que cuente con cientos de usuarios es mejor actualizar uno o dos servidores que cientos de clientes.

5.2.2.1.3.3 *Activex Data Objects* (ADO)

La tecnología de acceso a los datos escogida para la aplicación X es ADO, el motivo principal para haberla escogido es que al ser un conjunto de *controles Activex*, es neutral al lenguaje , esto quiere decir que puede ser usado desde *Visual Basic, Delphi, Visual C++, Java , Java Script ó VBScript*.

a. Diagramas de actividad ADO y MTS

Las restricciones tecnológicas más importantes en el caso del diseño de esta aplicación son el modelo de acceso a datos ADO (*Activex Data Object*) y MTS (*Microsoft Transaction Server*) para hacer el diseño físico se utilizaron Diagramas de Actividad para mostrar como manejar las conexiones a los datos, además de cómo manejar el contexto de los objetos dentro de MTS.

5.2.2.1.3.4 Diagrama de componentes

Para continuar con el diseño físico, ahora que ya se han diseñado las clases y se les ha aplicado las restricciones tecnológicas, se deben seguir los siguientes pasos:

- Agrupar las clases en componentes

- Agrupar los componentes en paquetes y procesos MTS

- Asignar los paquetes y procesos a las distintas máquinas con las que se va a trabajar.

a. Agrupar las clases en componentes

La gran mayoría de las clases usadas en MTS, son fáciles de agrupar en componentes. Cada clase COM es implementada en un componente in process. Si se tienen una clase que solamente puede ser instancia da vía otra clase, probablemente se necesite crear un componente con ambas clases.

b. Agrupar los componentes en paquetes y procesos

MTS usa los paquetes como unidades de confianza y unidades de implementación. Un paquete es simplemente un conjunto de componentes que desempeñan funciones que tienen algo en común. Un componente puede ser instalado en solamente un paquete en una máquina. Existen dos tipos de paquetes: *paquetes de librería* y *paquetes de servidor.* Un paquete de librería corre en el proceso del cliente que lo creó. Un paquete de servidor corre en un proceso separado.

c. Asignar paquetes y procesos a las máquinas

La mayoría de las decisiones a cerca de donde los paquetes y procesos deberían correr se deben postergar hasta el final. Si es que se identifica algún requerimiento específico de implementación que se aplique a ciertos componentes debería ser documentado. En el caso de este proyecto no existe ningún requerimiento especial, de manera que todos los componentes pueden correr en una sola máquina.

d. Diagramas utilizados en el diseño físico

En el diseño Físico se utilizan diagramas de componentes para describir como las clases se agrupan en componentes y paquetes. Se utilizan diagramas de implementación para describir como los paquetes y procesos son distribuidos entre las máquinas del sistema.

5.2.2.1.3.5 Diagrama de implementación

5.2.2.1.3.5.1 Diseño de los paquetes

Para poder diseñar los paquetes se deben tomar en consideración los siguientes puntos:

- Activación
- Comparición de recursos
- Aislamiento de fallas
- Seguridad

a. Activación

Se pueden activar los componentes de la aplicación de dos maneras: en el proceso del cliente o en un proceso de manejo específico de paquetes administrado por MTS. Los paquetes de librería no soportan seguridad declarativa, ni ofrecen los beneficios de los procesos de aislamiento. Estos paquetes son usados más comúnmente para componentes utilitarios que van a ser utilizados por múltiples aplicaciones. Son también útiles cuando no se desea la sobrecarga de un proceso separado y no se requiere chequeo de autorización. Los paquetes de servidor se usan para correr componentes en procesos separados administrados por MTS. Estos soportan seguridad declarativa, pooling de recursos además de otras características.

Decidir cuantos paquetes son necesarios en una aplicación es un acto de balanceo. Si se tienen muchos paquetes el administrador del sistema tendrá gran flexibilidad durante la implementación de la aplicación. Sin embargo, cada proceso de servidor requiere cierta cantidad de trabajo para administrar el

pooling de recursos, las propiedades compartidas, etc. Adicionalmente las llamadas ínter proceso COM son mucho más caras que las llamadas in-process (dentro del proceso).

b. Compartición de recursos

Los componentes que usan los mismos recursos, así como bases de datos, deberían ser agrupados en un solo paquete servidor. Recuérdese que MTS administra el pooling de recursos basándose en los procesos. Cada paquete de servidor consigue su propio pool de hilos de proceso, pool de conexiones a la base de datos. Si dos componentes utilizan la misma base de datos y son colocados en paquetes de servidor separados no pueden hacer pooling de conexiones a la base de datos. Si los componentes están localizados en el mismo paquete servidor, si pueden utilizar el pool de conexiones. Esta capacidad puede mejorar enormemente el desempeño y la escalabilidad de la aplicación de la misma manera los componentes que comparten propiedades deben localizarse en el mismo paquete para asegurar una operación correcta de la aplicación.

También se debería considerar la localización de los recursos cuando se están diseñando los paquetes. En general, los componentes deberían estar tan cerca como sea posible de los recursos que utilizan, particularmente repositorios de datos, para ayudar a reducir el tráfico de red dentro de la aplicación.

c. Aislamiento de fallas

Se debe considerar separar los componentes en diferentes paquetes para asegurar que una falla en uno de los componentes no cause que otros componentes fallen también ya que MTS termina un proceso si es que detecta corrupción interna. Las excepciones dentro de un componente pueden también forzar a un proceso servidor a terminar y cualquier estado de objeto que se estuviera manteniendo se perderá. Si la aplicación tiene componentes que mantienen su estado, se debe considerar colocar dichos componentes en paquetes de servidor separados.

d. Seguridad

Como ya se dio, los paquetes de servidor son las unidades de confianza de MTS. Las llamadas dentro de un paquete deben ser seguras. Por esto es que los requerimientos de seguridad de la aplicación tienen un gran impacto en el diseño de los paquetes. Si las llamadas dentro de un componente deben ser autorizadas, los clientes y el componente deben estar localizados en diferentes paquetes. Solo los componentes que pueden de manera segura llamar a otros componentes sin requerir autorización deberían estar localizados en el mismo paquete.

6. CUANDO UTILIZAR APLICACIONES EN 3 CAPAS

- En organizaciones que tengan usuarios distribuidos en zonas geográficas lejanas

- En organizaciones que desarrollen varias aplicaciones y deseen reutilizar el código

- En organizaciones que utilicen herramientas de desarrollo de distintos proveedores

- En organizaciones que desean utilizar el Internet como medio de hacer negocios.

Las aplicaciones desarrolladas con arquitectura n capas enfocadas en reglas del negocio proporcionan una gran variedad de ventajas, por lo tanto se recomienda implementar o desarrollar este tipo de aplicaciones a empresas que cumplan con los requerimientos descritos en el párrafo anterior, auque realmente proporcionan la gran gama de ventajas para cualquier tipo de aplicación.

6.1. Beneficios operativos

Continuando con la línea de pensamiento esbozada en los párrafos anteriores cabría esperar un ambiente operativo, donde se registrara un incremento paulatino en las demandas computacionales de los usuarios.

En este punto es donde comienza a cobrar legítima importancia una arquitectura distribuida donde:

La información se genera en los entornos más seguros y eficientes

Al deshacerse de todo soporte físico la información recupera el carácter abstracto que posee por definición. Sólo se considera la forma más eficiente de generarla y segura de almacenarla. Como consecuencia de este enfoque se experimenta una gradual desintermediación en la generación de información.

La información es accesible donde sea necesaria

La información abstracta es el primer paso, el siguiente es el acceso instantáneo a ésta. No más restricciones que las propias del ambiente de negocios.

La información se acerca a quien la necesita y no viceversa

Pierde sentido la idea de cliente autónomo autosuficiente y cobra relieve el concepto de PC invisible.

La ubicuidad de la información se debe corresponder con la ubicuidad del acceso físico a la misma.

Consecuencias: eliminar la instalación y las actualizaciones de software en pos de una noción de Automatic Desktop (Escritorio Automático) y una modalidad de acceso uniforme (la Navegación).

La idea se termina de confirmar de la mano de la portabilidad total del cliente: transición al concepto de Client in Place

El dispositivo de acceso a la red es el navegador.

Estos conceptos son los fundamentos para una infraestructura que soporte el curso de la informatización corporativa, concepto de empresa virtual.

Beneficios administrativos

Una arquitectura integradora ofrece beneficios significativos en lo que se refiere a administración.

El desacoplamiento experimentado entre recursos físicos y aplicaciones brinda una variedad de posibilidades:

Inserción de estrategias para asignar y redistribuir recursos computacionales sobre requerimientos técnicos / funcionales

Estos requerimientos reflejan directamente procesos del negocio y por lo tanto asimilan en forma directa sus prioridades. Estas estrategias conforman un nuevo nivel de administración directamente ligado a procesos del negocio, denominadas estrategias de macro administración.

Cero administración, en clientes

Se libera a los administradores de las arduas tareas de instalación y configuración en los clientes y se posibilita la integración de clientes heterogéneos en forma automática.

Supervisión centralizada

Punto uniforme de demarcación y monitoreo de actividades. Auditoria con grados de intensidad y focalización.

Independencia del Hardware y Sistema Operativo en la capa de aplicación y encapsulamiento del *back-end*

Posibilita una mayor libertad para negociar decisiones de hardware y sistema operativo.

El monitoreo centralizado permitirá identificar los puntos críticos de la actividad soportando un concepto de escalabilidad On Demand.

Integración del concepto de *Fault Tolerance* a nivel aplicación en la arquitectura

Racionalización en la distribución de equipamiento

La arquitectura puede utilizar ambientes de alto rendimiento y con altos niveles de seguridad según la demanda de los procesos, posibilitando concentrar la inversión en equipamiento y software de base como por ejemplo, servidores de datos muy potentes.

Los requerimientos de alta performance se concentran y son regulados por la arquitectura propuesta con una garantía de óptimo uso.

En consecuencia se observa una recentralización del rol de servidor que pasa de un uso departamental a ser utilizado por todas las aplicaciones de la empresa.

Los recursos de hardware de características convencionales y de bajo costo, también se consideran parte del poder de cómputo instalado.

Beneficios al desarrollo

Como consecuencia de la gradual expansión de los alcances del sistema hasta integrarse completamente a los propios del dominio de negocios, cabe esperar un creciente aumento en la complejidad lógica de los subsistemas y a la vez, altos requisitos de flexibilidad para que no queden obsoletos antes de establecerse en producción.

La arquitectura complementa y da soporte a estos nuevos requerimientos:

Soporte a desarrollos *Top-Down* o analíticos

Conscientes de los cambios que frecuentemente experimentan los requerimientos en el marco de un proyecto, el enfoque ideal es el de desarrollo top-down o analítico. Los problemas objetivo se descomponen en subobjetivos.

De esta forma, las decisiones de diseño se corresponden directamente con motivaciones provenientes del análisis.

Informática centrada en la red (*Network Focused Computing*)

La red Internet como estándar de facto ofrece alcance ilimitado y compatibilidad bajo la capa de red, argumentos más que suficientes para no poder eludir considerarla. Si bien el surgimiento de las intranets corporativas, explotan este concepto, lo utilizan sólo en forma parcial, distribuyendo únicamente la capa cliente.

La información aún sigue siendo generada en sistemas semi-cerrados. Web Computing lleva la descomposición hasta el sistema mismo.

La infraestructura de comunicación con y dentro del sistema es la infraestructura proporcionada por la Red.

Los nuevos emprendimientos con ciertas ambiciones de longevidad deberán adaptarse.

La arquitectura soporta esta integración y provee la necesaria capa de seguridad (Internet Ready).

Solución abierta, interoperabilidad

Una solución abierta tiende no sólo a integrar el capital tecnológico en forma de hardware diseminado por la empresa, sino también los sistemas vigentes.

La arquitectura debe soportar la incorporación de subsistemas preexistentes Legacy Systems, como primer paso a una paulatina migración on demand de soluciones propietarias a soluciones estándar, previendo una gradual ingeniería reversa.

Soporte a requerimientos de globalización y regionalización

En una mayoría de casos las empresas expanden sus contextos de actividad más allá de una única región y deben convivir con normativas particulares disímiles. La integración operativa de un sistema implica manejar estos requerimientos de globalización y regionalización en forma transparente y consistente.

Garantía de portabilidad

La lógica independiente de la plataforma operativa proporciona:

- Sistemas con ciclos de vida independientes de las plataformas donde corren.

- Lógica distribuida en una red heterogénea como las actuales.

- Escalabilidad sustentada por equipamiento y transparente a la aplicación.

Personalización y especialización a través del *scripting*

La customización se realiza a través de un lenguaje sencillo donde la sintaxis y lo semántico se encuentran simplificados al máximo para expresar la lógica de una aplicación de propósito concreto.

CONCLUSIONES

1. El problema enfrentado, es un caso muy común, debido a que la tendencia más fácil y directa de encarar una aplicación cliente / servidor con las herramientas hasta hace poco disponibles es la de dos capas. Por esto el

enfoque con el que se ataco el problema es aplicable a varias situaciones que tiene características similares.

2. Para poder realizar un enfoque similar en otros casos, se tiene que prever que la aplicación original este estructurada en forma modular y con una adecuada división funcional, que como en el caso del presente trabajo, en el cual se facilito grandemente el diseño de la solución, teniendo que concentrarlo principalmente en los objetos de la capa de negocios o reglas del negocio

3. Si una aplicación no tiene adecuada modularidad y división funcional a la que se hace referencia en el párrafo anterior, el trabajo puede llegar a requerir hasta una reingeniería total de la aplicación, incluyendo un diseño desde cero y por supuesto la nueva construcción del sistema.

4. El diseño obtenido como solución al problema esta listo para ser aplicado y pasar a la fase de construcción. En este diseño se ha considerado con especial énfasis, la reutilización del código y los elementos ya existentes, de manera que la inversión realizada no se pierda totalmente y la madurez ganada por el producto durante los años de operación se mantenga.

5. DNA presenta una arquitectura de aplicaciones de tres capas, basadas en componentes, cuenta con servicios específicos en cada capa que se comunican entre si, mediante COM (*Component Object Model*). El hecho de usar componentes COM y poder reutilizarlos en distintas partes de la

aplicación o en otras aplicaciones permite facilitar la interoperación de los sistemas. Gracias al uso de estándares comunes los componentes son independientes de la plataforma y pueden acoplarse con componentes construidos en otros lenguajes y sobre otras plataformas, característica que se extiende a las aplicaciones construidas en base a componentes.

6. A cerca del modelo utilizado modelo de diseño de soluciones se puede concluir que las perspectivas de diseño que propone como son el diseño conceptual, el diseño lógico y el diseño físico, ayudan al analista a que durante el proceso de diseño tanto el usuario como los desarrolladores puedan entenderse y el producto final satisfaga las necesidades planteadas sin haber ocasionado un costo muy alto en el proceso de desarrollo e implementación.

7. La notación UML utilizada contiene suficientes herramientas gráficas que encajan perfectamente en las perspectivas del modelo utilizado, de manera que es un lenguaje gráfico recomendable para ilustrar el diseño de aplicaciones en tres capas.

RECOMENDACIONES

1. Antes de desarrollar sistemas de información, utilizando la metodología cliente servidor de n capas se debe de tomar en cuanta, las ventajas y

desventajas que esta arquitectura ofrece y si cumple con las normas y requerimientos del sistema a desarrollar.

2. Al definir las reglas del negocio, es necesario profundizar más el tema, ya que en este trabajo solo se tomaron conceptos generales y básicos, no se entro en mayor detalle, ya que el tema es bien amplio y complejo. Se necesita un grupo de personas expertas en el área del negocio, para poder definir y desarrollar las reglas del negocio dentro de un sistema.

3. Para usar esta metodología es necesario involucrar a todas las personas que tengan algún tipo de relación con el sistema, desde los usuarios finales hasta la alta gerencia.

4. Después de la minería de reglas se debe depurar, clasificar, y determinar la ubicación de la misma, esto depende del tipo de regla. La ubicación de la regla puede definirse en cualquier capa de la arquitectura, se debe de ubicar en la capa donde tenga mayor funcionalidad.

5. Para el desarrollo e implementación de sistemas utilizando reglas del negocio en arquitectura n capas, es necesario tomar en cuenta, que esta metodología es flexible a cualquier lenguaje de programación.

BIBLIOGRAFIA:

1. Agulló, Soliveres. “**Revista profesional para programadores (RPP)”**. s.l. s.e. s.a.

2. Alain, Gougeon. **Seminario: Reglas del negocio.** Proyecto SIAF-SAG / Datum. Hotel Marriot Guatemala City. Agosto, 2001.

3. Almagesto. http://www.almagesto.com/Ubicaciones/Eidos/Cursos/DNA/, Octubre, 2001.

4. BR Approach. **El enfoque RN**. s.l. s.e. s.a.

5. Campos, Miguel. **Seminario: Arquitectura n capas.** Microsoft de Guatemala. Diciembre, 2000.

6. Crear sistemas. http://www.crear.com.ar/, Octubre, 2001.

7. Data Management Group. http://www.dmreview.com/, agosto, 2001.

8. Departamento de lenguajes y sistemas informáticos. http://www.lsi.us.es/, Octubre, 2001.

9. Hay David, Keri Anderson Healy. **Defining Business Rules, ¿What Are They Really?** s.l. s.e. s.a.