ULACIT

(Universidad Latinoamericana de Ciencia y Tecnología)

Facultad: Ingeniería

Escuela: Informática

Tipo de Trabajo: PROYECTO INFORMÁTICO

Arquitectura para el desarrollo de aplicaciones distribuidas.

Para optar por el grado de: Licenciatura en Ingeniería Informática con Énfasis en Gestión de Recursos Tecnológicos

Alumno:
Jorge Eduardo Murillo López
Cédula 1 0948 0142

Profesor: Miguel Pérez Montero

III Cuatrimestre
Septiembre, 2006

Dedicatoria

Quiero dedicar esta tesis a una persona muy especial, que en este momento se encuentra lejos de casa, esforzándose y luchando por forjar su futuro y descubrirse tanto internamente como espiritualmente, Susana, “Su” como siempre la llamo, quiero que sepa que esta tesis es en honor a todo el sacrificio que usted ha realizado en Francia, como un símbolo de lucha.

Agradecimiento

Primero que todo quiero dar gracias a Dios, que después de tantos intentos para realizar la tesis, ahora puedo concluirla. También a mis papás que siempre me han apoyado en todas las cosas que me he propuesto en la vida, a las personas en el trabajo que me respaldaron en este proyecto, a Margarita y todos aquellos que siempre me insistieron en que me esforzara un poco más para que terminara la tesis, muchas gracias.

Índice

Introducción

~~Muchas~~ Las empresas en su mayoría, cuyo negocio no es necesariamente el desarrollo de software, cuentan con departamentos de Tecnología de Información (TI) o Sistemas de Información (SI), y al alcanzar cierto grado de madurez, empiezan a resentir la falta de políticas o lineamientos claros sobre el~~a~~ ~~línea~~modelo que se debe seguir en el desarrollo de aplicaciones de información, en cuanto a cómo se deben estructurar y desarrollar las nuevas aplicaciones.

~~Por esta razón~~Como consecuencia de lo anterior, ~~muchas veces se tiene que~~ los sistemas desarrollados por una empresa o departamento, no se parecen en nada, a los demás sistemas existentes dentro de ella, ~~lo cual~~esto lleva a que las labores de mantenimiento de las aplicaciones, resulten muy costosas para las empresas.

~~Como se mencionó anteriormente~~Por esta razón, el departamento de TI de una empresa, es el encargado del desarrollo de las aplicaciones, por lo tanto, ellos serán los grandes beneficiados con la implementación de una arquitectura para el desarrollo de las aplicaciones, ya que desde el punto de vista administrativo, las diferentes jerarquías del departamento, podrán tener más control del desarrollo, y en especial, del mantenimiento de las aplicaciones.

Este beneficio se extiende por todos los niveles del departamento, hasta los programadores, ya que ~~permite~~ propicia la especialización del personal en distintas áreas técnicas, fundamentales para el desarrollo eficiente del software, ~~lo que~~pues asegura un mayor conocimiento de los componentes que conforman no solo una, sino el conjunto de aplicaciones de una organización, ~~permitiendo~~ lo que permite ~~esto,~~ una respuesta más rápida y acertada en el momento que surja algún tipo de problema programático.

De esta forma, tanto los desarrolladores propios de la organización, como proveedores externos de software, tendrán claro, cuál es su rol dentro de un proyecto y su marco de acción.

Los Sistemas de información en la actualidad

Actualmente, los sistemas de información son aplicaciones diseñadas por los analistas, y creadas por los programadores, con la finalidad de cubrir las necesidades de las organizaciones en sus distintas áreas de operación.

Durante el inicio de la utilización de los sistemas de información, su forma de operar era por lotes, lo que quiere decir, que se les ingresaba gran cantidad de información y luego era procesada.~~,~~ ~~posteriormente~~posteriormente, se obtenían listados o reportes que constituían el resultado de dicho procesamiento.

Luego se evolucionó a las **aplicaciones en línea**, que para Senn (1992), sus principales características son “la respuesta inmediata a las solicitudes del usuario, demanda poco predecible y contacto directo entre la computadora y el usuario.” (pág, 576)

Las comúnmente conocidas, **aplicaciones cliente servidor**, son un claro ejemplo de la definición destacada anteriormente, ya que en ellas el usuario ingresa un dato y ejecuta una acción, y la aplicación contesta inmediatamente a su solicitud. También lo son el caso de las aplicaciones que pueden ser utilizadas a través de Internet, o en las intranets de las organizaciones con la ayuda de un *browser*.

Con el paso del tiempo y la experiencia acumulada de las personas que trabajan en el desarrollo de sistemas, se ha generado la necesidad de establecer pautas o lineamientos para el desarrollo de los sistemas, ~~los cuales~~que han dado origen a **estándares de programación**, y a la **arquitectura de software** (AS).

Para Reynoso (2004) “todavía no se ha escrito una historia aceptable de la AS. desde que Mary Shaw o David Garlan reseñaran escuetamente la prehistoria de la especialidad a principios de los 90” (Parr. 1), y según este mismo autor “la AS acostumbra remontar sus antecedentes al menos hasta la década de 1960” (Parr 3), pero aclara que su historia no ha sido tan continua como la de la ingeniería de

software. También menciona que por lo tanto es una práctica joven, que en el 2004, tenía apenas unos doce años de trabajo constante, y que en ese momento experimentaba una nueva ola creativa en el desarrollo de sus técnicas.

Comenta Reynoso (2004) que “se reconoce que en un principio, hacia 1968, Edsger Dijkstra, propuso que se establezca una estructuración correcta de los sistemas de software antes de lanzarse a programar, escribiendo código de cualquier manera” y que Edsger Dijkstra, “fue uno de los introductores de la noción de sistemas operativos organizados en capas que se comunican sólo con las capas adyacentes y que se superponen (como capas de cebolla)” (Parr. 5) Como se puede observar a lo descrito en este párrafo, las bases o fundamentos para la arquitectura de componentes en “n” capas, con gran auge en la actualidad, se remontan a los mismos principios de la AS.

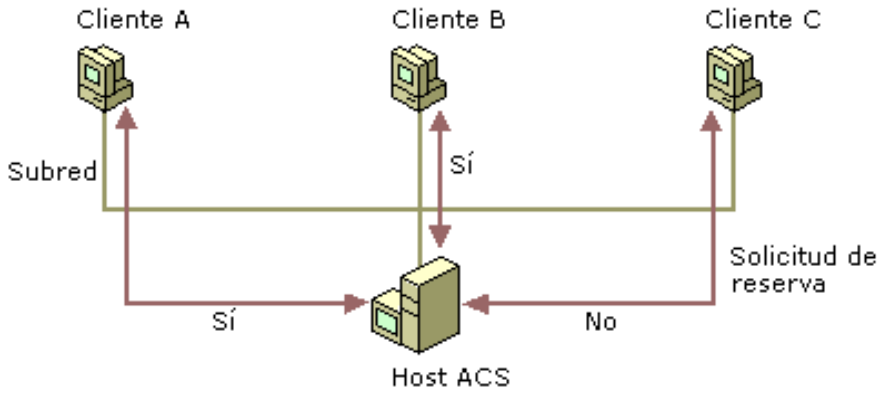
En Wikipedia (2006) se puede encontrar la siguiente definición: “Una Arquitectura de Software, también denominada Arquitectura lógica, consiste en un conjunto de patrones y abstracciones coherentes que proporcionan el marco de referencia necesario para guiar la construcción del software para un sistema de información.
La arquitectura de software establece los fundamentos para que analistas, diseñadores, programadores, etc. trabajen en una línea común que permita alcanzar los objetivos y necesidades del sistema de información.” (Parr. 2 y 3)

Dos tecnologías de las más representativas en los últimos años para el desarrollo de aplicaciones son:

1. **Cliente-servidor**. En esta arquitectura la carga de procesamiento es repartida en dos computadores; el servidor, que es el que se encarga de procesar todas las solicitudes del cliente, que por lo general, se concentran en solicitudes de datos de la base de datos, mientras que el cliente, es el encargado de dar el tratamiento adecuado a la información retornada por el servidor.

La siguiente imagen muestra en forma simplificada la arquitectura cliente/servidor.

Figura #1
"Arquitectura cliente/servidor"

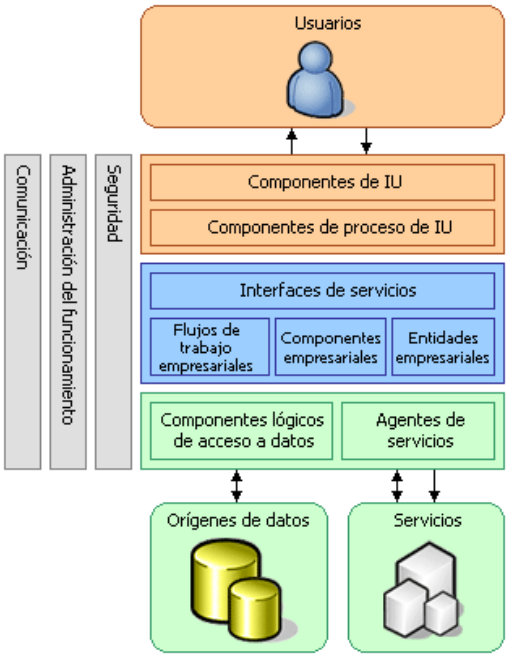


Fuente:
http://www.microsoft.com/windows2000/es/advanced/help/default.asp?url=/windows2000/es/advanced/help/sag_qosacsscene.htm

2. **Arquitectura de "n" capas**. Para esta arquitectura, la carga de procesamiento se divide en al menos tres partes o capas, con una distribución clara de funciones a implementar en cada capa: una capa para la presentación, otra, para las reglas del negocio y otra, para el almacenamiento de los datos. Una capa solamente tiene relación con la siguiente.

La siguiente imagen muestra la arquitectura de una aplicación distribuida.

Figura #2
"Capas de componentes de servicios y aplicaciones distribuidas creadas con .NET"



Fuente: Microsoft Corporation. Patterns & Practices.

En la actualidad, este tipo de arquitectura está tomando mucho auge, tanto por la organización del código dentro de la aplicación, como por la importancia de la reutilización del código contenido en las aplicaciones.

Arquitectura “n” capas

La arquitectura de “n” capas permite alcanzar el concepto de “clientes livianos” de una forma más estructurada y ordenada en cuanto a la codificación, dejando en las capas de reglas de negocios y almacenamiento de los datos, el grueso de la funcionabilidad, y en el cliente, únicamente el despliegue de información y la comunicación directa con el usuario final.

La ventaja de este tipo de aplicaciones es que el grueso de la inversión en equipo se concentra en los servidores (de aplicaciones y de base de datos), mientras que los clientes pueden ser computadoras con un poder de procesamiento no tan elevado como las últimas versiones de las computadoras personales que se encuentran en el mercado.

Si incorpora el término “escalabilidad” a esta arquitectura, y se toman algunas capas de la aplicación y se ubican en otros equipos, para distribuir aun más la carga de procesamiento de los componentes de la aplicación, se obtiene como resultado una “Aplicación distribuida”.

Las aplicaciones distribuidas solucionan muy bien el problema al que se enfrentan los departamentos de TI, cuando el volumen de transacciones de alguna aplicación aumenta, como consecuencia lógica del crecimiento de la organización, lo que se hace es tomar un componente de la aplicación, que es el que está generando más consumo de recursos, y moverlo a otro servidor, ~~el cual~~que tendrá toda su capacidad de procesamiento disponible, para cumplir con las funciones requeridas del componente mencionado, de esta manera, no se tuvo que modificar la aplicación, ni se tuvo que hacer ningún "*upgrade*" del equipo de cómputo utilizado, pero si se resolvió el problema de una manera bien sencilla.

Con formato: Fuente: Cursiva

La incorporación de nuevas tecnologías a la forma de hacer negocios, como ~~lo son~~ los dispositivos móviles (*PDA's*, *Tablet's* y teléfonos celulares), han impulsado aun más, dicha arquitectura, ya que por ejemplo, el desarrollo de una aplicación, para el registro de las ventas de "x" artículos (facturación), que tradicionalmente se utilizaba en la bodega principal de la compañía~~,~~; se puede ~~modificando~~tomar únicamente la capa del cliente y modificarla (desarrollando una nueva interfase de usuario), de esta forma p~~o~~ued~~ría~~e ser utilizada también, desde una posible sucursal a través de Internet con un programa navegador, o de igual forma (modificando la capa del cliente), se puede~~odría~~ implementar la misma aplicación en el teléfono móvil de un vendedor de la empresa, y realizar la venta ~~(factura) de los artículos en el momento preciso en que se realiza la venta.~~desde dicho

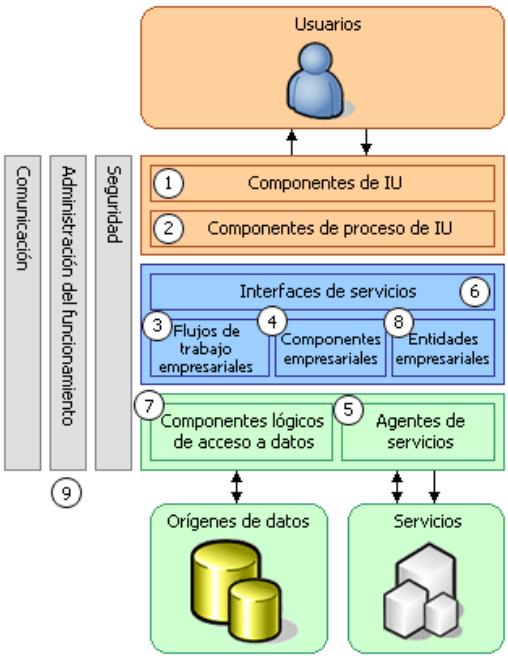
dispositivo interactuando directamente con el bloque principal de la aplicación de facturación.

El ejemplo anterior, explica cómo se reutiliza ~~todo~~ el código previamente ~~desarrollado~~ escrito para una aplicación de facturación, desarrollando únicamente las interfases, para los nuevos dispositivos en los que deberá funcionar la aplicación.

Diseño de una aplicación "n" capas distribuida

A continuación, se describe la forma de~~n~~ c~~o~~ómo está compuesta una aplicación de "n" capas, así como los componentes que integran cada capa y el modo en que cada uno de ~~los cuales~~ellos, realiza una tarea diferente. Todas las aplicaciones contienen tipos de componentes similares, independientemente de las necesidades empresariales que deban cubrir. La siguiente figura ilustra lo antes mencionado, y posteriormente, la explicación de cada componente.

Figura #3
"Tipos de componentes utilizados"



Fuente: Microsoft Corporation. Patterns & Practices.

Tipos de componentes

La figura anterior, representa un ejemplo de una aplicación comercial. Los tipos de componentes identificados en el escenario de diseño del ejemplo, son los siguientes:

1- Componentes de interfaz de usuario (IU). Las aplicaciones del tipo que se ha venido discutiendo en el desarrollo de este documento, deben permitir al usuario tener una manera de interactuar con la misma.

Las interfaces de usuario son desarrolladas utilizando formularios de *Windows Forms*, páginas ASP.NET, controles u otro tipo de tecnología que permita procesar y dar formato a los datos de los usuarios, así como validar los datos entrantes.

2- Componentes de proceso de usuario. En este tipo de aplicaciones la interacción del usuario con el sistema se realiza en el componente (1), explicado anteriormente, por lo que las interacciones en este componente de la aplicación, se realizan de acuerdo a un proceso predecible.

La utilización de componentes de proceso de usuario individuales, facilita la organización de las interacciones con el usuario, ya que de esta manera, el código escrito para controlar el flujo del proceso y la lógica de administración, no se incluye en el componente de (IU), lo que permite que varias aplicaciones con interfaces distintas, o diferentes interfaces en la misma aplicación, puedan utilizar el mismo componente de interacción básica.

3- Flujos de trabajo empresariales. Una vez que el proceso de usuario ha recopilado los datos necesarios, éstos se pueden utilizar para realizar un proceso empresarial.

La mayoría de los procesos empresariales implementados en las aplicaciones, involucran la realización de varios pasos, los mismos deben estar claramente definidos y se deben llevar a acabo en un orden determinado.

Estos procesos empresariales que constan de varios pasos de ejecución larga, según el autor "se pueden implementar utilizando herramientas de administración de procesos empresariales, como *BizTalk Server Orchestration*" (Microsoft. 2002a), con lo que se logra redireccionar los esfuerzos de desarrollo en áreas más sensibles, y dejar como administrador de estos flujos, programas ampliamente probados como el descrito anteriormente.

4- Componentes empresariales. Son todos aquellos que implementen reglas empresariales y realicen tareas empresariales (lógica empresarial de la aplicación), por lo tanto, toda aplicación requiere el uso de estos componentes, independientemente de si el proceso empresarial consta de un único paso o de un flujo de trabajo, como el descrito en el componente anterior, este componente deberá ser utilizado.

5- Agentes de servicios. Están constituidos por código para administrar la semántica de la comunicación con algún servicio externo, ellos son el vínculo entre un componente empresarial que requiera el uso de alguna funcionalidad proporcionada por un servicio externo.

Los agentes de servicios, además de administrar las llamadas a varios servicios desde la aplicación, pueden proporcionar servicios adicionales, como el formato de los datos que expone el servicio de formato que requiere la aplicación.

6- Interfaces de servicios. Para exponer la lógica empresarial como un servicio, se requiere crear interfaces de servicios que admitan los contratos de comunicación "(comunicación basada en mensajes, formatos, protocolos, seguridad y excepciones, entre otros) que requieren los clientes." (Microsoft. 2002a)

7- Componentes lógicos de acceso a datos. La mayoría de las aplicaciones y servicios necesitan obtener acceso a un almacén de datos en un momento determinado del proceso empresarial.

Para obtener acceso a los datos, es recomendable abstraer la lógica en una capa independiente de componentes lógicos de acceso a datos, lo ~~cual~~ que permite

centralizar la funcionalidad del acceso a los datos, y además, facilita la configuración y el mantenimiento del código en este componente.

8- Componentes de entidad empresarial. Como se explicó en el componente (7), la mayoría de aplicaciones realizan en algún momento accesos a un almacén de datos, luego los datos deben pasar entre los distintos componentes de la aplicación.

Las entidades empresariales que se utilizan de forma interna en la aplicación para pasar los datos entre componentes, suelen ser estructuras de conjuntos de datos, como *DataReader* o secuencias de lenguaje de marcado extensible (*XML*), aunque también se pueden implementar utilizando clases orientadas a objetos personalizadas.

9- Componentes de seguridad, administración operativa y comunicación. Para este grupo de componentes, las aplicaciones deben utilizar también componentes que manejen la administración de excepciones, autoricen a los usuarios a que realicen tareas determinadas y comunicarse con otros servicios y aplicaciones.

¿Cómo diseñar los componentes?

Como se explicó anteriormente, una aplicación distribuida, está conformada por muchos componentes, en esta sección, se desarrollará la explicación de cómo diseñar los componentes de tres de las capas descritas previamente: ~~, las cuales son:~~ capa de presentación o interfase de usuario, capa de negocios o componentes empresariales y la capa de datos.

Diseño de capas de presentación

En la capa de presentación se encuentran los componentes necesarios que permiten a los usuarios interactuar con la aplicación. Las capas de presentación van desde las más simples hasta las más complejas, según Microsoft, las más simples “contienen componentes de interfaz, como formularios de Windows Forms o formularios Web de ASP.NET” (Microsoft. 2002a), mientras que las más complejas “conllevan el diseño de componentes de proceso de usuario que permiten organizar los elementos de la interfaz y controlar la interactuación con el usuario. Los componentes de proceso de usuario resultan especialmente útiles cuando la interactuación del usuario sigue una serie de pasos predecibles, como al utilizar un asistente para realizar una tarea determinada.” (Microsoft. 2002a)

Normalmente las interfaces de usuario constan de una página o formulario con varios elementos que permiten mostrar datos, además de aceptar la entrada del usuario.

Dentro de las funciones de los componentes de la interfaz de usuario se encuentran: “mostrar datos al usuario, obtener y validar los datos procedentes del mismo e interpretar las acciones de éste que indican que desea realizar una operación con los datos. Asimismo, la interfaz debe filtrar las acciones disponibles con el fin de permitir al usuario realizar sólo aquellas operaciones que le sean necesarias en un momento determinado.” (Microsoft. 2002a)

Diseño de capas empresariales

Cualquier aplicación realiza al menos un proceso empresarial, ~~el cual~~que consta de una o varias tareas, y que desde los casos más simples, cada tarea se puede encapsular en un método de un componente .NET y ser llamado de forma sincrónica o asincrónica.

Los componentes empresariales, además de contener la lógica de negocios, también pueden realizar solicitudes de servicios externos, con el fin de obtener

alguna información ajena a la aplicación, pero útil para algún proceso, para realizar ésto, se recomienda la utilización de *web services*.

Los componentes empresariales implementan las reglas del negocio, aceptan y devuelven estructuras de datos simples o complejas. Dichos componentes deben exponer su funcionalidad de modo que sea independiente de los almacenes de datos y servicios requeridos para la realización de su tarea.

En el caso que un proceso empresarial tuviera que invocar a otros procesos empresariales dentro de una transacción atómica, todos los procesos invocados deben garantizar que sus operaciones participan en la transacción existente, por lo tanto, si la lógica empresarial que realiza las llamadas se interrumpe, todas las operaciones dentro de la transacción deben deshacerse.

Diseño de capas de datos

Casi todas las aplicaciones empresariales y servicios necesitan tener acceso a leer y almacenar su información en algún repositorio de datos.

La mayoría de las aplicaciones empresariales utilizan una base de datos relacional para almacenar sus datos, pero también se puede utilizar el sistema de archivos o

servicios de administración de documentos. Por lo que la aplicación o servicio puede disponer de uno o varios orígenes de datos, que a su vez pueden ser de tipos diferentes.

Es debido a esto, que las aplicaciones o servicios deben incorporar componentes lógicos de acceso a datos que proporcionen los métodos necesarios para la consulta y actualización de datos. Su función, es abstraer la semántica del almacén de datos y proporcionar una interfaz simple de programación para la recuperación y realización de operaciones con datos.

Cuando estos componentes obtienen datos de la base de datos, deben almacenarlos inicialmente en un objeto con formato de conjunto de datos, como puede ser un *DataReader* o un *DataSet*, este último objeto no es recomendado utilizarlo ya que es más ineficiente que un *DataReader*. Después de recuperados los datos, éstos se deben transferir entre las distintas capas y niveles de la aplicación para que el componente que realizó el proceso haga uso de ellos. Es más recomendable para transferir datos entre capas y niveles de la aplicación, utilizar estructuras de datos menos complejas o más livianas, y que permitan serializarse para su transmisión, que utilizar un objeto complejo como un DataReader, por ejemplo, puede utilizar un objeto de entidad personalizado, como una clase.

Componentes lógicos de acceso a datos

Estos componentes de acceso a datos separan el procesamiento empresarial (reglas de negocio) de la lógica de acceso a datos. Dichos componentes, cuentan en su mayoría, con los métodos para realizar operaciones a la base de datos como *Create*, *Read*, *Update* y *Delete* (CRUD), ~~las cuales~~que van relacionadas a una entidad empresarial determinada (tabla) de la base de datos.

Microsoft proporciona una serie de código fuente, agrupado en bloques con distintas funcionabilidades, para este caso de componentes en específico es el *Data Access Application Block para .NET*, que se puede ~~utilizar~~ emplear como un componente de ayuda de acceso a datos genéricos en la aplicación, al utilizar bases de datos SQL Server. Este tipo de bloques de código se explica más detalladamente en la siguiente sección.

Herramientas a utilizar

Visual Studio 2005 Team System

Para González (2005) "*Visual Studio 2005 Team System* (VSTS) es la propuesta de Microsoft para maximizar la productividad de los equipos TI de una organización. VSTS está compuesto por un conjunto de herramientas integradas, productivas y extensibles, que apoyan dramáticamente a cada uno de los roles, fases, y aspectos del ciclo de vida de desarrollo de sistemas, permitiendo a los equipos y sus miembros TI de toda una organización reducir la complejidad del desarrollo, facilitando una mejor comunicación y colaboración durante el proceso. Como consecuencia de esto se incrementa el nivel predecible del éxito de los proyectos de desarrollo que la organización emprende."

Debido al gran impulso tecnológico y metodológico que ha realizado Microsoft en materia de desarrollo de sistemas (arquitecturas y mejores prácticas de programación), y la gran cantidad de material de estudio, ejemplos y casos de éxito que existen empleando las recomendaciones de Microsoft en Internet y en la literatura, se ha decidido para efectos de esta propuesta, recomendar la utilización de herramientas y metodología brindadas por Microsoft; lo ~~cual~~ que requiere el empleo de al menos dos herramientas que se encuentran contenidas dentro del VSTS, la primera, *Visual Source Safe 2005* (VSS) para compartir el código entre los programadores que forman parte del equipo de desarrollo y la administración

de versiones, y *Visual Studio 2005 Team Edition for Software Developers*, que es la aplicación que cuenta con las herramientas de desarrollo con las ~~cuales~~ que se elabora el código necesario para la construcción de las aplicaciones.

Visual Source Safe 2005

Esta herramienta está diseñada para el control de versiones de los proyectos, y permite una gran integración con los demás productos contenidos en el *Visual Studio*®. Dicha herramienta permite a los desarrolladores y otras personas que comparten documentos, realizar alteraciones seguras de los documentos utilizados, además, le permite llevar control de los cambios realizados por los usuarios y las horas. También, ~~permite~~ contar con puntos para restaurar código o información de otra índole, ~~además~~ y de modificaciones en paralelo de un mismo documento o programa.

Visual SourceSafe es un sistema de control de versiones que: protege a los usuarios de pérdidas accidentales de sus archivos, permite el rastreo de versiones anteriores de un archivo;~~,~~ ~~también permite~~ la separación, el compartir, el mezclar, y la administración de los archivos y sus versiones.

Se encuentra orientado al mantenimiento del histórico de cambios de los diferentes archivos, rastros de auditoria, y recuperación en casos de desastre.

Visual Studio 2005 Team Edition for Software Developers

Este producto provee herramientas avanzadas de desarrollo de software, que para esta propuesta, es el contenido que interesa, aunque también cuenta con un conjunto integrado de herramientas de análisis que ayudan a detectar defectos en el código y problemas en el desempeño de las aplicaciones, todo esto, de manera temprana en el ciclo de desarrollo del programa o aplicación, o sea, antes de que entre en producción.

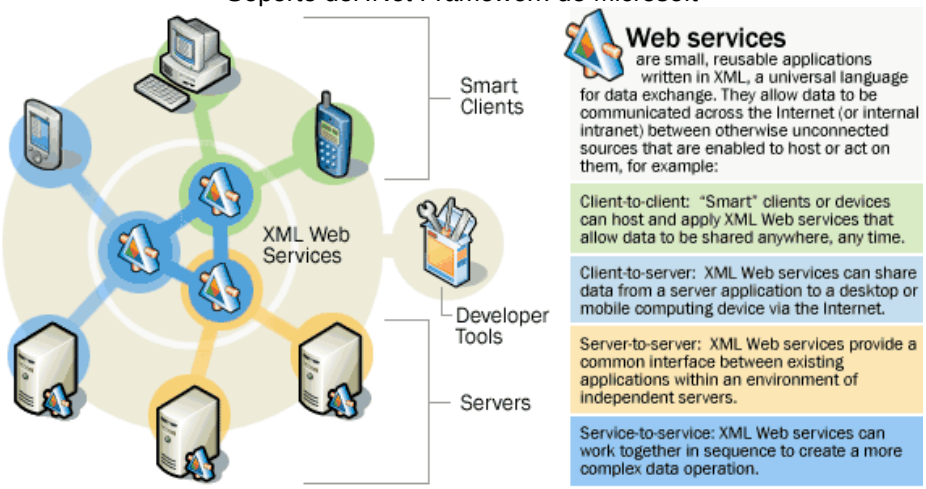
También estas herramientas aseguran la producción de programas de calidad, usando *testing* y herramientas de cobertura de código, para determinar la efectividad de las pruebas.

Las aplicaciones creadas con las herramientas de desarrollo del Visual Studio 2005, pueden ser ejecutadas en todas aquellas máquinas o dispositivos que cuenten con el *Framework 2.0* de Microsoft instalado, con excepción de las que fueron diseñadas para funcionar con un *browser*.

El ***Framework 2.0*** es un ambiente de desarrollo y ejecución que permite a varios lenguajes de programación y librerías, trabajar juntos para la creación y ejecución de las aplicaciones basadas en *Windows*. “Administra gran parte de los detalles de infraestructura, permitiendo a los desarrolladores centrarse en escribir el código de la lógica empresarial para sus aplicaciones” (Microsoft. 2002b), el *.NET Framework* también incluye el *Common Language Runtime* y bibliotecas de clases.

El *.Net Framework* provee la infraestructura básica que las aplicaciones basadas en Windows necesitan para realizar una conexión entre información, gente, sistemas, y dispositivos a través de la visión .Net de Microsoft. El *.Net Framework* de Microsoft soporta: los protocolos estándar de red, diferentes lenguajes de programación, librerías desarrolladas en diferentes lenguajes, y diferentes plataformas, lo cual se verá ilustrado en la siguiente imagen.

Figura #4
"Soporte del *.Net Framework* de Microsoft"



Fuente: http://msdn.microsoft.com/netframework/gettingstarted/default.aspx

***Enterprise Development Reference Architecture* (EDRA)**

EDRA es un modelo para construir aplicaciones distribuidas con las siguientes características: proyectos empresariales escalables, con una arquitectura avanzada, que sean grandes y diversos, y con equipos de desarrollo distribuidos. También trae consigo guías para la implementación de patrones de diseño y los *applications blocks*. EDRA no es un producto soportado por Microsoft, ni es tampoco el único modo de construir una aplicación distribuida, pero para efectos de esta propuesta es el que se utilizará.

Los desafíos de esta arquitectura de referencia, se enfocan en las áreas que se mencionan a continuación, y de la siguiente forma:

Mensajería: ¿cómo manejar mensajes duplicados? ¿cómo manejar los tiempos de espera vencidos? ¿cómo recibir mensajes de clientes sobre múltiples canales?

Transacciones: ¿cómo soportar transacciones? ¿cómo y cuándo se deberían usar transacciones atómicas?

Manejo: ¿cómo istrumentar los servicios? ¿cómo monitorear los servicios? ¿cómo publicar eventos para los servicios?

Arquitectura: ¿cómo separar el código de la lógica de negocios? ¿cómo separar la interfase del servicio de la implementación del servicio? ¿cómo proveer un modelo de programación asincrónica simple?

Seguridad: ¿cómo autenticar usuarios? ¿cómo autorizar usuarios? ¿cómo validar mensajes?

La resolución de todos estos desafíos planteados anteriormente, conducen necesariamente a la utilización de los *applications blocks* ~~los cuales~~que ~~,~~ se describen brevemente a continuación.

Enterprise Library (Aplication Blocks)

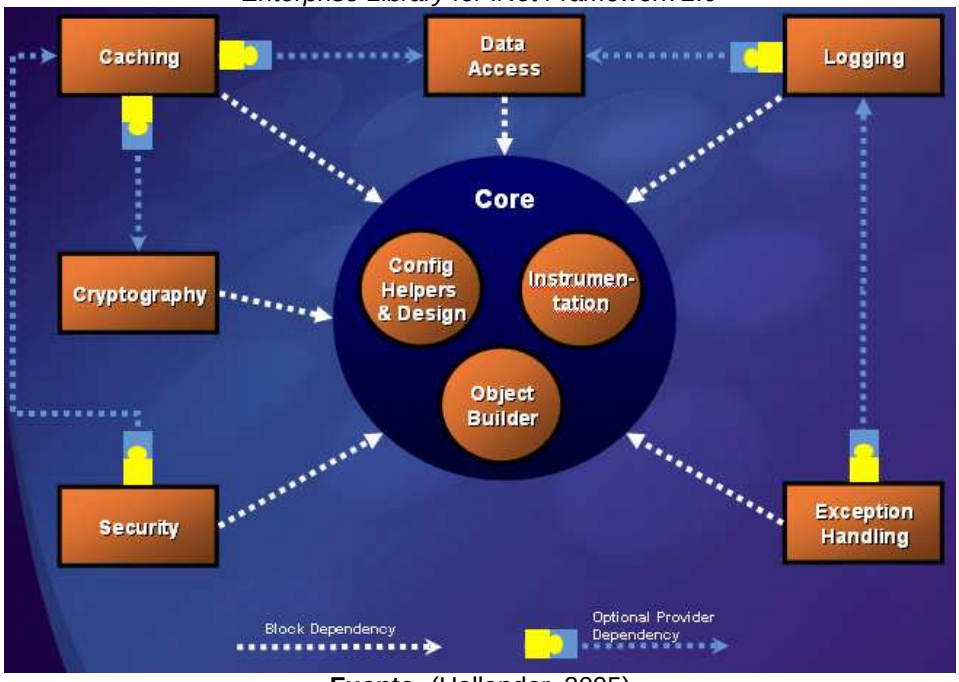
Los *Application Blocks* son componentes de código fuente, reusables y extensibles que proveen una guía para el desarrollo de las aplicaciones.

El *Enterprise Library* es una librería que contiene siete *application blocks*, los cuales son: "*Caching, Configuration, Cryptography, Data Access, Exception Handling, Logging & Instrumentation, y Security*" (Hollander, 2005).

Como se mencionó anteriormente, el *Enterprise Library* es una librería de *applications blocks* ~~los cuales~~que son utilizados por los programadores en el uso diario, para resolver retos comunes del trabajo, este conjunto de clases de ayuda, funcionan en cualquier estilo de arquitectura de aplicaciones, además, se encuentran en código fuente, para permitirle a los desarrolladores, modificarlos y extenderlos, según sean las necesidades de la empresa, y se encuentran disponibles para descargarlos en Internet de forma gratuita. No son parte del *.Net Framework* de Microsoft, por lo que Microsoft no les da soporte.

En la siguiente imagen, se muestra cómo está conformado el *Enterprise Library* y la interrelación que existe entre los diferentes *applications blocks*.

Figura #5
"*Enterprise Library for .Net Framework 2.0*"



Fuente: (Hollander, 2005)

Conclusiones y Recomendaciones

Al definir la empresa una arquitectura de “n” capas como estándar para los nuevos desarrollos, la institución obtiene una serie de beneficios que representan mejoras tanto en aspectos técnicos como a nivel organizacional, ya que permite enfocarse en la administración de los recursos: personal, tecnología, tiempo y dinero.

Con respecto a las mejoras técnicas, se puede destacar la independencia del código de los programas de las “fuentes de los datos”, la disminución en el esfuerzo realizado por los programadores para dar mantenimiento a las aplicaciones, estandarización del código a todos los niveles de las diferentes capas que componen las aplicaciones, la reutilización del código y la escalabilidad, este último aspecto, muy importante en la visión de crecimiento de la aplicación según la carga de trabajo que va a soportar.

No se puede dejar de lado la calidad de los productos resultantes, ya que al ser las aplicaciones desarrolladas en unidades separadas (capas y componentes), a éstas se les realizan muchas pruebas y controles con mucho detalle, lo que permite obtener un producto muy estable.

Con este panorama, los recursos del departamento de Sistemas de Información pueden dedicarse a funciones específicas (“especialización” en diversas áreas), lo

que les permitiría dar una mayor producción debido al alto grado de conocimiento y práctica con el que contarían.

Lo anterior, conllevaría a que el mantenimiento a las aplicaciones se simplifique considerablemente ya que todas siguen la misma línea, y los tiempos en la curva de aprendizaje de los programadores encargados disminuye, ya que al entender y conocer cómo se encuentra diseñado un componente, todos se conceptualizan igual.

La reutilización del código es un aspecto fundamental en la reducción del costo de mantenimiento, ya que como se comentó anteriormente en este documento, la funcionabilidad encapsulada en cada componente, puede ser utilizada en otros módulos de una misma aplicación, o inclusive en otras aplicaciones que requieran de una funcionabilidad ya existente y probada, como es el caso de las distintas interfaces de usuario, como interfaces para PDA's, interfaces para web, o interfaces para Windows, que pueden reutilizar funciones de otros sistemas ya existentes.

Hay que recalcar que cualquier desarrollo nuevo que se vaya a realizar, por pequeño y simple que parezca, se debe ajustar al diseño adoptado por la organización, ya que, de no hacerse, no sería posible alcanzar todos los beneficios de este esquema, expuestos anteriormente.

El personal de sistemas se puede apoyar también en herramientas que han sido utilizadas con éxito por otras organizaciones, como es el caso del .NET Framework de Microsoft, en el ~~cual~~ que se ha basado gran parte de este documento, que ofrece además, una sólida plataforma para el desarrollo de aplicaciones en 'n' niveles. Debido a la utilización de estas fuentes externas de información, es importante chequear periódicamente el sitio web de los fabricantes del *software*, ya que en él, constantemente se publica información sobre mejores prácticas a la hora de diseñar estas arquitecturas e implementarlas, además de consejos y actualizaciones de código que pueden ser utilizados para mejorar el desempeño de las aplicaciones.

Un tema en el que no se profundizó en este documento, es el referente a las transacciones dentro y entre los componentes, debido a que se sale un poco de lo que es el diseño de la arquitectura propuesta, pero que no deja de tener relevancia en el control de los procesos; para más Información relacionada a transacciones se puede tomar como guía lo siguiente: "Las transacciones de los servidores que ejecutan Windows se pueden administrar utilizando el Controlador de transacciones distribuidas (DTC), utilizado por .NET Enterprise Services (COM+). Para administrar transacciones distribuidas en entornos heterogéneos, puede utilizar COM *Transaction Integrator* (COMTI) y *Host Integration Server* 2000. Si desea obtener más información sobre COMTI y *Host Integration Server*,

se puede consultar la dirección en Internet http://www.microsoft.com/hiserver (en inglés).”

Bibliografía

(Hollander T., Densmore S., Jacobs R., 2005) Enterprise Library for .NET Framework 2.0 (patterns & practices Live!). Microsoft Corporation.

Microsoft Corporation. (2002) a. Patterns & Practices. Arquitectura de aplicaciones de .NET: Diseño de aplicaciones y servicios. Recuperado de http://www.microsoft.com/spanish/msdn/arquitectura/das/distapp.asp el 8/05/2006.

Microsoft Corporation. (2002) b. Información general del producto (.NET Framework). Recuperado de http://www.microsoft.com/latam/netframework/producto/resumen.asp el 12/06/2006.

Microsoft Corporation. (2004). Introducción a la Arquitectura de Software. Carlos Billy Reynoso. UNIVERSIDAD DE BUENOS AIRES. Versión 1.0. Recuperado de http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/intro.asp el 6/06/2006.

Microsoft Corporation. Enterprise Development Reference Architecture (EDRA). Microsoft patterns & practices. Recuperado de http://www.microsoft.com/practices el 01/11/2005.

Microsoft Corporation. (Haaron González, 2005) Visual Studio 2005 Team System, herramientas integradas para un efectivo desarrollo. Recuperado de http://www.microsoft.com/spanish/msdn/comunidad/mtj.net/voices/MTJ_4300.asp el 12/06/2006.

Senn, James. (1992). Análisis y Diseño de Sistemas de Información. Segunda Edición. Ed. McGRAW-HILL: Mexico.

Wikipedia. (2006). Arquitectura de software, breve reseña histórica. Recuperado de http://es.wikipedia.org/wiki/Arquitectura_software el 29/05/2006.