Universidad de las Ciencias Informáticas
Facultad 4

Título: “Diseño de la arquitectura del proyecto TeleBanca”

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas

Autor(es): Erick Fleitas Chang

Robert Alberto Peña Yantá

Tutor: Lic. David Batard Lorenzo.

Ciudad de la Habana

Año 2007

DECLARACIÓN DE AUTORÍA

Declaramos que somos los únicos autores de este trabajo y autorizamos a la Facultad 4 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmamos la presente a los ____ días del mes de _____ del año 2007.

Erick Fleitas Chang

_____

Firma del autor

Robert Alberto Peña Yantá

_____

Firma del autor

David Batard Lorenzo

_____

Firma del tutor

DEDICATORIA

Erick:

A mi madre Eunice, por ser la inspiración de mi vida, mi amiga, mi apoyo moral, mi refugio, el origen de mis sueños y la fuente de mis fuerzas. Gracias a ti tuve la mejor formación que pudiera pedir, no me arrepiento ni por un instante de todos aquellos golpes y regaños que me dabas por ser un cabeza dura, es más, los extraño un montón. Las veces que tropecé por no hacerte caso, pero tú siempre estabas ahí para hacerme reflexionar. Son infinitas las palabras que quedaron pendientes entre nosotros y son tan tantas las cosas que deseo expresarte pero no hay documento que soporte todo lo que quiero decirte, porque lo que siento por ti es inmenso, porque te amo con todas mis fuerzas y siempre estarás junto a mí, en mi corazón.

A mi hermana Cinthia, por ser una de las personas que más quiero en la vida, por darme su apoyo y cariño, por ser paciente conmigo, por hacerme enojar en ocasiones, por haberse convertido en una mujer y por obligarme a tratar de ser un buen ejemplo a seguir.

A mi abuela Silvia, por haberme malcriado tanto, por ser tan buena y por darme su amor.

A Maria Elena por haberme guiado en todo este tiempo, por todo su amor, sus consejos y sus regaños, por nunca perder la fe en mí. Gracias Minena.

A Chirino por ser mi amigo, mi hermano, mi familia, por compartir sus sabios consejos.

A Jesús y Javier por ser mis hermanos.

A Yoarys por todo su amor y su cariño, por estar conmigo en las buenas y en las malas.

A Rosa por ser la mejor suegra del mundo, por su cariño y su apoyo.

A Elien por su apoyo, por ser mi hermano y mi amigo.

A mi familia que tanto me ha apoyado en todos estos años, siempre se los voy a agradecer.

A todos mis amigos y profesores de la universidad que me han apoyado y me han brindado su amistad. A ustedes que han depositado su confianza en mí.

A la UCI por haberme formado como profesional y por todo el apoyo que me dieron en la etapa más difícil de mi vida.

A todos mis amigos, son tantos los que han estado a mi lado, que no puedo darme el lujo de pasar por alto el nombre de alguno de ustedes, saben lo mucho que representan para mí.

A la inmensa familia que tengo, que aunque no lleve su sangre, forman parte de mi ser.

Robert:

A mi familia que me apoyó anímica, moral, material y económicamente durante todos estos años:

A mi madre por ser mi fuente de inspiración, por darme todo el apoyo que necesité en los momentos más difíciles, por su incondicionalidad y su eterno amor.

A mi padre, por tu ejemplo, por querer que fuese siempre alguien en este mundo, por sus consejos y sobre todo por su plena confianza en mí.

A mi hermana, la que me guió y me dio las fuerzas necesarias para seguir adelante, por ser un ejemplo a seguir en la vida, por ser la que me entiende más que nadie y la que me soporta todos mis egoísmos.

A Kelvis por su gran apoyo y por confiar plenamente en mi. Por ser mi hermano y mi amigo.

A todos mis amigos por apoyarme en todo momento y confiar en mí, gracias por compartir lo mejor de sí mismos y por aceptarme en sus vidas, los quiero.

De ambos: A nuestro Comandante en Jefe Fidel Castro Ruz y a la Revolución.

AGRADECIMIENTOS

A todas aquellas personas que hicieron posible que esta investigación se hiciera realidad, a nuestros amigos y hermanos de tantos años en la universidad que nos han brindado toda su confianza y su apoyo en especial Yeslidier, Lissett, Alejandro, Osniel, Johan y Miguel.

A nuestros profesores que desde primer año dieron lo mejor de sí para formarnos como mejores profesionales.

A David Batard y a Yulierky Torres por la preocupación y la ayuda que nos proporcionaron en la elaboración de esta investigación.

RESUMEN

El presente trabajo de diploma contiene un estudio de los elementos fundamentales que constituyen la Arquitectura de Software, durante su desarrollo se realiza un análisis de estos elementos con el objetivo de lograr una organización de la estructura del Sistema TeleBanca.

El éxito de esta aplicación viene dado por las decisiones arquitectónicas que fueron tomadas durante su desarrollo, por la definición de las tecnologías y aspectos esenciales de diseño y suministrando a los miembros del equipo de desarrollo una idea bien clara de lo que se está desarrollando.

En este documento se plasman los resultados del estudio realizado acerca de la arquitectura de software y sus principales corrientes. Se realiza una investigación sobre el estado de los diferentes tipos de patrones que intervienen en la materialización de una arquitectura para un sistema de software. Así como la descripción y construcción de la solución propuesta.

ÍNDICE

INTRODUCCIÓN

La Informática como ciencia que estudia el tratamiento automático de la información. Está sujeta a los cambios tecnológicos que aparecen continuamente con tendencia al desarrollo constante. Es una corriente innovadora, dinámica y muy popular que ha cautivado la atención de la población y de las empresas a nivel internacional.

Mediante la tecnología digital se pueden realizar operaciones que facilitan el intercambio de información seguro, rápido y eficiente. Debido a esto se hace necesaria la construcción de grandes y complejos sistemas de software que requieren de la combinación de diferentes tecnologías y plataformas tanto de hardware como de software. Para lograr esto, se requiere de profesionales dedicados al desarrollo de software con la experiencia y el conocimiento suficiente para diseñar una arquitectura sólida que soporte el funcionamiento del sistema.

Un Centro de Llamada es una prueba irrefutable de los sistemas que utilizan esta tecnología. Su uso es a nivel internacional y en especial por una variedad significativa de empresas, pues ofrecen un canal de comunicación interno y externo dentro de una organización. Tienen un elevado nivel de aceptación, porque depende solo de una llamada telefónica, donde se ofrecen una serie de servicios: como solicitar información, realizar compras, realizar el pago de servicios, atención a clientes, el estado de las finanzas de empresas, entre otros. El objetivo primordial de estos centros de llamadas es responder a las inquietudes de los clientes de forma rápida y en el menor tiempo posible. Entre los Centros de Llamadas se tienen como ejemplo: ETECSA (Cuba), 911 (Argentina), Entel (Chile).

Las aplicaciones bancarias de otros países ponen en práctica las mejores técnicas y tecnologías actuales del desarrollo de software, debido al aumento de las demandas por parte de los usuarios que requieren un incremento notable de servicios, a raíz del desarrollo actual de las telecomunicaciones (Internet, Celulares). Estos sistemas utilizan lenguajes de programación de alto nivel y su arquitectura se compone por estilos arquitectónicos como: Arquitectura en Capas, Orientada a Objetos, Orientada a Servicios, Basada en Componentes. Estas aplicaciones no se rigen por una arquitectura en específico sino que se realiza una

mezcla de diferentes estilos arquitectónicos, donde se da un matiz propio según los requerimientos a satisfacer por el sistema.

La Revolución Cubana, a pesar del recrudecimiento del criminal bloqueo impuesto por los gobiernos estadounidenses desde hace más de una década, se ha propuesto la tarea de informatizar la sociedad como una de las medidas de la Batalla de Ideas que libra nuestro pueblo, asegurando el proceso de automatización de escuelas, empresas e instituciones estatales.

El objetivo que se propone es insertar a nuestro país eventualmente entre las principales potencias de la producción de software a través de la creación de instituciones productoras de software, con el fin de ampliar nuestras posibilidades de competir en un mercado lucrativo como es el caso de la Informática y las Telecomunicaciones.

En nuestro país, la mayoría de los sistemas bancarios, están definidos por una aplicación en el lenguaje FoxPro, basada en los estilos arquitectónicos: Centrado en Datos, Flujos de Datos y Tuberías y Filtros. Estas arquitecturas no cumplen con todas las expectativas que se requieren y la aplicación que existe actualmente resulta ser, trabajosa, con demora en el tiempo de respuesta ante operaciones realizadas y con una interfaz de usuario poco amigable para el cliente. Estos sistemas centran la lógica de los procesos del negocio en las operaciones y transformaciones que se les realizan a los datos durante el proceso de almacenamiento, en otras palabras esta lógica se encuentra embebida dentro de la persistencia de los datos.

Existe problemas como la pérdida e inconsistencia de la información en operaciones bancarias que afectan significativamente el correcto funcionamiento de transacciones de este tipo, debido a que los medios para organizar y almacenar la información no son los más eficientes ni proveen la calidad requerida. La aglomeración de archivos que deben ser generados y guardados de manera organizada en locales que usan herramientas físicas, imposibilitan el buen desempeño por parte de aquellos que están al frente de todos los procesos diarios, inhabilitando la capacidad de estaciones de trabajo y espacios que pudieran utilizarse con un fin más productivo. Otros de los problemas que dificultan el trabajo son las pérdidas de información y el alto tiempo que demoran las transacciones y operaciones.

El Banco Metropolitano desea crear el sistema de software TeleBanca para realizar el pago de una serie de servicios a la población mediante un Centro de Llamadas, donde se pueda pagar servicios como la electricidad, el agua, las multas, el teléfono, brindar información, entre otros.

Este sistema necesita una arquitectura que soporte el desarrollo del ciclo de vida del proyecto, siendo esta esencial para su éxito o su fracaso. Es necesaria para la toma de decisiones importantes sobre la organización del sistema, sus elementos estructurales, sus interfaces, sus colaboraciones y su composición.

Tomando como referencia lo expuesto anteriormente, se tiene como:

Problema científico:

¿Cómo diseñar una arquitectura que permita tomar un conjunto de decisiones importantes sobre los aspectos dinámicos y estáticos más significativos del Sistema TeleBanca?

Objeto de estudio:

El proceso de diseño de la arquitectura de software sobre la tecnología .NET del proyecto TeleBanca.

Campo de acción:

El diseño y descripción de la arquitectura base para el desarrollo del sistema TeleBanca.

Objetivo general:

Diseñar una arquitectura de software para el Sistema TeleBanca, que permita al arquitecto tomar decisiones sobre los aspectos dinámicos y estáticos más significativos de la organización del sistema, la selección de elementos estructurales mediante los cuales se compone dicho sistema, sus interfaces, su comportamiento y sus colaboraciones y que permita a los desarrolladores saber con claridad que forma tiene el sistema que se está desarrollando.

Objetivos específicos:

- Definir componentes y funcionalidades reusables que agilicen el desarrollo de la aplicación.
- Desarrollar un marco arquitectónico que permita el desarrollo de la aplicación de forma paralela en cada una de las capas lógicas.
- Definir patrones de diseño útiles para el desarrollo del proyecto.

Tareas de investigación:

- Investigar las diferentes propuestas arquitectónicas a partir de consultar la bibliografía especializada.
- Comparar las distintas propuestas de arquitecturas consultadas, teniendo en cuenta aspectos positivos, negativos.
- Revisar bibliografía científica teórica usada en el desarrollo de arquitecturas para sistemas empresariales.
- Identificar patrones arquitectónicos a utilizar, fundamentación de estos patrones, así como su aplicación.
- Definir las herramientas y tecnologías a utilizar para el desarrollo.
- Diseñar una propuesta arquitectónica que cumpla con los requerimientos de la aplicación y que garantice el desarrollo paralelo y la reutilización de componentes de la misma.

Para realizar dichas tareas de investigación existen varios **métodos**:

Métodos teóricos:

Análisis y Síntesis: para el procesamiento de la información y arribar a las conclusiones de la investigación, así como precisar las características del modelo arquitectónico propuesto.

Histórico – Lógico: Para determinar las tendencias actuales de desarrollo de los modelos y enfoques arquitectónicos.

Métodos empíricos:

Observación: Para la percepción selectiva de las restricciones y propiedades del sistema y sistémica para el control de la evolución de la arquitectura inicial.

CAPÍTULO 1

1.1 Introducción

En el presente capítulo se hace un análisis del surgimiento de la arquitectura de software (en lo adelante AS) y de sus tendencias, así como la descripción de los principales conceptos asociados. Además se describen las principales herramientas y tecnologías que hoy en día juegan un rol importante a la hora de diseñar una buena arquitectura de software.

1.2 Arquitectura de Software:

La AS tiene sus raíces en 1968, donde Edsger Dijkstra, de la Universidad Tecnológica de Eindhoven en Holanda y Premio Turing 1972, propuso que se establezca una estructuración correcta de los sistemas de software antes de lanzarse a programar, escribiendo código de cualquier manera [1]. Dijkstra, quien sostenía que las ciencias de la computación eran una rama aplicada de las matemáticas y sugería seguir pasos formales para descomponer problemas mayores, fue uno de los introductores de la noción de sistemas operativos organizados en capas que se comunican sólo con las capas adyacentes

En 1975, Frederick Phillips Brooks Jr, diseñador del sistema operativo OS/360 y Premio Turing 2000, utilizaba el concepto de arquitectura del sistema para designar “la especificación completa y detallada de la interfaz de usuario” y consideraba que el arquitecto es un agente del usuario, igual que lo es quien diseña su casa [2], también distinguía entre arquitectura e implementación; mientras aquella decía qué hacer, la implementación se ocupa de cómo.

La AS siguió tomando auge con el paso de los años pero no es hasta 1992 que AS se define como disciplina de software en la publicación “Foundations for the study of software architecture” escrito por Dewayne Perry, Alexander Wolf donde planteaban [3]:

“*La década de 1990, creemos, será la década de la arquitectura de software…*”

En el año 2000 la IEEE hace la definición “oficial” de AS en su documento IEEE 1471, que reza así:

“La Arquitectura de Software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución”.

Este documento de la IEEE 1471 ha sido adoptado por todo el mundo y por todas las organizaciones que tienen algo que ver con la AS.

Esta definición, deja claro que la AS no se inscribe en ninguna metodología de desarrollo, sino que es en sí, una disciplina que se desarrolla durante todo el ciclo de desarrollo de software.

Existen otras definiciones clásicas de la arquitectura tales como:

“…más allá de las estructuras de los algoritmos y de datos del cómputo; diseñar y especificar la estructura del sistema total emerge como nueva clase de problema. Las ediciones estructurales incluyen la organización gruesa y la estructura global del control; los protocolos para la comunicación, la sincronización, y el acceso a los datos; asignación de la funcionalidad para diseñar elementos; distribución física; composición de los elementos del diseño; escalamiento y funcionamiento; y selección entre alternativas del diseño” [4].

Rational Unified Process, 1999, plantea:

“Una arquitectura es el sistema de decisiones significativas sobre la organización de un sistema de software, la selección de los elementos estructurales y de sus interfaces por los cuales el sistema es compuesto, junto con su comportamiento según lo especificado en las colaboraciones entre estos elementos, la composición de estos elementos estructurales y del comportamiento en subsistemas progresivamente más grandes y el estilo arquitectónico que dirigen esta organización, los elementos y sus interfaces, sus colaboraciones y su composición” [5].

Esta define la AS como una etapa más de la Ingeniería de Software, da una visión de la arquitectura como algo más que la construcción de herramientas o de tecnologías a usar en el desarrollo del sistema. Está enfocada a la orientación a objetos y a UML, según lo plantea la metodología de desarrollo RUP.

A continuación veremos las principales corrientes arquitectónicas:

- ***Arquitectura estructural, basada en un modelo estático de estilos, ADLs y vistas.*** Constituye la corriente fundacional y clásica de la disciplina. Los representantes de esta corriente son todos académicos, mayormente de la Universidad Carnegie Mellon en Pittsburgh: Mary Shaw, Paul Clements, David Garlan, Robert Allen, Gregory Abowd, John Ockerbloom. En toda la corriente, el diseño arquitectónico no sólo es el de más alto nivel de abstracción, sino que además no tiene por qué coincidir con la configuración explícita de las aplicaciones; rara vez se encontrarán referencias a lenguajes de programación o piezas de código.

- ***Arquitectura*** como una ***Etapa de ingeniería y diseño orientada a objetos.*** Esta corriente es una alternativa de la descrita anteriormente. Es el modelo de James Rumbaugh, Ivar Jacobson, Grady Booch, Craig Larman y otros, ligado estrechamente al mundo de UML y Rational. En esta postura, la arquitectura se restringe a las fases iniciales y preliminares del proceso y concierne a los niveles más elevados de abstracción. Importa más la abundancia y el detalle de diagramas y técnicas disponibles que la simplicidad de la visión de conjunto. La definición de arquitectura que se promueve en esta corriente tiene que ver con aspectos formales a la hora del desarrollo. Las definiciones revelan que la AS, en esta perspectiva, concierne a decisiones sobre organización, selección de elementos estructurales, comportamiento, composición y estilo arquitectónico susceptibles de ser descriptas a través de las cinco vistas clásicas del modelo 4+1 de Kruchten [6] [7].

- ***Arquitectura basada en patrones,*** esta corriente se basa principalmente en la redefinición de los estilos como patrones POSA, el diseño consiste en identificar y articular patrones preexistentes, que se definen en forma parecida a los estilos de arquitectura [8] [9] [10].
- ***Arquitectura procesual y metodologías.*** Esta surge desde comienzos del siglo XXI, con centro en el SEI y con participación de algunos arquitectos de Carnegie Mellon de la primera generación y muchos nombres nuevos de la segunda: Rick Kazman, Len Bass, Paul Clements, Felix Bachmann, Fabio Peruzzi, Jeromy Carrière, Mario Barbacci y Charles Weinstock, Intenta establecer modelos de ciclo de vida y técnicas de diseño, análisis, selección de alternativas, validación, comparación, estimación de calidad y justificación económica específicas para la arquitectura de software [11].

1.2.1 Importancia y necesidad de una arquitectura

Se necesita una arquitectura para:

- Comprender el sistema
- Organizar el desarrollo
- Fomentar la reutilización
- Hacer evolucionar el sistema

1.3 Estilos Arquitectónicos

La clave del trabajo arquitectónico tiene que ver con la correcta elección del estilo arquitectónico.

¿Qué es un estilo arquitectónico?

En el caso de los “estilos arquitectónicos” de software son arquitecturas de software comunes, marcos de referencias arquitectónicas, formas comunes o clases de sistemas.

Los estilos de arquitectura se definen como las 4C [3]:

- Componentes (Elementos)
- Conectores
- Configuraciones
- Restricciones (Constraints)

A la hora de definir un estilo arquitectónico es necesario tener en cuenta el tipo de aplicación ya que puede imponer restricciones que acotan la interacción de los componentes, además se tiene en cuenta el patrón de organización general.

Algunos de los principales estilos arquitectónicos se usan en la actualidad están divididos por Clases de Estilos las que engloban una serie de estilos arquitectónicos específicos:

1.3.1 Estilos de Flujo de datos: Esta familia de estilos enfatiza la reutilización y la modificabilidad. Es apropiada para sistemas que implementan transformaciones de datos en pasos sucesivos.

- Tuberías y Filtros (estilo arquitectónico específico): Una tubería es una popular arquitectura que conecta componentes computacionales a través de conectores, de modo que el procesamiento de datos se ejecuta como un flujo. Los datos se transportan

a través de las tuberías entre los filtros, transformando gradualmente las entradas en salidas. Este estilo se percibe como una serie de transformaciones sobre sucesivas piezas de los datos de entrada.

1.3.2 Estilos centrados en datos: Esta familia de estilos enfatiza la integrabilidad de los datos. Se estima apropiada para sistemas que se fundan en acceso y actualización de datos en estructuras de almacenamiento.

- Arquitecturas de Pizarra o repositorio: Esta arquitectura está compuesta por dos componentes principales: una estructura de datos que representa el estado actual y una colección de componentes independientes que operan sobre él. Estos sistemas se han usado en aplicaciones que requieren complejas interpretaciones de proceso de señales (reconocimiento de patrones, reconocimiento de habla).

1.3.3 Estilos de Llamada y Retorno: Esta familia de estilos enfatiza la modificabilidad y la escalabilidad. Son los estilos más generalizados en sistemas en gran escala.

- Arquitectura en Capas: En este estilo arquitectónico cada capa proporciona servicios a la capa superior y se sirve de las prestaciones que le brinda la inferior, al dividir un sistema en capas, cada capa puede tratarse de forma independiente, sin tener que conocer los detalles de las demás. La división de un sistema en capas facilita el diseño modular, en la que cada capa encapsula un aspecto concreto del sistema y permite además la construcción de sistemas débilmente acoplados, lo que significa que si se minimiza las dependencias entre capas, resulta más fácil sustituir la implementación de una capa sin afectar al resto del sistema.

- Arquitectura Orientada a Objetos: Los componentes de este estilo son los objetos, o más bien instancias de los tipos de dato abstractos. En la caracterización clásica de David Garlan y Mary Shaw [15], los objetos representan una clase de componentes que ellos llaman managers, debido a que son responsables de preservar la integridad de su propia representación. Un rasgo importante de este aspecto es que la representación interna de un objeto no es accesible desde otros objetos.

- Arquitectura basada en Componentes: Los sistemas de software basados en componentes se basan en principios definidos por una ingeniería de software específica. Los componentes son las unidades de modelado, diseño e implementación,

las interfaces están separadas de las implementaciones, y conjuntamente con sus interacciones son el centro de incumbencias en el diseño arquitectónico. Los componentes soportan algún régimen de introspección, de modo que su funcionalidad y propiedades puedan ser descubiertas y utilizadas en tiempo de ejecución.

1.3.4 Estilo de Código Móvil: Esta familia de estilos enfatiza la portabilidad. Ejemplos de la misma son los intérpretes, los sistemas basados en reglas y los procesadores de lenguaje de comando.

- Arquitectura de Máquinas Virtuales: Esta arquitectura se conoce como intérpretes basados en tablas ó sistemas basados en reglas. Estos sistemas se representan mediante un seudo-programa a interpretar y una máquina de interpretación. Estas variedades incluyen un extenso espectro que está comprendido desde los llamados lenguajes de alto nivel hasta los paradigmas declarativos no secuenciales de programación.

1.3.5 Estilos Peer To Peer: Esta familia se conoce también como componentes independientes, enfatiza la modificabilidad por medio de la separación de las diversas partes que intervienen en la computación. Consiste por lo general en procesos independientes o entidades que se comunican a través de mensajes.

- Arquitecturas Basadas en Eventos: Estas se han llamado también arquitectura de invocación implícita, estas se vinculan con sistemas basados publicación-suscripción. La idea dominante en la invocación implícita es que, en lugar de invocar un procedimiento en forma directa un componente puede anunciar mediante difusión uno o más eventos.
- Arquitecturas Orientadas a Servicios (SOA en inglés): Esta construye toda la topología de la aplicación como una topología de interfaces, implementaciones y llamados a interfaces; es una relación entre servicios y consumidores de servicios, ambos lo suficientemente amplios como para representar una función de negocio completa.
- Arquitecturas Basadas en Recursos: Esta define recursos identificables y métodos para acceder y manipular el estado de esos recursos. El caso de referencia es nada menos que la World Wide Web, donde los URIs identifican los recursos y HTTP es el protocolo de acceso. El argumento central es que HTTP mismo, con su conjunto mínimo de

métodos y su semántica simplísima, es suficientemente general para modelar cualquier dominio de aplicación.

1.4 ¿Qué relación existe entre Estilos y Patrones?

Una vez que se ha decidido que estilos se van a usar en la aplicación los patrones que tienen relación con estos estilos ayudaran a derivar de esa formulación arquitectónica el diseño y ulteriormente la programación correspondiente.

¿Qué son los Patrones?

Los patrones fueron sugeridos por Christopher Alexander en 1977, el cual escribió una serie de libros y artículos referentes a ¿que son los patrones?, los textos más frecuentemente encontrados se refieren a patrones arquitectónicos en cuanto a arquitectura real, o sea de edificios y ciudades y no de arquitectura de aplicaciones de software.

Una de las ideas brillantes de la década de los 90 fue vincular las estructuras recurrentes, aquellas entidades que ocurrían una y otra vez; las soluciones recurrentes a problemas en determinados contextos con esta idea de Alexander que estaba referida como se dijo anteriormente a la arquitectura real.

- Un patrón es la solución a un problema en un contexto.
- Un patrón codifica conocimiento específico acumulado por la experiencia en un dominio.
- Un sistema bien estructurado está lleno de patrones.
- “Cada patrón describe un problema que ocurre una y otra vez en nuestro ambiente, y luego describe el núcleo de la solución a este problema, de tal manera que puede usar esa solución un millón de veces más, sin hacer jamás la misma cosa dos veces”

Los patrones pueden dividirse o clasificarse en:

Patrones de Arquitectura**:** Relacionados a la interacción de objetos dentro o entre niveles arquitectónicos Problemas arquitectónicos, adaptabilidad a requerimientos cambiantes, performance, modularidad, acoplamiento.

Patrones de Diseño: que fueron construidos dado los problemas con la claridad de diseño, multiplicación de clases, adaptabilidad a requerimientos cambiantes, proponiendo como solución: Comportamiento de factoría, Clase-Responsabilidad-Contrato (CRC).

Patrones de Análisis: Usualmente específicos de aplicación.

Patrones de Proceso o de Organización: que tratan todo lo concerniente con el desarrollo o procesos de administración de proyectos, o técnicas, o estructuras de organización.

Patrones de Idioma: que reglan la nomenclatura en la cual se escriben, se diseñan y desarrollan nuestros sistemas.

Los elementos de un patrón son:

1. Nombre

- Define un vocabulario de diseño.
- Facilita la abstracción.

2. Problema

- Describe cuando aplicar el patrón.
- Conjunto de fuerzas: objetivas y restricciones.
- Prerrequisitos.

3. Solución

- Elementos que constituyen el diseño (plantilla).
- Forma canónica para resolver fuerzas.

4. Consecuencias

- Resultados.
- Extensiones.
- Consensos.

Al contrario de los estilos arquitectónicos los patrones son muchos y a su vez muy variados y es casi imposible revisar todos los patrones que existen a la hora de hacer una determinada aplicación, por eso se recomienda el uso de los patrones que estén asociados en cada uno de los estilos que se seleccionen para el desarrollo de la arquitectura.

1.5 Lenguajes de Descripción Arquitectónica (ADLs en inglés)

En la década de 1990 y en lo que va del siglo XXI, se han materializado diversas propuestas para describir y razonar en términos de arquitectura de software, muchas de ellas han asumido la forma de lenguajes de descripción de arquitectónicas, o ADLs, estos ocupan una parte importante del trabajo arquitectónico desde la fundación de la disciplina. Se trata de un conjunto de propuestas de variado nivel de rigurosidad, casi todas ellas de extracción académica, que fueron surgiendo desde comienzos de la década de 1990 hasta la actualidad con el proyecto de unificación de los lenguajes de modelado bajo la forma de UML. Los ADLs difiere sustancialmente de UML, que al menos en su versión 1.x se estima inadecuado en su capacidad para expresar conectores en particular y en su modelo semántico en general para las clases de descripción y análisis que se requieren. Los ADLs permiten modelar una arquitectura mucho antes que se lleve a cabo la programación de las aplicaciones que la componen, analizar su adecuación, determinar sus puntos críticos y eventualmente simular su comportamiento.

Entre las características a considerar de estos lenguajes se puede citar las siguientes:

- **Composición:** Permiten la representación del sistema como composición de una serie de partes.
- **Configuración:** La descripción de la arquitectura es independiente de la de los componentes que formen parte del sistema.
- **Abstracción:** Describen los roles o papeles abstractos que juegan los componentes dentro de la arquitectura.
- **Flexibilidad:** Permiten la definición de nuevas formas de interacción entre componentes.
- **Reutilización:** Permiten la reutilización tanto de los componentes como de la propia arquitectura.
- **Heterogeneidad:** Permiten combinar descripciones heterogéneas.
- **Análisis:** Permiten diversas formas de análisis de la arquitectura y de los sistemas desarrollados a partir de ella.

Entre los principales ADLs mas usados en la actualidad están:

- ACME
- Armani
- Jacal
- CHAM

1.6 Lenguaje Unificado de Modelado (UML en inglés)

UML no es un lenguaje de programación, es un lenguaje para visualizar, especificar, construir y documentar los artefactos de un sistema que involucra una gran cantidad de software. Un artefacto es una información que es utilizada o producida mediante un proceso de desarrollo de software.

UML se quiere convertir en un lenguaje estándar con el que sea posible modelar todos los componentes del proceso de desarrollo de aplicaciones. Sin embargo, hay que tener en cuenta un aspecto importante del modelo: no pretende definir un modelo estándar de desarrollo, sino pretende unificar la experiencia pasada sobre técnicas de modelado e incorporar las mejores prácticas actuales en un acercamiento estándar.

UML capta la información sobre la estructura estática y el comportamiento dinámico de un sistema. Un sistema se modela como una colección de objetos discretos que interactúan para realizar un trabajo que finalmente beneficia a un usuario externo. Las herramientas pueden ofrecer generadores de código de UML para una gran variedad de lenguaje de programación, así como construir modelos por ingeniería inversa a partir de programas existentes.

1.7 Proceso Unificado de Desarrollo y Arquitectura de Software

RUP es el resultado de varios años de desarrollo y uso práctico en el que se han unificado técnicas de desarrollo, a través del UML, y trabajo de muchas metodologías utilizadas por los clientes. Fue creado por Jacobson, Rumbaugh y Booch, este utiliza el UML como lenguaje de representación visual, es Orientado a Objetos, unifica los mejores elementos de metodologías anteriores y está preparado para desarrollar grandes y complejos proyectos.

El Proceso Unificado es un proceso de desarrollo de software y está definido por un conjunto de actividades necesarias para transformar los requisitos del usuario en un sistema software.

Según el Proceso Unificado de Desarrollo de Software la arquitectura involucra los elementos más significativos del sistema y está influenciada entre otros por plataformas software,

sistemas operativos, manejadores de bases de datos, protocolos, consideraciones de desarrollo como sistemas heredados y requerimientos no funcionales. Se representa mediante varias vistas que se centran en aspectos concretos. [14]

El conjunto de todas estas vistas forman el llamado modelo 4+1 de la arquitectura, y lo forman las vistas lógica, de implementación, proceso y despliegue, más la de casos de uso que es la que da cohesión a todas.

Las principales **características de RUP** son:

- **Guiado por Casos de Uso**
- **Centrado en la Arquitectura**
- **Iterativo e Incremental**

Que esté **guiado por casos de uso** significa que los casos de uso reflejan lo que los usuarios futuros necesitan y desean, lo cual se capta cuando se modela el negocio y se representa a través de los requerimientos. A partir de aquí los casos de uso guían el proceso de desarrollo ya que los modelos que se obtienen, como resultado de los diferentes flujos de trabajo, representan la realización de los casos de uso, esto permite una mejor organización del sistema en cuanto a módulos. Otra característica fundamental es que este **centrado en la arquitectura,** lo que significa que la arquitectura muestra la visión común del sistema completo en la que el equipo de proyecto y los usuarios deben estar de acuerdo, por lo que describe los elementos del modelo que son más importantes para su construcción, los cimientos del sistema que son necesarios como base para comprenderlo, desarrollarlo y producirlo económicamente. La otra característica es que sea **Iterativo e Incremental;** aquí se divide el producto en mini proyectos, donde cada mini proyecto es una iteración que resulta en un incremento. Una iteración es una secuencia de actividades con un plan establecido y un criterio de evaluación, las iteraciones hacen referencia a pasos en el flujo de trabajo, y los incrementos al crecimiento del producto. La selección de lo que se implementará en una iteración se basa en casos de uso de mayor utilidad y los riesgos más importantes.

1.8 Servicios Web (Web Services en inglés)

Un servicio web es una colección de protocolos y estándares que sirven para intercambiar datos entre aplicaciones. Distintas aplicaciones de software desarrolladas en lenguajes de programación diferentes, y ejecutadas sobre cualquier plataforma. Estas aplicaciones o tecnologías intercambian datos entre sí, utilizando estándares de comunicación, con el

objetivo de ofrecer unos servicios. Los proveedores ofrecen sus servicios como procedimientos remotos y los usuarios solicitan un servicio llamando a estos procedimientos a través de la Web.

Describe una forma estandarizada de aplicaciones usando XML, SOAP, WSDL y UDDI, sobre el protocolo Internet. XML es usado para etiquetar los datos, SOAP es el encargado de la transferencia de la información, WSDL describe los servicios disponibles y UDDI es el responsable de que los servicios WEB disponibles sean listados. Los servicios WEB permiten a las organizaciones el intercambio de datos, incluso sin tener presente los sistemas que se encuentran detrás del CortaFuegos (Firewall en inglés).

1.9 Navegadores Web

Estos juegan un papel importante dentro de las aplicaciones web ya que desempeñan el papel de clientes.

La función principal del navegador es descargar documentos HTML y mostrarlos en pantalla. En la actualidad, no solamente descargan este tipo de documentos sino que muestran con el documento sus imágenes, sonidos e incluso vídeos en diferentes formatos y protocolos. Además, permiten almacenar la información en el disco o crear favoritos de las páginas más visitadas.

Existen hoy en día un sin número de navegadores, donde por mencionar algunos ejemplos podemos destacar a: NetScape Navigator, Internet Explorer, Firefox, Safari, Opera, Netscape y AOL; siendo los dos más populares: Internet Explorer y Firefox.

1.10 Lenguaje de Marcado Extensible (XML)

XML, cuya sigla en inglés de e**X**tensible **M**arkup **L**anguage (lenguaje de marcas extensible), permite definir la gramática de lenguajes específicos, esto facilita declaraciones más precisas de contenido y resultados de búsquedas con más significado entre muchas plataformas.

La tecnología XML busca dar solución al problema de expresar información estructurada de la manera más abstracta y reutilizable posible. Que la información sea estructurada quiere decir que se compone de partes bien definidas, y que esas partes se componen a su vez de otras partes. Entonces se tiene un árbol de pedazos de información, estas partes se llaman elementos, y se señalan mediante etiquetas.

Una etiqueta consiste en una marca hecha en el documento, que señala una porción de este como un elemento, un pedazo de información con un sentido claro y definido.

Para que un objeto de texto sea considerado documento XML debe estar “bien formado” de acuerdo con la especificación XML.

- Los documentos han de seguir una estructura estrictamente jerárquica con lo que respecta a las etiquetas que delimitan sus elementos. Una etiqueta debe estar correctamente anidada. Los elementos con contenidos deben estar correctamente cerrados.
- Los documentos XML sólo permiten un elemento raíz del que todos los demás sean parte, es decir, sólo puede tener un elemento inicial.
- Los valores atributos en XML siempre deben estar encerrados entre comillas simples o dobles.
- El XML es sensible a mayúsculas y minúsculas. Existe un conjunto de caracteres llamados espacios en blanco (espacios, tabuladores, retornos de carro, saltos de línea) que los procesadores XML tratan de forma diferente en el marcado XML.
- Es necesario asignar nombres a las estructuras, tipos de elementos, entidades, elementos particulares, etc. En XML los nombres tienen alguna característica en común.
- Las construcciones como etiquetas, referencias de entidad y declaraciones se denominan marcas; son partes del documento que el procesador XML espera entender. El resto del documento entre marcas son los datos entendibles por las personas.

1.11 Protocolo de Acceso Simple a Objetos (SOAP)

SOAP (siglas de Simple Object Access Protocol) es un protocolo que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML, SOAP es uno de los protocolos utilizados en los servicios web.

Este protocolo usa el código fuente en XML. Esto es una ventaja ya que facilita su lectura por parte de cualquier persona, pero también es un inconveniente dado que los mensajes resultantes son más largos. El intercambio de mensajes se realiza mediante tecnología de componentes. El término Object en el nombre significa que se adhiere al paradigma de la programación orientada a objetos.

1.12 Lenguaje de Descripción de Servicios Web (WSDL)

Lenguaje basado en XML para describir servicios web. Permite describir la interfaz pública de los mismos; eso significa que detalla los protocolos y los formatos de los mensajes necesarios para interactuar con los servicios listados en su catálogo. Las operaciones y mensajes que soporta se describen en abstracto y se ligan después al protocolo concreto de red y al formato del mensaje.

Las principales ventajas de WSDL son:

- Facilita escribir y mantener servicios mediante una aproximación estructurada para definir interfaces web
- Facilita el acceso a esos servicios web reduciendo el código que hay que escribir para hacer un cliente
- Facilita hacer cambios para ampliar los servicios, reduciendo la posibilidad de que los clientes dejen de funcionar al llamar a esos servicios

1.13 Subversion

Es un sistema de control de versiones libre y de código fuente abierto. Es decir, maneja ficheros y directorios a través del tiempo. Existe un árbol de ficheros en un *repositorio* central. El repositorio es como un servidor de ficheros ordinario, excepto porque recuerda todos los cambios hechos a sus ficheros y directorios. Esto le permite recuperar versiones antiguas de sus datos, o examinar el historial de cambios de los mismos. Subversion puede acceder al repositorio a través de redes, lo que le permite ser usado por personas que se encuentran en distintos ordenadores.

Brinda la capacidad para que varias personas puedan modificar y administrar el mismo conjunto de datos desde sus respectivas. Subversion es un sistema general que puede ser usado para administrar cualquier conjunto de ficheros. Existen varias interfaces a Subversion, ya sea programas individuales como interfaces que lo integran en entornos de desarrollo, donde se encuentra TortoiseSVN, el cual provee integración con el explorador de Windows, siendo este la interfaz más popular en este sistema operativo.

1.14 TortoiseSVN

Es un cliente gratuito de código abierto para el sistema de control de versiones Subversion. Maneja ficheros y directorios a lo largo del tiempo. Los ficheros se almacenan en un repositorio central. El repositorio es prácticamente lo mismo que un servidor de ficheros ordinario, salvo que recuerda todos los cambios que se hayan hecho a sus ficheros y directorios. Esto permite que pueda recuperar versiones antiguas de sus ficheros y examinar la historia de cuándo y cómo cambiaron sus datos, y quién hizo el cambio. Es fácil de usar, permite ver el estado de los archivos desde en el explorador de Windows y permite también crear gráficos de todas las revisiones/asignadas.

1.15 .NET Framework

Es neutral en cuanto al lenguaje de programación y funciona en base a librerías. Se puede programar utilizando Visual Basic.NET, C++.NET, C#, J# y otros más. Esta neutralidad en el lenguaje de programación es posible gracias a la arquitectura del .NET Framework.

Es un componente integral de Windows que admite la creación y la ejecución de la siguiente generación de aplicaciones y servicios Web XML. El diseño de .NET Framework está enfocado a cumplir los objetivos siguientes:

- Proporcionar un entorno coherente de programación orientada a objetos, en el que el código de los objetos se pueda almacenar y ejecutar de forma local, ejecutar de forma local pero distribuida en Internet o ejecutar de forma remota.
- Proporcionar un entorno de ejecución de código que reduzca lo máximo posible la implementación de software y los conflictos de versiones.
- Ofrecer un entorno de ejecución de código que fomente la ejecución segura del mismo, incluso del creado por terceras personas desconocidas o que no son de plena confianza.
- Proporcionar un entorno de ejecución de código que elimine los problemas de rendimiento de los entornos en los que se utilizan secuencias de comandos o intérpretes de comandos.
- Ofrecer al programador una experiencia coherente entre tipos de aplicaciones muy diferentes, como las basadas en Windows o en el Web.
- Basar toda la comunicación en estándares del sector para asegurar que el código de .NET Framework se puede integrar con otros tipos de código.

.NET Framework contiene dos componentes principales: Common Language Runtime y la biblioteca de clases de .NET Framework. Common Language Runtime es el fundamento de la tecnología. El motor de tiempo de ejecución se puede considerar como un agente que administra el código en tiempo de ejecución y proporciona servicios centrales, como la administración de memoria, la administración de subprocesos y la interacción remota, al tiempo que aplica una seguridad estricta a los tipos y otras formas de especificación del código que fomentan su seguridad y solidez. El código destinado al motor de tiempo de ejecución se denomina código administrado, a diferencia del resto de código, que se conoce como código no administrado. La biblioteca de clases es una completa colección orientada a objetos de tipos reutilizables que se pueden emplear para desarrollar aplicaciones que abarcan desde las tradicionales herramientas de interfaz gráfica de usuario (GUI en inglés) o de línea de comandos.

1.16 Visual Studio 2005

Visual Studio .NET es un entorno de desarrollo integrado (IDE en inglés) desarrollado por Microsoft a partir de 2002, puede utilizarse para construir aplicaciones dirigidas a Windows (utilizando Windows Forms), Web (usando ASP.NET y Servicios Web) y dispositivos portátiles.

Esta es la herramienta de desarrollo multilenguaje más completa para construir e integrar rápidamente aplicaciones y servicios Web XML. Aumenta de un modo extraordinario la productividad de los desarrolladores y crea nuevas oportunidades de negocio. En su diseño se han integrado a fondo los estándares y protocolos de Internet, como XML y SOAP, por lo que Visual Studio .NET simplifica considerablemente el ciclo de vida del desarrollo de aplicaciones.

Visual Studio presenta un nuevo diseñador de páginas Web que incluye muchas mejoras para la creación y edición de páginas Web de ASP.NET y páginas HTML. Proporciona una forma más fácil y rápida de crear páginas de formularios Web Forms que en Visual Studio .NET 2003. Ayuda a los programadores a sentirse cómodos durante el diseño de soluciones complejas basadas en la Web. Utilizando código de validación de cliente generado automáticamente, los programadores Web pueden reducir la cantidad de código JavaScript de cliente y garantizar que la aplicación funcione tanto en Microsoft Internet Explorer como en Netscape.

1.16 ASP.NET

ASP.NET es un conjunto de tecnologías de desarrollo de aplicaciones web comercializado por Microsoft. Es usado por programadores para construir sitios web domésticos, aplicaciones web

y servicios XML. Forma parte de la plataforma .NET de Microsoft y es la tecnología sucesora de la tecnología Active Server Pages (ASP).

¿Qué mejoras trae ASP.net? ¿Es realmente mejor que ASP?

Sin duda, es mucho mejor que el ASP tradicional, pues ASP.net trae diversas mejoras entre las cuales se destacan:

- Rendimiento: la aplicación en cada petición tiene una compilación Jit (Just In Time en inglés), es decir se compila desde el código nativo, lo que permite mucho mejor rendimiento. También permite el almacenamiento del caché en el servidor

- Rapidez en programación: mediante diversos controles, podemos con unas pocas líneas y en menos de 5 minutos mostrar toda una base de datos y hacer rutinas complejas.

- Servicios Web: La presencia de esta herramienta permite compartir datos e información entre distintos sitios.

- Escalabilidad y disponibilidad: ASP.NET se ha diseñado teniendo en cuenta la escalabilidad, con el fin de mejorar el rendimiento en entornos agrupados y de múltiples procesadores, además, el motor de tiempo de ejecución de ASP.NET controla y administra los procesos de cerca, por lo que si uno no se comporta adecuadamente se puede crear un proceso nuevo en su lugar, lo que ayuda a mantener la aplicación disponible constantemente para controlar solicitudes.
- Seguridad: Con la autenticación de Windows integrada y la configuración por aplicación, se puede tener la completa seguridad de que las aplicaciones están a salvo.

1.17 C#

C# es un lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma .NET, que después fue aprobado como un estándar por la European Computer Manufacturers Association (ECMA, siglas en inglés) y la Organización Internacional para la Estandarización (ISO, siglas en inglés).

Este lenguaje de programación combina los mejores elementos de múltiples lenguajes de amplia difusión como C++, Java, Visual Basic o Delphi. De hecho, su creador Anders Heljsberg fue también el creador de muchos otros lenguajes y entornos como Turbo Pascal, Delphi o Visual J++. La idea principal detrás del lenguaje es combinar la potencia de lenguajes como

C++ con la sencillez de lenguajes como Visual Basic, y que además la migración a este lenguaje por los programadores de C/C++/Java sea lo más inmediata posible.

Aunque es posible escribir código para la plataforma .NET en muchos otros lenguajes, C# es el único que ha sido diseñado específicamente para ser utilizado sobre ella, por lo que programar usando C# es mucho más sencillo e intuitivo que hacerlo con cualquiera de los otros lenguajes, disponibles en .NET, por eso se suele decir que es el lenguaje nativo de .NET.

A continuación se muestra de forma resumida las principales características que hacen que el lenguaje sea una buena elección:

Sencillo: Elimina muchos elementos de otros lenguajes que son innecesarios en .NET. Por ejemplo: el código escrito en C# es autocontenido. No necesita de ficheros adicionales tales como ficheros de cabecera. El tamaño de los tipos de datos básicos es fijo e independiente del compilador, sistema operativo o máquina para quienes se compile. No se incluyen macros, herencia múltiple o la necesidad de un operador diferente del punto (.) para acceder a miembros de espacios de nombres.

Moderno: Incorpora elementos útiles para el desarrollo de aplicaciones por ejemplo: un tipo básico decimal que permita realizar operaciones de alta precisión con reales de 128 bits, la instrucción “foreach” que permite recorrer colecciones con facilidad y es ampliable a tipos definidos por el usuario.

Orientado a objetos: Es más puro en tanto que no admite ni funciones, ni variables globales, sino que todo el código y datos han de definirse dentro de definiciones de tipos de datos, lo que reduce problemas por conflictos de nombres y facilita la legibilidad del código. Soporta encapsulación, herencia y polimorfismo.

Orientado a componentes: Permite definir cómodamente propiedades (similares a campos de acceso controlado), eventos (asociación controlada de funciones de respuesta a notificaciones) o atributos (información sobre un tipo o sus miembros).

Eficiente: Pueden marcarse regiones de código como inseguras y podrán usarse en ellas punteros, lo que puede resultar vital para situaciones donde se necesite una eficiencia y velocidad de procesamiento muy grandes.

Gestión automática de memoria: Tiene a su disposición el recolector de basura del CLR (Common Language Runtime). No es necesario incluir instrucciones de destrucción de objetos.

Seguridad de tipos: Incluye mecanismos que permiten asegurar que los accesos a tipos de datos siempre se realicen correctamente. Para ello se toman medidas tales como:

- Sólo se admiten conversiones entre tipos compatibles.
- No se pueden usar variables no inicializadas.
- Se comprueba que todo acceso a los elementos de una tabla se realice con índices que se encuentren dentro del rango de la misma.
- Se puede controlar la producción de desbordamientos en operaciones aritméticas, informándose de ello con una excepción cuando ocurra.
- Pueden definirse métodos que admitan un número indefinido de parámetros de un cierto tipo y siempre se comprueba que los valores que se les pasen en cada llamada sean de los tipos apropiados.

Instrucciones seguras: Se han impuesto una serie de restricciones en el uso de las instrucciones de control más comunes. Por ejemplo, Se evitarán errores por confusión del operador de igualdad (==) con el de asignación (=).

1.17 Mono

Es la implementación libre del CLI (Common Language Infraestructure) y C#. Esta implementación es de código fuente abierto (Open source). El Mono incluye el CLI, el cual contiene la máquina virtual que se encarga de cargar las clases, el compilador Jit (Just-in-time) y el colector de basura (garbage collector en inglés); todo esto escrito desde cero. Mono también incluye un compilador de C#, el cual paradójicamente está escrito en C# y al igual que el CLI.

¿Es Mono compatible con el .Net Framework?

Uno de los objetivos de Mono es alcanzar un alto grado de compatibilidad con.Net Framework. Con esto se busca que un binario compilado en Windows con el .Net Framework pueda ejecutarse en alguna de las plataformas de Mono sin tener que recompilar el binario, y que a su vez pueda hacer uso de las librerías compatibles de Mono, Ej: System.Data, System.Xml, etc. Las librerías proporcionadas por Mono son casi 100% compatibles con su contra parte del .Net FrameworkWindowsForms.

1.18 Herramientas CASE:

Rational Rose Enterprise Suite: Esta herramienta es una solución integrada y completa para todo el ciclo de vida del desarrollo de Software. Acelera el desarrollo mediante el modelado visual, la generación de código y las capacidades de ingeniería reversa.

La suite de Rational ofrece varios productos, destacándose los siguientes:

Rational Rose: Es una herramienta de modelación visual para el proceso de modelación del negocio, análisis de requerimientos y diseño de arquitectura de componentes. Proporciona mecanismos para realizar la denominada Ingeniería Inversa, es decir, a partir del código de un programa, se puede obtener información sobre su diseño.

Rational Requisite Pro: Mantiene a todo el equipo de desarrollo actualizado a través del proceso de desarrollo de aplicaciones haciendo que los requerimientos se puedan escribir, comunicar y cambiar fácilmente. Facilita que los miembros del equipo colaboren con los requerimientos del proyecto.

Rational ClearQuest: Es un sistema poderoso y altamente flexible para seguimiento de defectos y cambios que captura y maneja todo tipo de solicitud de cambio a lo largo del ciclo de vida del desarrollo.

Rational SoDA: Automatiza la producción de documentación para todo el proceso de desarrollo de software, reduciendo dramáticamente el tiempo y el costo de documentar el software. Brinda a sus usuarios una interfaz única para reportar requerimientos, diseño, test, y estado de defectos.

Rational ClearCase: Es una herramienta de administración de configuración de software, simplifica el proceso de cambio, ofrece una gestión fiable, ampliable y flexible de los activos de software para equipos de desarrollo de gran tamaño y tamaño medio.

1.19 Sql Server 2000

Microsoft SQL Server 2000, es un RDBMS (Relational DataBase Management System), que aporta todo lo necesario para facilitar la integración de sus datos en Internet. Ofrece herramientas de análisis y gestión de almacén de datos. Es un potente motor de bases de datos de alto rendimiento capaz de soportar millones de registros por tabla con un interface intuitivo y con herramientas de desarrollo integradas como .NET.

Microsoft SQL Server ha pasado de ser una plataforma de bases de datos del tipo cliente/servidor a ser una plataforma de tareas con todo el poder y funcionalidad que necesita para producir aplicaciones cliente/servidor y aplicaciones para Web que sorprenden a los usuarios. Con nuevos tipos de bases de datos, funciones definidas por el usuario, asistencia a través de Internet y otras nuevas características.

1.20 Controlador de Versiones Visual Source Safe

El sistema de control de versiones de Microsoft Visual Source Safe 6.0 es la última edición del premiado sistema de control de versiones de Microsoft para administrar el software y el desarrollo de sitios Web. Está completamente integrado con los entornos de desarrollo de .NET, Visual Basic, Visual C++, Visual J++, Visual InterDev y Visual FoxPro, así como con las aplicaciones de Microsoft Office, y proporciona un control de versiones fácil de utilizar y orientado al proyecto. Visual SourceSafe funciona con cualquier tipo de archivo producido por cualquier lenguaje de desarrollo, herramienta de edición o aplicación. Los usuarios pueden trabajar tanto al nivel de archivo como de proyecto. Ofrece una administración sencilla del contenido Web y de HTML con los mapas de sitios, la comprobación de hipervínculos y la distribución directa en el Web. Reconcilia los cambios de archivo e impide que el código se sobrescriba de forma accidental usando el Bloqueo de un archivo desprotegido.

1.21 Conclusiones

La aplicación que se desea es del tipo cliente/servidor y debe prestar diferentes servicios orientados a mejorar la calidad de las funcionalidades que automatiza, por lo que se propone para la realización del proyecto TeleBanca crear una arquitectura utilizando una arquitectura de cuatro capas, Presentación, Lógica de Negocio, Acceso a Datos y Modelo de Datos, utilizando interfaces para la comunicación entre dichas capas, para que el desarrollo del proyecto sea paralelo e independiente. Veamos a continuación las características de esta arquitectura con un mayor nivel de detalles.

CAPÍTULO 2

Diseño de la Arquitectura del Sistema

2.1 Introducción

En este capítulo se hace un análisis sobre los aspectos más importante que debe desempeñar el rol de arquitecto de software, que son la definición de una Línea Base de la Arquitectura y el Documento Descripción de la Arquitectura.

Rol del arquitecto:

El Arquitecto de Software debe dominar la mayor cantidad de tecnologías de software y prácticas de diseño, para así poder tomar decisiones adecuadas para garantizar el mejor desempeño de las aplicaciones. El rol del arquitecto de software es crítico y sumamente importante, puesto que requiere de una gran variedad de conocimientos, tales como: ingeniería de requerimientos, teoría de arquitecturas de software, codificación, tecnologías de desarrollo, plataformas de hardware y software.

De igual manera, requiere de saber negociar intereses encontrados de múltiples involucrados en el desarrollo de un sistema de software; promover la colaboración entre el equipo; entender la relación entre atributos de calidad y estructuras; ser capaz de transmitir claramente la arquitectura a los equipos; escuchar, y entender múltiples puntos de vista. El arquitecto de software debe interaccionar con todos los involucrados en el desarrollo de un sistema de software, y ser capaz de dialogar con el analista para obtener los requerimientos significativos, diseñarlos y transmitirlos al programador para su codificación.

2.2 LINEA BASE DE LA ARQUITECTURA

2.2.1 Introducción

La Línea Base de la Arquitectura es un esqueleto del sistema con pocos músculos de software, pues no es más que la versión interna del sistema al final de la fase de elaboración, en fin tiene las versiones de todos los modelos que un sistema terminado contiene al final de la fase de construcción. Incluye el mismo esqueleto de subsistemas, componentes y nodos que un sistema definitivo, pero no existe toda la musculatura.

En la misma se exponen los estilos arquitectónicos de la aplicación, así como los principales elementos de la arquitectura. Se describen los principales patrones de arquitectura utilizados y las tecnologías y herramientas de software que se utilizarán en el sistema a desarrollar.

La Línea Base de la Arquitectura tiene un gran **Alcance** dentro del sistema a desarrollar, esta describe la estructura organizativa de la arquitectura según los estilos arquitectónicos que se utilizarán, además se propone la utilización de un conjunto de patrones que resuelven problemas que de una forma u otra se pudieran presentar a lo largo del desarrollo del sistema. También se hace una propuesta de las principales tecnologías y herramientas que soportan los estilos y patrones especificados y cumplen con las restricciones del sistema.

Los usuarios de la Línea Base son:

- Los arquitectos del proyecto, que le sirve de guía para la toma de decisiones arquitectónicas y son los encargados del mantenimiento y refinamiento de esta línea base.
- Los integrantes del equipo de desarrollo, quienes la usan como guía para la implementación del sistema.
- Los clientes tienen en ella una garantía de la calidad y el conocimiento sobre en qué tecnología está desarrollada su solución.

Alcance

La Línea Base de la Arquitectura describe la estructura del sistema en un alto nivel de abstracción. Describe detalladamente el organigrama de la arquitectura según los estilos arquitectónicos que se utilizarán.

Se propone la utilización de un conjunto de patrones que resuelven problemas que se podrían presentar a lo largo del desarrollo del sistema y las principales tecnologías y herramientas que soportan los estilos y patrones especificados y que además deben cumplir con las restricciones del sistema.

2.2.2 Metodología de desarrollo utilizada

El sistema se desarrollará haciendo uso de la metodología de desarrollo de software: Proceso Unificado de Desarrollo (Rational Unified Process (RUP)), donde se pone de manifiesto tres elementos importantes para el desarrollo de la arquitectura como parte de la solución.

- **Dirigido por casos de uso**
- **Centrado en la arquitectura**
- **Iterativo e Incremental**

2.2.3 Estructura organizativa de la arquitectura

Como se había visto en el capitulo anterior, la arquitectura de software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución.

Los estilos arquitectónicos son el nivel de abstracción mayor para estructurar el sistema, la elección del mismo está dada por el tipo de aplicación que se vaya a desarrollar.

A partir de los elementos mencionados anteriormente, la arquitectura del sistema se ha estructurado a partir del estilo Arquitectura en Capas.

2.2.4 Arquitectura en Capas

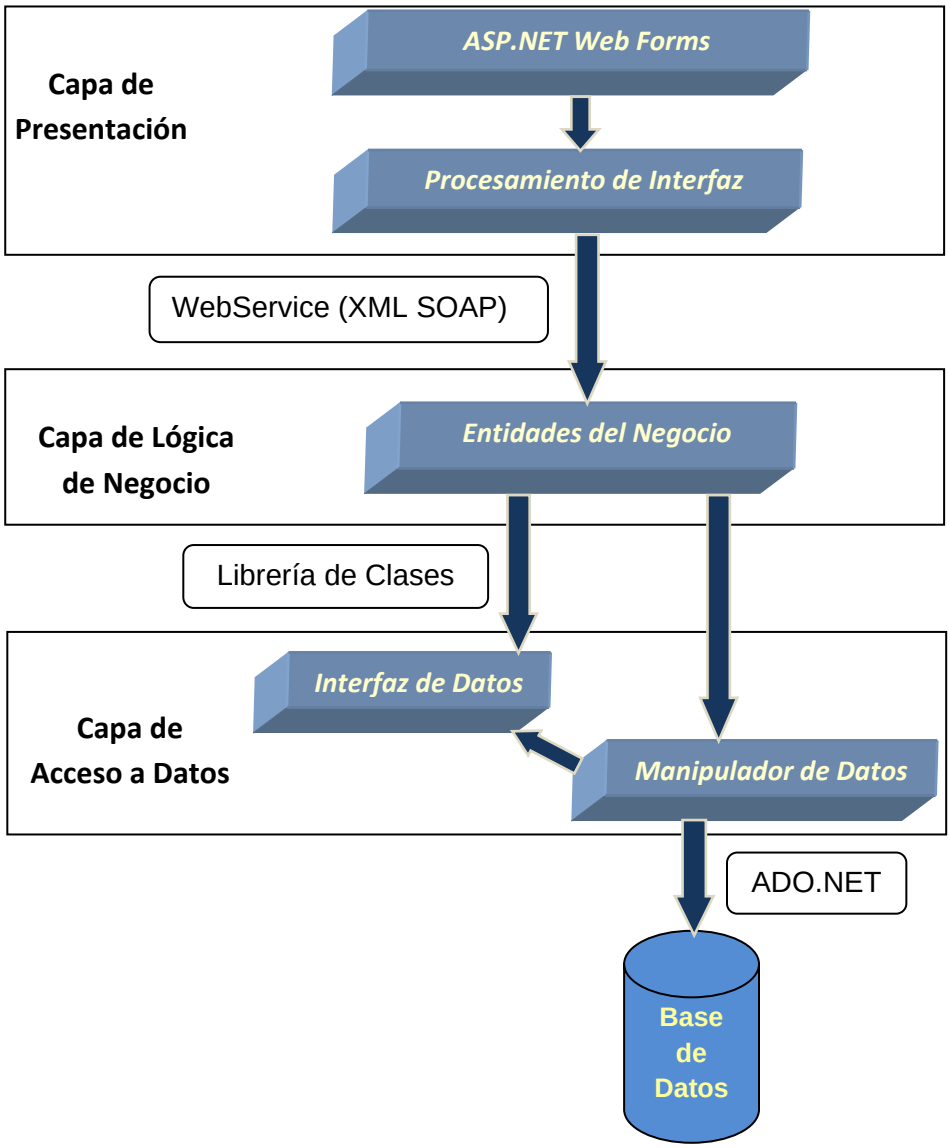

Fig1. Diagrama de capas del sistema.

Componentes o elementos

Los principales componentes que distinguen la arquitectura por 4 capas son:

- **Presentación**:

  En esta capa tenemos los ASP.NET Web Forms, que contienen las interfaces necesarias para que el usuario y el sistema intercambien toda la información necesaria

y está también Procesamiento de Interfaz que no es más que la definición de una estructura que soporten los procesos de interacción del sistema con el usuario de este, o sea aquí es donde se definen algunas clases que se encargarán de llevar a cabo funciones tales como mostrar los errores, control del sistema de navegación web en el servicio de pago, etc.

Fig2. Autenticación de TeleBanca.

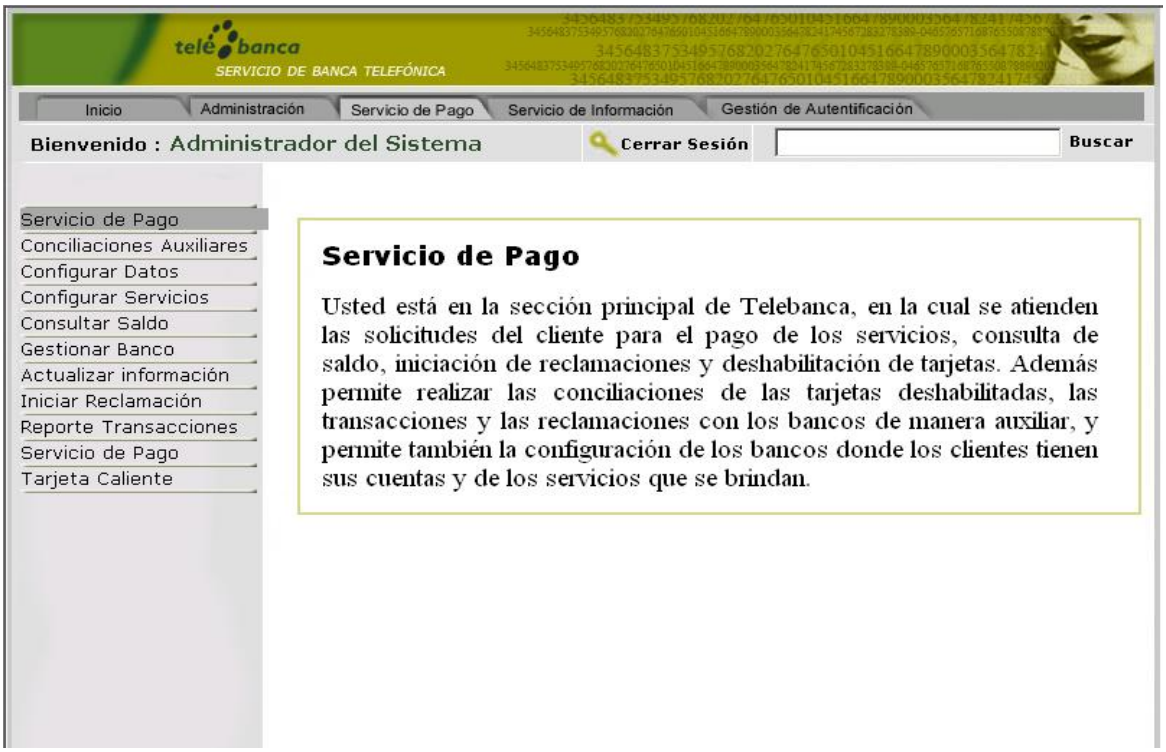

Fig3. Presentación de la aplicación TeleBanca.

- **Negocio:**

  Esta capa almacena todos los procesos del negocio, todas las transformaciones y procesos del sistema, que en nuestro caso sería el WebService.

- **Acceso a Datos**

  Esta capa es la encargada de manipular toda la información que respecta con el acceso a la base de datos, está formada por la Interfaz de Datos y por el Manipulador de Datos, este último es el encargado de interactuar con la BD para realizar todos los procesos de persistencia. Además interactúa con los WS de los bancos para garantizar la comunicación con ellos. Extrae también los datos de la BD y los convierte en objetos persistentes propios del sistema. La interfaz de datos no es más que las clases persistentes de las cuales sus objetos transitan por las demás capas de la aplicación (preferentemente la capa del negocio).
- **Base de Datos**

  Es capa contiene el modelo de datos de la aplicación. Está compuesta por tablas sus relaciones, que almacenan los datos requeridos por la aplicación.

Conectores / Configuraciones

Los conectores son las formas de comunicación entre los distintos componentes o elementos definidos en el estilo arquitectónico, no es más que la forma en que está representada la información que fluye entre estos.

- Presentación – Negocio

  La capa de presentación y la capa de lógica del negocio van a interactuar a través del WebService. En la capa de presentación se cargan las referencias web (WebReferences, en inglés) correspondientes al WS, y de esta forma se pueden crear objetos del tipo WS para poder tener visualización de todos los servicios que brinda esta capa.

  Por ejemplo:
- Negocio – Acceso a datos

  La capa de negocio, carga un fichero que no es más que la librería de datos correspondiente a la capa de acceso a datos, y es así como puede tener visibilidad sobre todos sus métodos, por ejemplo:
- Acceso a datos – Base de Datos

  Esta conexión se hace a través de ADO.NET

El estilo de capas Facilita la modularidad del sistema, la localización de errores y mejora considerablemente el soporte del mismo, cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior y las interacciones entre estas ocurren generalmente por invocación de métodos.

Sus principales **ventajas** son:

- Permite el desarrollo paralelo del sistema por cada capa.
- Facilita el mantenimiento y soporte del proyecto.
- Mayor flexibilidad (se pueden añadir nuevos módulos para dotar al sistema de nueva funcionalidad).
- Soporta un diseño basado en niveles de abstracción crecientes, lo cual a su vez permite a los implementadores la partición de un problema complejo en una secuencia de pasos incrementales.
- Proporciona amplia reutilización. Esto conduce a la posibilidad de definir interfaces de capa estándar, a partir de las cuales se pueden construir extensiones o prestaciones específicas.

Desventajas:

- Muchos problemas no admiten un buen mapeo en una estructura jerárquica. Incluso cuando un sistema se puede establecer lógicamente en capas, consideraciones de performance pueden requerir acoplamientos específicos entre capas de alto y bajo nivel.
- A veces es también extremadamente difícil encontrar el nivel de abstracción correcto; por ejemplo, la comunidad de comunicación ha encontrado complejo mapear los protocolos existentes en el framework ISO, de modo que muchos protocolos agrupan diversas capas, ocasionando que en el mercado proliferen los drivers o los servicios monolíticos.
- Además, los cambios en las capas de bajo nivel tienden a filtrarse hacia las de alto nivel, en especial si se utiliza una modalidad relajada; también se admite que la arquitectura en capas ayuda a controlar y encapsular aplicaciones complejas, pero complica no siempre razonablemente las aplicaciones simples.

2.3 Patrones Arquitectónicos

2.3.1 Model-View-Controller (MVC)

El MVC ha sido propio de las aplicaciones en Smalltalk por lo menos desde 1992, antes que se generalizaran las arquitecturas en capas múltiples. Un propósito común en numerosos sistemas es el de tomar datos de un almacenamiento y mostrarlos al usuario. Luego que el usuario introduce modificaciones, las mismas se reflejan en el almacenamiento. Dado que el flujo de información ocurre entre el almacenamiento y la interfaz, una tentación común, un impulso espontáneo (hoy se llamaría un anti-patrón) es unir ambas piezas para reducir la cantidad de código y optimizar la performance. Un problema es que las aplicaciones tienden a incorporar lógica de negocios que van más allá de la transmisión de datos.

El patrón conocido como Modelo-Vista-Controlador (MVC), ver fig.5, separa el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres clases diferentes:

- Modelo: El modelo administra el comportamiento y los datos del dominio de aplicación, responde a requerimientos de información sobre su estado (usualmente formulados desde la vista) y responde a instrucciones de cambiar el estado (habitualmente desde el controlador).

- Vista: Maneja la visualización de la información.

- Controlador: Interpreta las acciones del ratón y el teclado, informando al modelo y a la vista para que cambien según resulte apropiado.

2.4 Patrones de Diseño

2.4.1 Fábrica Abstracta (Abstract Factory, por sus siglas en inglés)

Este patrón de creación que proporciona una interfaz para crear familias de objetos relacionados o que dependen entre sí, sin especificar sus clases concretas.

Se puede usar Fábrica Abstracta cuando un sistema debe ser independiente de cómo se crean, componen y representan sus productos. Cuando una familia de objetos productos relacionados está diseñada para ser usada conjuntamente, y es necesario hacer cumplir esta restricción. O cuando se quiera proporcionar una biblioteca de clases de productos y solo quiere revelar sus interfaces, no sus implementaciones.

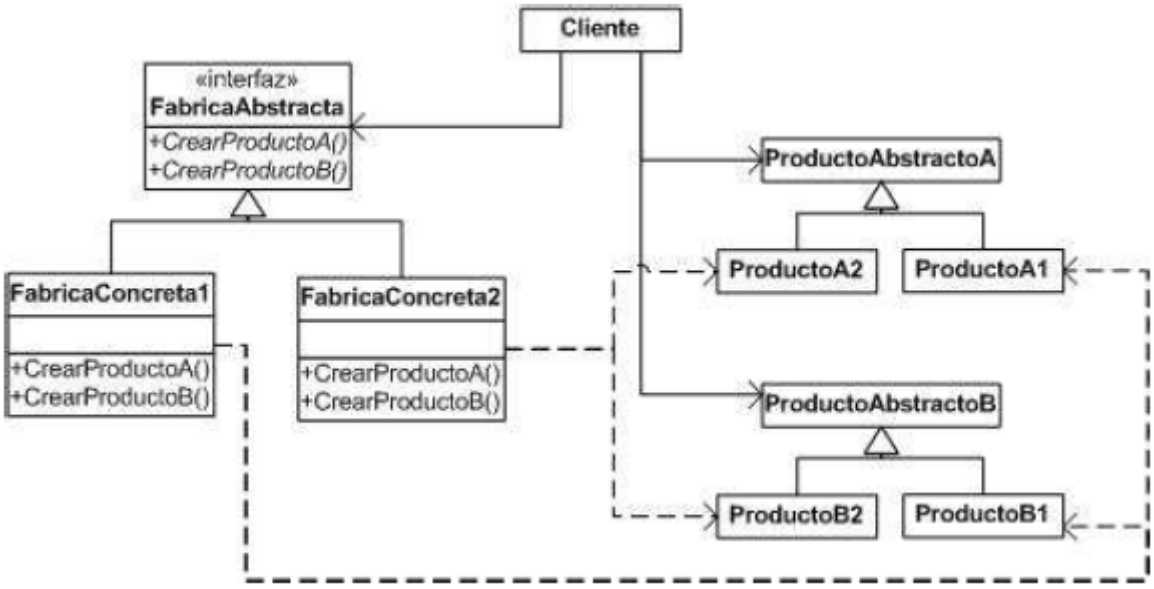

Fig.4 Estructura de Fabricación Abstracta

Dentro de su estructura encontramos diferentes clases participantes, las cuales citamos a continuación:

1) FabricaAbstracta: Declara una interfaz para operaciones que crean objetos productos abstractos.
2) FabricaConcreta: Implementa las operaciones para crear objetos producto concreto.
3) ProductoAbstracto: Declara una interfaz para un tipo de objeto producto.
4) ProductoConcreto: Define un objeto producto para que sea creado por la fábrica correspondiente. Implementa la interfaz ProductoAbstracto.
5) Cliente: Sólo usa interfaces declaradas por las clases FabricaAbstracta y ProductoAbstracto.
Normalmente solo se crea una única instancia de una clase FabricaConcreta en tiempo de ejecución, quien crea objetos producto que tienen una determinada implementación. Para crear diferentes objetos producto, los clientes deben usar una fábrica concreta diferente. FabricaAbstracta delega la creación de objetos producto en su subclase FabricaConcreta.

Ventajas:

1) Aísla las clases concretas: Ayuda a controlar las clases de objetos que crea una aplicación, encapsula la responsabilidad y el proceso de creación de objetos producto, aísla a los clientes de las clases de implementación, manipulan las instancias a través de sus

interfaces abstractas. Los nombres de las clases producto quedan aisladas en la implementación de la fábrica concreta; no aparecen en el código cliente.

2) Facilita el intercambio de familias de productos: La clase de una fábrica concreta solo aparece una vez en una aplicación, cuando se crea. Esto facilita cambiarla (como crea una familia completa de productos, cambia toda de una vez).

3) Promueve la consistencia entre productos: Se diseñan objetos producto en una familia para trabajar juntos.

Desventaja:

Es difícil dar cabida a nuevos tipos de productos: Ampliar las fábricas abstractas para producir nuevos tipos de productos no es fácil ya que la interfaz FabricaAbstracta fija el conjunto de productos que se pueden crear (implica cambiar la clase FabricaAbstracta y todas sus subclases).

2.4.2 Patrón Mediator

Este patrón de comportamiento define un objeto que encapsula cómo interactúan una serie de objetos. Promueve un bajo acoplamiento al evitar que los objetos se refieran unos a otros explícitamente y permite variar la interacción entre ellos de forma independiente.

Se puede utilizar esta patrón cuando un conjunto de objetos se comunican de forma bien definida, pero compleja. Las interdependencias resultantes no están estructuradas y son difíciles de comprender. Es difícil reutilizar un objeto, ya que este se refiere a otros muchos objetos con los que se comunica. Un comportamiento que está distribuido entre varias clases debería poder ser adaptado sin necesidad de una gran cantidad de subclases.

Estructura

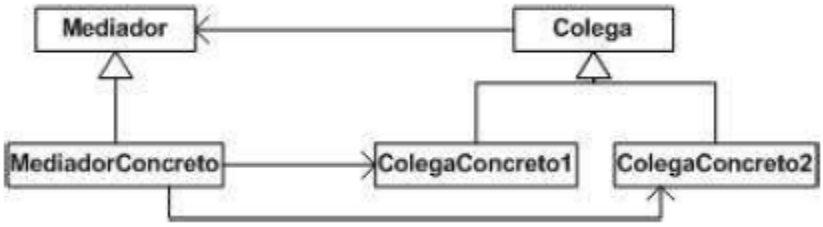

Fig.5 Estructura de objetos típica

Participantes

1) Mediador: Define una interfaz para comunicarse con sus objetos Colega.

2) MediadorConcreto: Implementa el comportamiento cooperativo coordinando objetos Colega. Conoce a sus colegas.
3) Clases Colega: Cada clase conoce a su objeto Mediador y se comunica a este cada vez que, de no existir el Mediador, se hubiera comunicado con otro Colega.

Ventaja:
1) Reduce la herencia: Un Mediador localiza el comportamiento que de otra manera estaría distribuido en varios objetos. Para cambiarlo solo es necesario crear una subclase del Mediador; las clases Colega pueden ser reutilizadas tal cual.
2) Desacopla a los Colegas: Un Mediador promueve un bajo acoplamiento entre Colegas. Las clases Colega pueden usarse y modificarse de forma independiente.
3) Simplifica los protocolos de los objetos: Un Mediador sustituye interacciones muchos-a-muchos por interacciones uno-a-muchos entre el objeto mediador y sus Colegas, que son más fáciles de comprender, mantener y extender.
4) Abstrae como cooperan los objetos: Hacer de la mediación un concepto independiente y encapsularla en un objeto permite centrarse en cómo interactúan los objetos en vez de en su comportamiento individual, ayudando a clarificar cómo interactúan los objetos de un sistema.

Desventaja:
Centraliza el control: El patrón Mediator cambia complejidad de interacción por complejidad en el mediador, pudiendo hacerse más complejo que cualquier Colega individual, difícil de mantener.

2.5 Patrones de Idiomas

Estos se utilizan en los flujos de implementación, mantenimiento y despliegue, comúnmente reconocidos como estándares de codificación y proyecto, describen como codificar y representar operaciones comunes bien conocidas en un nuevo ambiente, o a través de un grupo, brindan legibilidad y predictibilidad.

2.5.1 Mayúsculas y Minúsculas Pascal

La primera letra del identificador y la primera letra de las siguientes palabras concatenadas están en mayúsculas. El estilo de mayúsculas y minúsculas Pascal se puede utilizar en identificadores de tres o más caracteres. Por ejemplo:

"**M**étodo**P**ascal"

2.5.2 Mayúsculas y Minúsculas Camel

La primera letra del identificador está en minúscula y la primera letra de las siguientes palabras concatenadas en mayúscula. Por ejemplo:

"**m**étodo**C**amel"

Para nuestra aplicación se propone el uso de una combinación de ambas notaciones, que son ampliamente recomendados por los usuarios de C#, teniendo de esta forma que se utilice el estilo de Mayúsculas y minúsculas Camel para los nombres de parámetros y el estilo Pascal para nombres de los métodos.

Ejemplo:

Type GetType(string typeName)

2.6 DOCUMENTO DE DESCRIPCION DE LA ARQUITECTURA

2.6.1 Introducción

El conjunto de modelos que describen la Línea Base de la Arquitectura se denomina ***Descripción de la Arquitectura***. El papel de la descripción de la arquitectura es guiar al equipo de desarrollo a través del ciclo de vida del sistema. Esta descripción puede adoptar diferentes formas, puede ser un extracto (un conjunto de vistas), o puede ser una reescritura de los extractos de forma que sea más fácil leerlos. Estas vistas incluyen elementos arquitectónicamente significativos (casos de uso, subsistemas, interfaces, algunas clases y componentes, nodos y colaboraciones).

La descripción de la arquitectura tiene cinco secciones, una para cada modelo: una vista del modelo de casos de uso, una vista del modelo de análisis (opcional), una vista del modelo de diseño, una vista del modelo de despliegue, y una vista del modelo de implementación.

Propósito

Este documento brinda información sobre la arquitectura del sistema, mediante las diferentes vistas arquitectónicas (vista de CU, despliegue, implementación, lógica). Este permite tomar decisiones arquitectónicamente significativas.

Lo que se persigue alcanzar con la elaboración de dicho documento es:

- Una mejor comprensión del sistema.
- La organización del desarrollo.
- Fomentar la reutilización.
- Hacer evolucionar el sistema.

Debe capacitar a los desarrolladores, directivos, clientes, otros usuarios para comprender con suficiente detalle y para facilitar su propia participación.

Alcance

Este documento influye en la toma de decisiones arquitectónicas con respecto al Sistema TeleBanca.

Representación arquitectónica

Este documento presenta la arquitectura como una serie de vistas:

1. *Vista de casos de uso*
2. *Vista lógica*
3. *Vista de despliegue*
4. *Vista de implementación*

2.2 Metas y restricciones arquitectónicas

Las metas y restricciones del sistema en cuanto a seguridad, portabilidad y reusabilidad, que son significativas para la arquitectura son:

2.2.1 Requerimientos de Hardware

Estaciones de trabajo (PC Cliente)

1. Periféricos: Mouse y Teclado
2. Tarjeta de Red.
3. 128 MB de RAM.
4. Puerto USB

PC impresora Pines

1. Tarjeta de Red.
2. 128 MB de RAM.
3. Puerto LPT1

PC impresora Tarjetas

1. Tarjeta de Red.
2. 128 MB de RAM.
3. Puerto USB

Servidor de Aplicaciones y Base de Datos

1. Periféricos: Mouse y Teclado.
2. 2 Tarjeta de Red.
3. 2 GB de RAM.
4. 120 GB de espacio en disco.

Servidor FTP

1. Periféricos: Mouse y Teclado.
2. Tarjeta de Red.
3. 512 MB de RAM.

4. 80 GB de espacio en disco.

Web Services

1. Periféricos: Mouse y Teclado.
2. Tarjeta de Red.
3. 1GB MB de RAM.
4. 80 GB de espacio en disco.

Impresora de Pines (Matriz de punto)

Impresora de Tarjetas (Láser)

Impresora de Reportes (Láser)

2.2.2 Requerimientos de Software

Estaciones de trabajo

1. Sistema Operativo: Windows XP o superior
2. Navegador Web: Internet Explorer v 6.0
3. Microsoft Office 2000 o superior.

Servidor de Aplicaciones y de Base de Datos

1. Sistema Operativo: Windows 2003 server
2. SQL Server 2000
3. Internet Information Server
4. .NET Framework v2.0
5. Crystal Report 10.0

Servidor FTP

1. Sistema Operativo: Windows 2000 server o superior
2. Internet Information Server

Web Services

1. Sistema Operativo: Windows 2003 server
2. .NET Framework v2.0
3. Internet Information Server

2.2.3 Redes

1. Tipo de cable a utilizar UTP categoría 5, con trasmisiones de datos de hasta 100 Mbps.
2. El Servidor de Aplicaciones tiene dos tarjetas de red porque existen 2 dominios, por lo que tiene una tarjeta para cada dominio.

3. La red existente en las instalaciones debe de soportar la transacción de paquetes de información de al menos unas 26 máquinas a la vez.
4. Debe estar protegida contra fallos de corriente y conectividad.
5. La transmisión se implementarán utilizando el protocolo TCP/IP que garantiza la trasmisión correcta de los datos, mediante el protocolo TPC.

2.2.4 Seguridad

1. Se garantizará un fuerte tratamiento de excepciones.
2. Un porciento de la seguridad corre por parte del Frameworks propuesto (.Net), servidores (Internet Information Server, File Transfer Protocol).
3. Se implementará un mecanismo de acceso a la base de datos, que está dado por la diferenciación de las acciones que el sistema realiza en cada momento. Es decir, un usuario para lectura cuando se requiera solo leer, uno para escritura cuando se requiera modificar los datos y otro con los privilegios administrativos, para la realización de copias de seguridad y otras acciones.
4. La encriptación de contraseñas.
5. La asignación de usuarios y sus funcionalidades sobre el sistema se definirán desde el modulo de Administración.
6. Se registrarán y auditarán las trazas dejadas por los usuarios al realizar las diferentes operaciones en el sistema. La programación de las auditorías será programada según corresponda.

2.2.5 Portabilidad, Escalabilidad, Reusabilidad

1. La aplicación se construirá utilizando estándares internaciones y patrones, para facilitar su integración futura, con componentes desarrollados por cualquier empresa y garantizar posibilidades de un mantenimiento ágil.
2. El sistema deberá poder ser accedido desde cualquier Sistema Operativo.
3. Se utilizara la tecnología cliente/servidor para la implementación del sistema.
4. La documentación de la arquitectura debe ser reutilizable para poder documentarla como una familia de productos.
5. El sistema deberá hacer un uso racional de los recursos de hardware, sobre todo en estaciones de trabajo clientes.

2.2.6 Restricciones de acuerdo a la estrategia de diseño

1. El diseño de la aplicación se hará aplicando Programación Orientada a Objetos.

2. Se utilizarán las tecnologías que brinda el framework .net sobre la Programación orientada a objetos en todas las capas.

2.2.7 Integración de los componentes y su comunicación

Todo el código dentro un mismo componente utiliza llamadas a métodos o eventos de forma directa. La comunicación entre diferentes componentes se realiza mediante llamadas a servicios web o de forma directa a nivel de negocio, en caso de utilizarse servicios web, la información que es transmitida debe cumplir con los estándares internacionales que hay establecidos para facilitar la integración entre nuevos componentes y otros sistemas bancarios.

2.2.8 Herramientas de desarrollo

1. IDE de desarrollo, Visual Studio 2005 que demanda para su funcionamiento recomendado 512 MB de RAM, integrado con los plugins siguientes:
    a. AnkhSVN 1.0 para el control de versiones.
2. Servidor de Bases de Datos: SQL Server 2000

2.3 Estructura del equipo de desarrollo

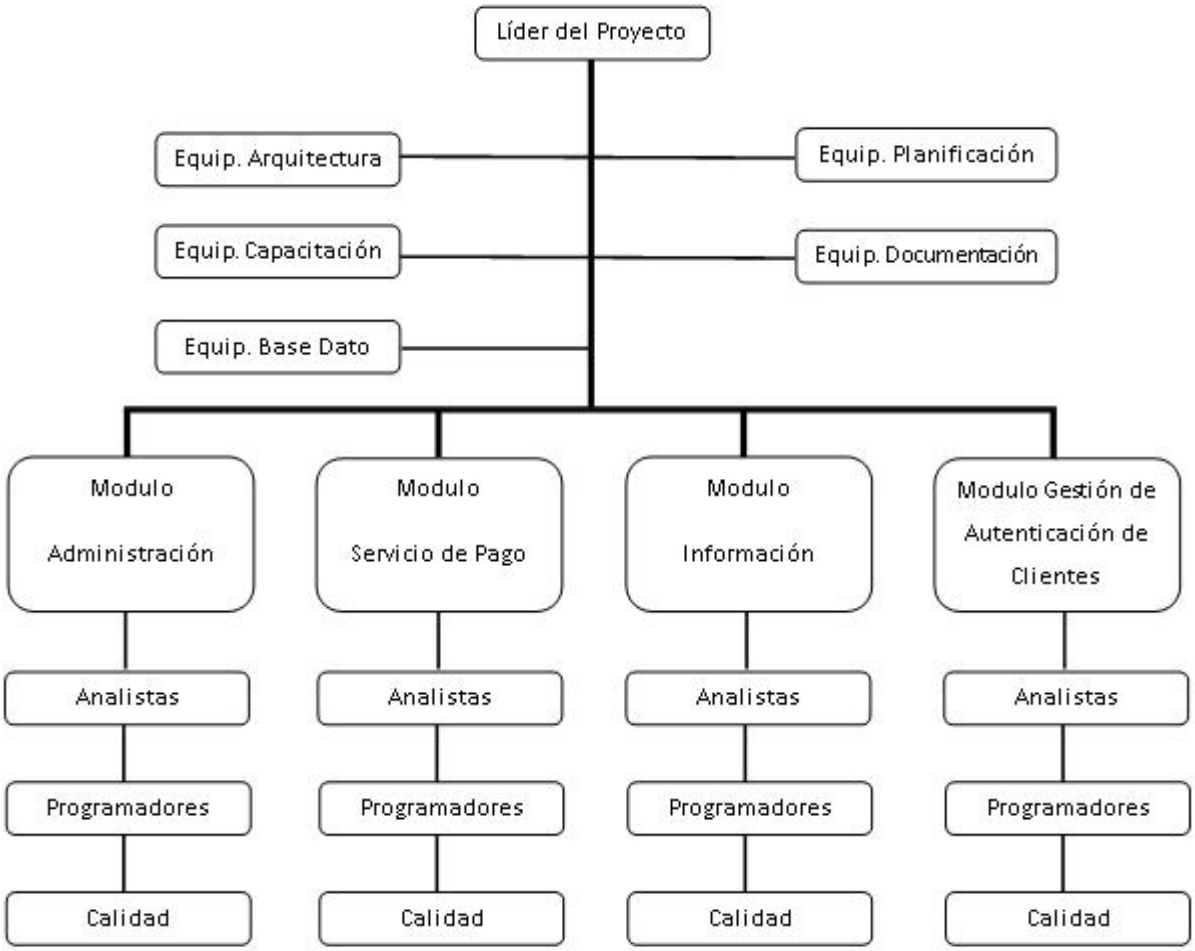

Fig.6 Estructura del equipo de trabajo.

2.4 Configuración de los puestos de trabajo por roles

- Analista
    - PC con mouse y teclado.
    - Rational Rose Suite 2003.
    - Paquete de Office, para manejar la documentación.
- Programador
    - PC con mouse y teclado, 1 GB de Memoria RAM.
    - Visual Studio 2005, para generar los códigos de la aplicación.
    - .NET FrameWork V.2.0
- Documentador
    - PC con mouse y teclado.
    - Rational Rose Suite 2003, para modelar casos de prueba.
    - Paquete de Office, para manejar la documentación.
- Diseñador BD.
    - PC con mouse y teclado, 1 GB de Memoria RAM.
    - SQL Server 2000, para modelar Bases de datos.
    - Visual Studio 2005, para administrar la base de datos.
- Calidad
    - PC con mouse y teclado
    - Visual Studio 2005, para revisar el código.
    - Paquete de Office, para crear los juegos de datos necesarios para las pruebas.

2.5 Vista de Casos de Uso

Esta vista nos brinda información de escenarios y casos de usos arquitectónicamente significativos con funcionalidades imprescindibles para el sistema. Los casos de uso arquitectónicamente significativos son aquellos que sirven para validar la arquitectura propuesta y describen las funcionalidades imprescindibles para el sistema.

2.5.1 Módulo Administración: Este el modulo se encarga del administración del sistema. Contiene la realización de casos de uso correspondientes Gestión de Usuario y Roles, Autenticación de Usuario, Acceso Ayuda, Actualizar Información, Configuración, entre otros.

Los casos de usos arquitectónicamente significativos correspondientes a este módulo son:

- CU_Autenticar_Usuario: El caso de uso comienza cuando el Usuario solicita entrar al Sistema, el mismo se identifica con su usuario y contraseña. Termina cuando el Usuario entra al Sistema o es denegado el acceso en caso de que no esté autorizado.

- CU_Gestionar_Usuario: El caso de uso comienza cuando el Administrador selecciona la opción gestionar usuarios. A partir de ahí el administrador tiene la opción de Crear, Modificar, o Eliminar un usuario. Finaliza el caso de uso cuando el Administrador termina la acción que ha seleccionado.

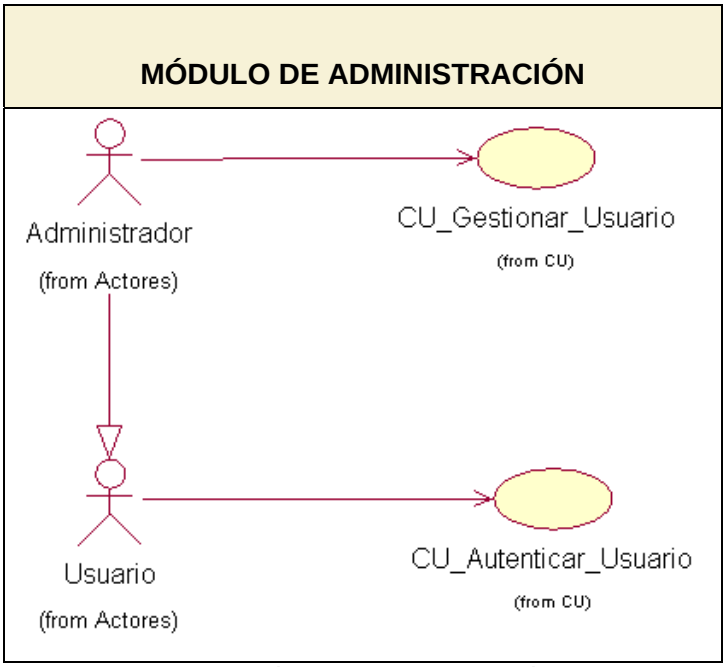

Fig.7 Módulo de Administración

2.5.2 Módulo Gestión de Autenticación de Clientes: Este modulo a grandes rasgos se encarga de crear las tarjetas para cada cliente, garantizando además la impresión de las mismas, necesarios para la identificación de los clientes asociados. También debe notificar automáticamente a los bancos asociados las autenticaciones creadas e impresas, y las solicitudes de bajas procesadas al final del día.

Los casos de usos arquitectónicamente significativos correspondientes a este módulo son:

- CU_Crear_Autenticación: El caso de uso tiene como objetivo crear las autenticaciones de los clientes de TeleBanca, o sea crear una tarjeta con matriz y pin para cada cliente.
- CU_Imprimir_Tarjetas: Este caso de uso es el que permite a la operadora de impresión de tarjetas imprimir las tarjetas que han sido creadas al generarse las autenticaciones.
- CU_Imprimir_Pines: Este caso de uso es el que permite a la operadora de impresión de pines imprimir los PIN que han sido creados al generarse las autenticaciones.

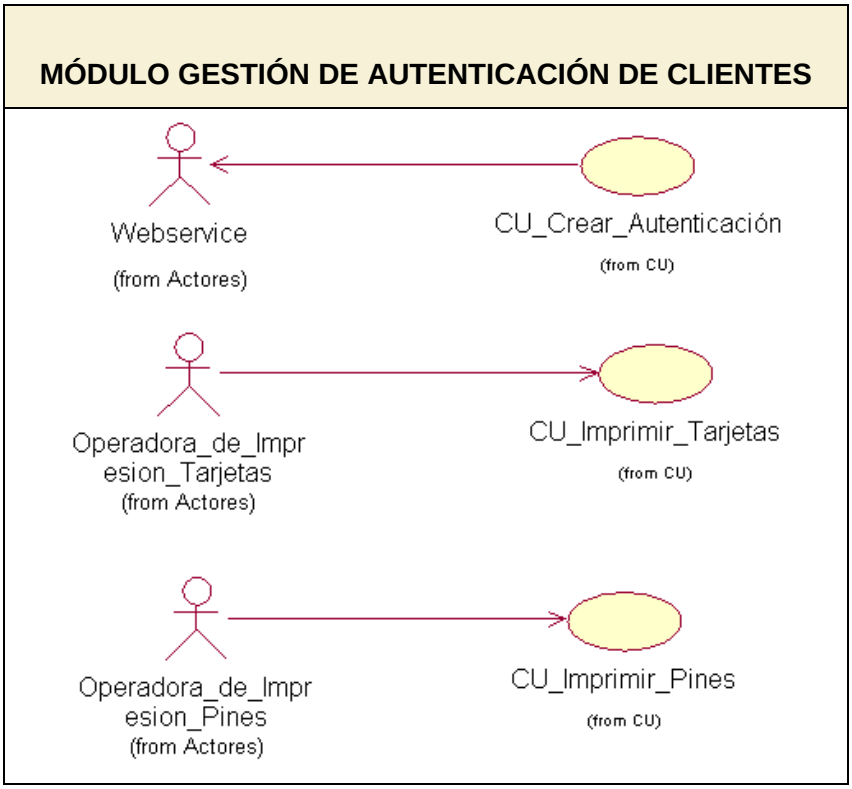

Fig.8 Módulo de Gestión de Autenticación de Clientes

2.5.3 Módulo Servicio de Pago: Este módulo controla todos los procesos referentes a la atención al cliente, además se encarga de la gestión de información de los servicios que se acogen al método de pago a través de la banca telefónica y de los bancos asociados a la misma.

Los casos de usos arquitectónicamente significativos correspondientes a este módulo son:

- CU_Pagar_Servicio: El caso de uso como el nombre lo indica es el encargado de llevar a cabo el pago de los servicios donde cada servicio tiene un nivel de autenticación definido, el pago puede ser simple o complejo en dependencia de la configuración y un cliente puede pagar uno o más servicios a varios asociados si los tiene contratados.
- CU Configurar Servicios: Este caso de uso permite insertar un nuevo servicio, modificar o eliminar un servicio existente, después de realizar cualquiera de estas operaciones se envía notificación a todos los Web Services de los bancos asociados a TeleBanca.
- CU Información del Servicio Complejo: Este caso de uso tiene como objetivo descargar el o los ficheros que entregan las empresas y que se ubica en el servidor ftp concebido para ello y realiza la actualización de la información referente al o los servicios complejos.
- Configurar_Datos: Este caso de uso inicia cuando el administrador de banca telefónica solicita la configuración de los datos. Permite insertar, modificar o eliminar datos. Autenticar_Cliente: El caso de uso se inicia cuando la operadora de servicio de pago desea acceder al pago de servicio, cada servicio tiene un nivel de autenticación definido, el pago puede ser simple o complejo en dependencia de la configuración. Un cliente puede pagar uno o más servicios a varios asociados si los tiene contratados.
- Verificar_Tarjeta: El caso de uso se inicia cuando el sistema verifica el estado en que se encuentra la tarjeta de un cliente dado dentro de la banca telefónica. Dentro de los estados están deshabilitada, pedida, creada o no válida.

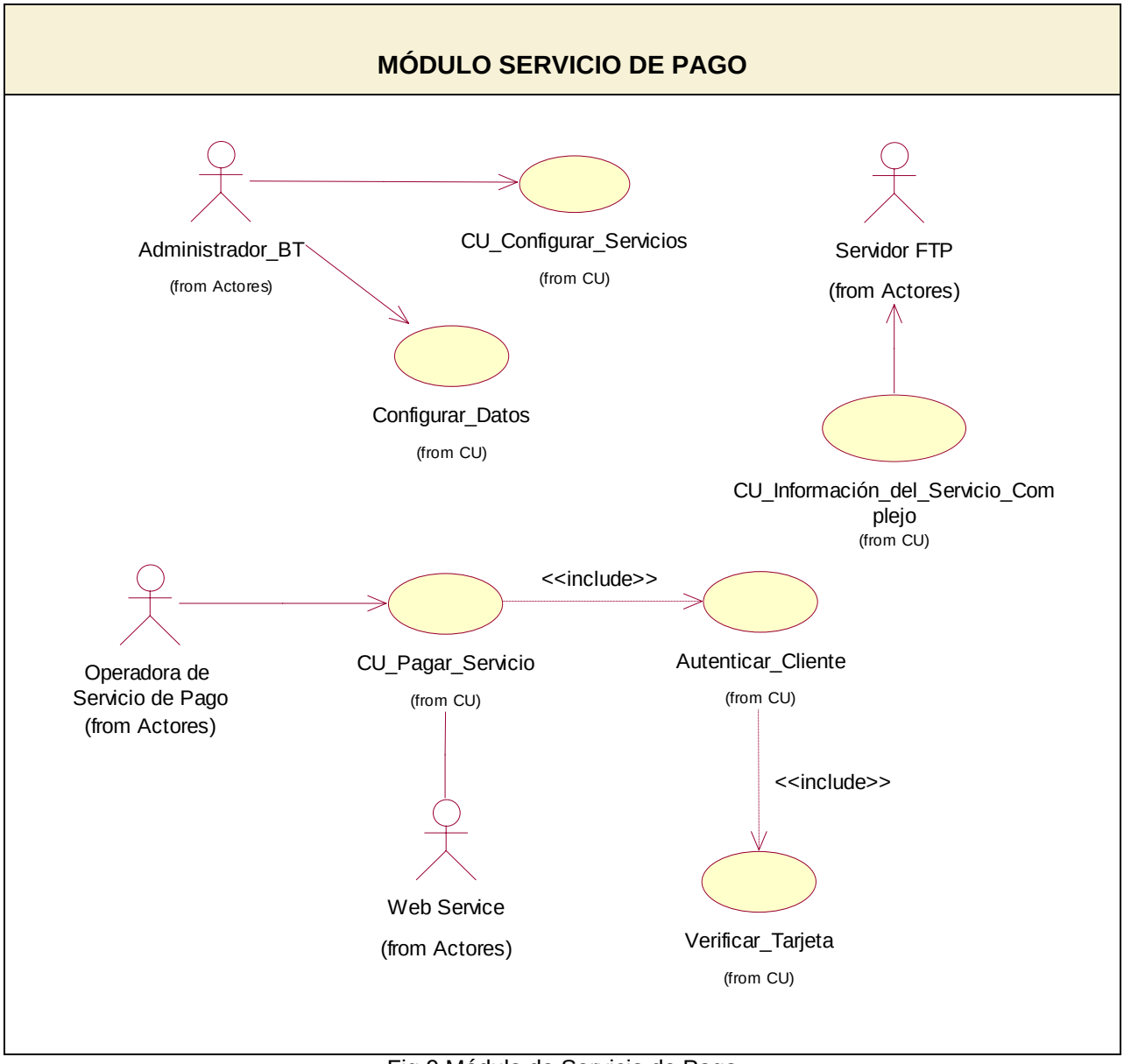

Fig.9 Módulo de Servicio de Pago

2.5.4 Módulo Información: Este modulo se encarga de brindar al cliente información de la de Agenda Electrónica y de los procesos solicitados a través de la operadora que lo atiende, lo cual será registrado por el sistema. Además se gestionará toda la información que se brinda y se harán reportes.

Los casos de usos arquitectónicamente significativos correspondientes a este módulo son:

- CU_Gestionar_Información: El objetivo de este caso de uso es permitirle al Administrador de Información acceder a la gestión de la información de la Agenda Electrónica o de los Procesos.
- CU_Gestión_de_Información_de_Procesos: El caso de uso tiene como objetivo permitirle al Administrador de Información gestionar la información referente a los procesos que se realizan en la Banca Telefónica.
- CU_Gestión_de_Agenda_Electrónica: El objetivo de este caso de uso es permitirle al Administrador de Información poder gestionar la información referente a la Agenda Electrónica.

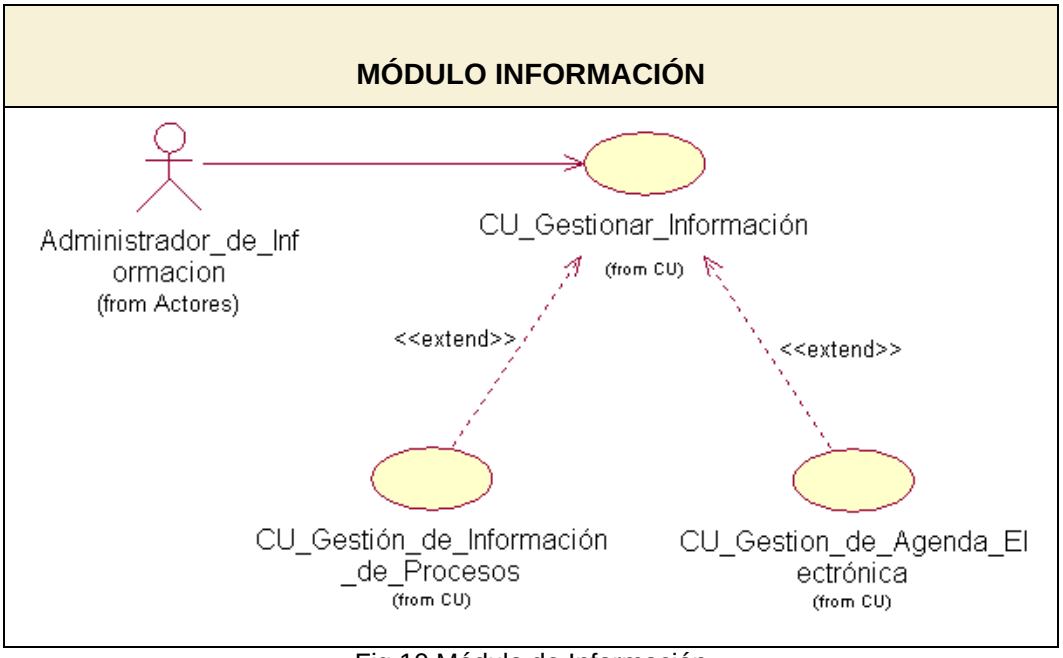

Fig.10 Módulo de Información

2.6 Vista lógica

En esta vista se describen las clases más importantes que formarán parte del ciclo de desarrollo del sistema de software. Se describen también los paquetes más abstractos que conforman el sistema y las relaciones de dependencia o de uso que existen entre ellos.

Mediante la siguiente figura se pretende mostrar la división del sistema en módulos, con el objetivo de promover la reusabilidad y facilitar el mantenimiento del sistema ante cambios en los procesos de negocio, garantizando que solo se modifique el módulo al que pertenece la funcionalidad afectada por el cambio. También se facilita el trabajo del equipo de desarrollo ya que permite que se puedan desarrollar módulos paralelos, con sus dependencias.

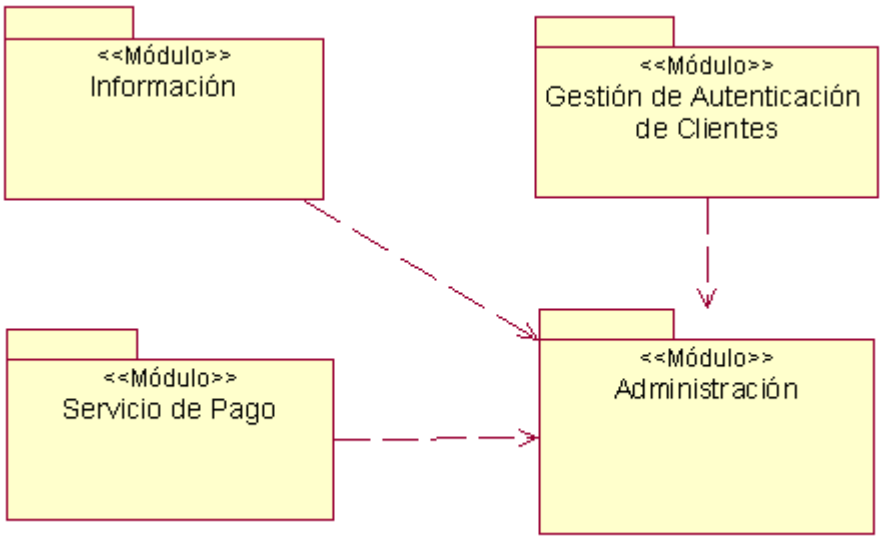

Fig.11 Representación de los Módulos

La distribución de la aplicación por capas:

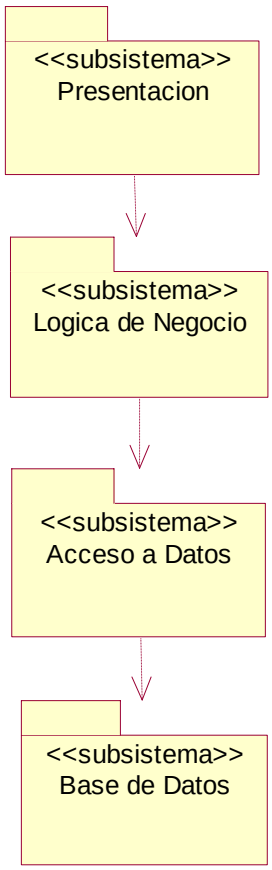

Fig.12 Estructura en capas.

Descripción de los paquetes		
Nombre	**Estereotipo**	**Descripción**
Presentación	Paquete	Conjunto de clases que componen las interfaces web del sistema.
Lógica de Negocio	Paquete	Conjunto de clases encargas de gestionar todos los procesos de la lógica de negocio.
Acceso a Datos	Paquete	Conjunto de clases que controlan el tratamiento de los datos.
Base de Datos	Paquete	Conjunto de tablas relacionadas que contienen los datos.

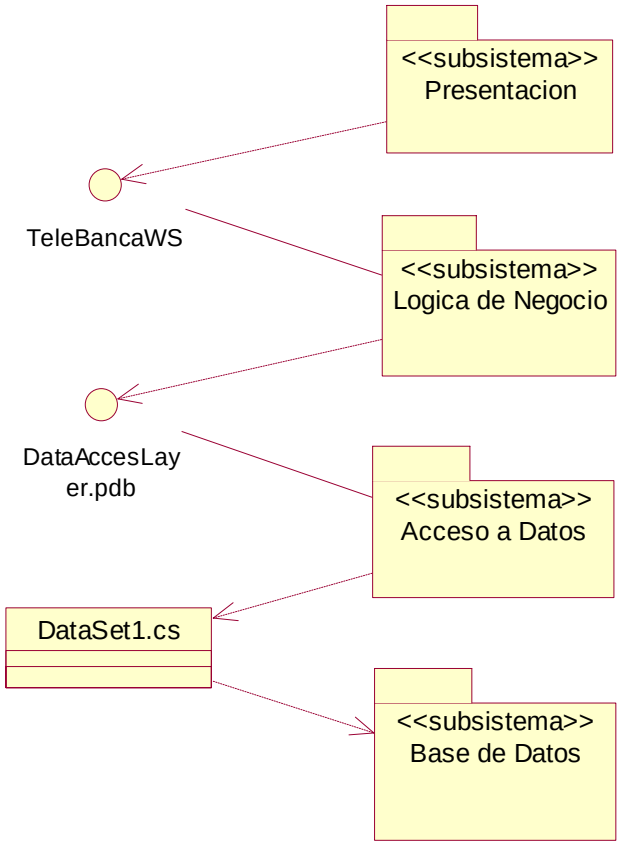

Fig.13 Subsistemas de diseño de la aplicación

En el paquete de **Presentación:**

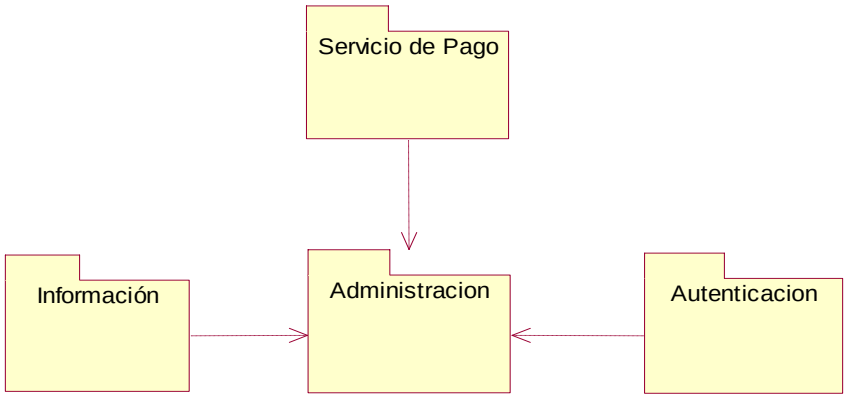

Fig.14 Paquetes de Presentación

Clases por Módulos	
Información	**Servicio de Pago**
CI_Gestionar_Información	UI_ModifcarDatos
CI_Gestionar_Agenda_Electrónica	UI_InsertarDatos
CI_Modificar_Informacion	UI_ConfigurarDatos
CI_Insertar_Informacion	UI_ConfigurarServicios
CI_Gestión_Informacion_Procesos	UI_ServiciosContratados
CI_Modificar_Informacion	UI_PagarSerComplejo
CI_Gestión_Informacion_Procesos	UI_PagarSerSimple
CI_Insertar_Informacion	UI_NumTarjeta
CI_Insertar_Información_Procesos	UI_CoordPin
CI_Modificar_Información_Procesos	UI_CoordMatriz

Clases por Módulos	
Administración	**Autenticación**
CI_SalvaryRestaurarDatos	CI_ComunicacionWebServiceBanco
CI_GestionarRoles	CI_ReimprimirPines
CI_GestionarUsuario	CI_ImprimirPines
CI_AutenticarUsuario	CI_ImprimirTarjetas
	CI_ReimprimirTarjetas

Para el paquete **Lógica de Negocio** se tiene:

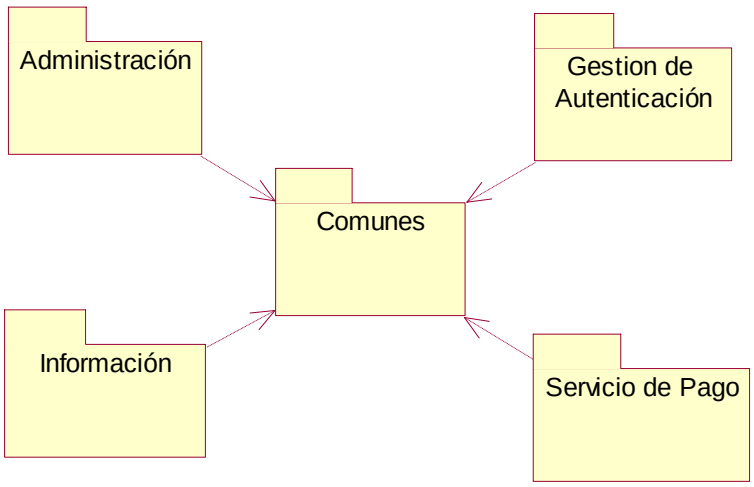

Fig.15 Paquetes de Lógica de Negocio.

Clases por Paquetes		
Administración	**Información**	**Comunes**
CC_DataHandler	CC_TeleBancaWS	CC_Usuario
CC_TeleBancaWS	CC_Usuario	CC_Tarjeta
CC_Usuario	CInterface_DataHandler	CC_OperacionesAutomaticas

Clases por Paquetes	
Gestión de Autenticación	**Servicio de Pago**
CC_Tarjeta	CC_Servicio
CC_TeleBancaWS	CC_UsuarioNegocio
CC_Usuario	CC_ManipuladorInformacion
CriptografiaTeleBanca	CC_ManipuladorFTP
CC_COperacionAutomatica	CC_ServicioComplejo
CC_DataHandler	CC_OperacionesAutomaticas

La estructura de la **Capa de Acceso** a datos es la siguiente:

Fig.16 Capa de Acceso a Datos

Manipulador de Datos: En este paquete está definida la clase que manipula el acceso a los datos.

Interfaz de Datos: Conjunto de clases persistentes del proyecto.

Clases por Paquetes	
Manipulador de Datos	**Interfaz de Usuario**
DataHandler	CE_DT_Usuario
DataHandler1	CE_DT_Configuración
DataHandler2	CE_DT_Roles
DataHandler4	CE_InformacionPersistente
	CE_EntidadPersistente
	CE_Tarjeta
	CE_Lote
	CE_Matriz
	CE_AccionesUsuario
	CE_Rol
	CE_Notificacion
	CE_Datos
	CE_Servicios
	CE_ConfiguracionSistema
	CE_Notificaciones
	CE_AuxTransacciones
	CE_PagoComplejo
	CE_Transacciones

2.7 Vista de implementación

La vista de implementación proporciona una descripción de las principales capas y subsistemas de componentes de la aplicación. Los paquetes principales de la aplicación por cada uno de los módulos son:

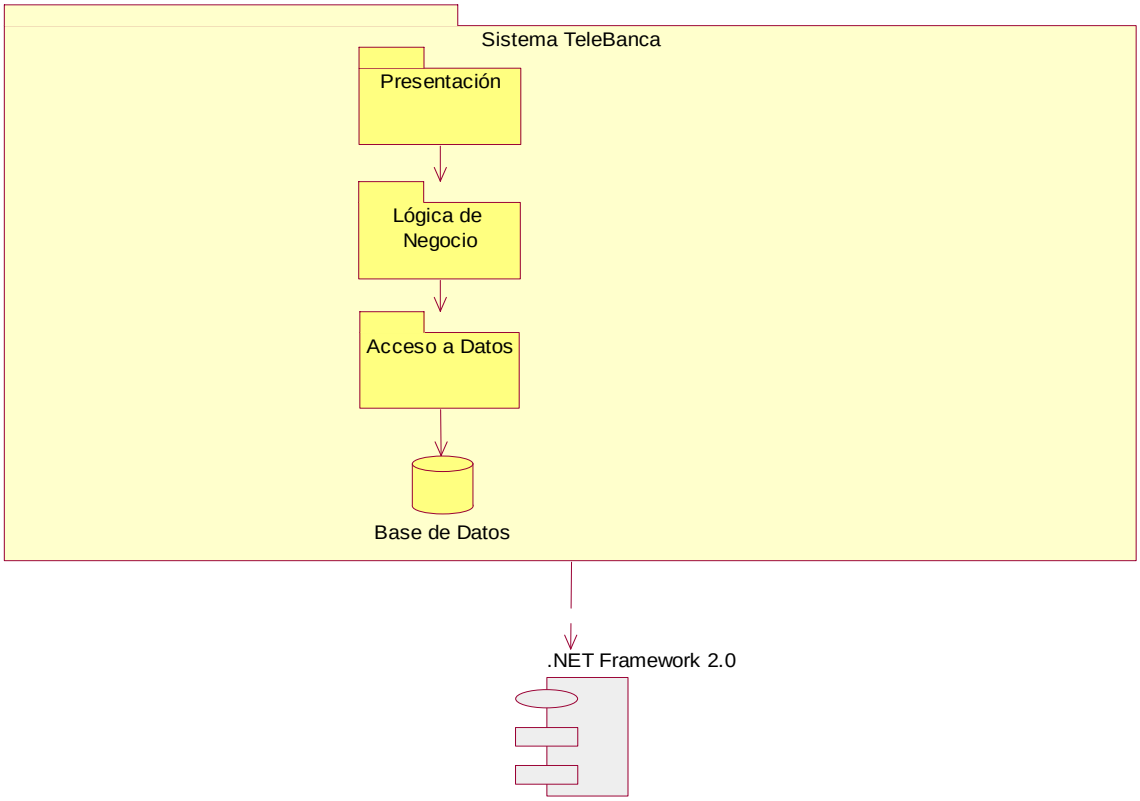

Fig.17 Vista general de Implementación.

Cada uno de los paquetes representados en el diagrama anterior encapsulan varios componentes que se relacionan entre sí para lograr un correcto funcionamiento del sistema, estableciendo además una dependencia general con el Framework .NET 2.0.

Los paquetes se encuentran distribuidos por capas de la siguiente forma:

Presentación:

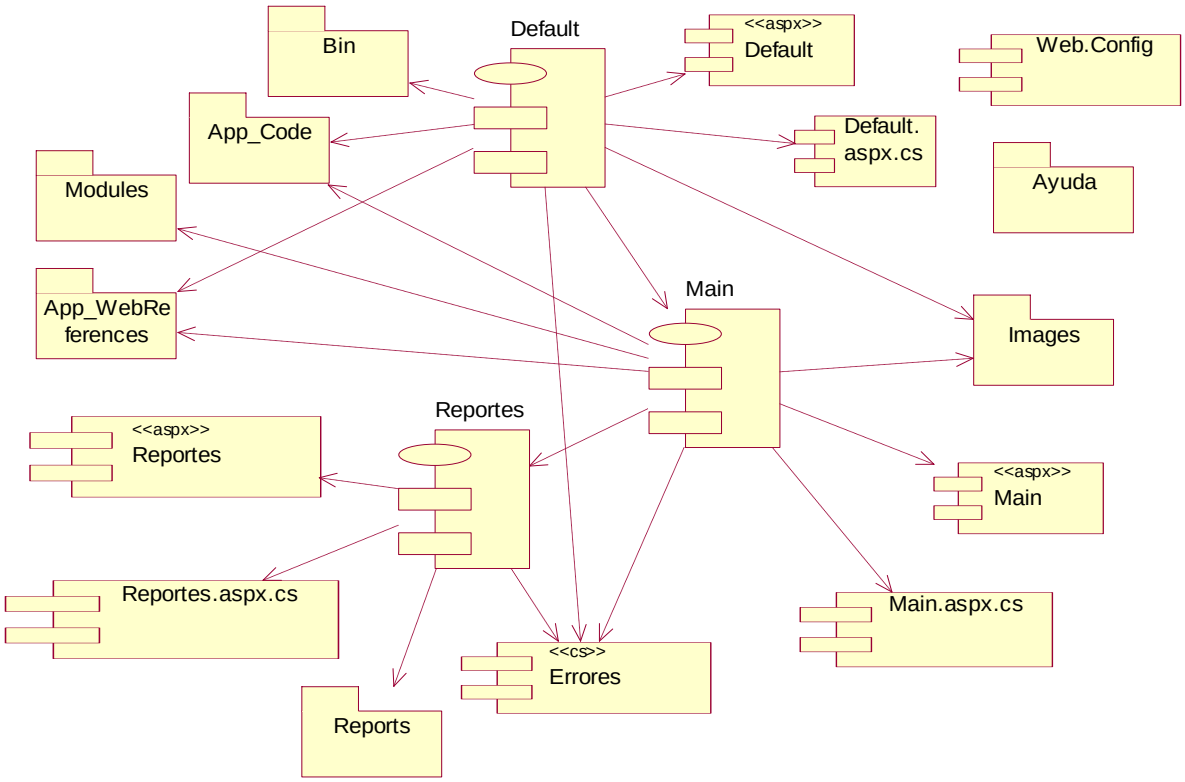

Fig.18 Componentes correspondientes a la Presentación.

Capa de Presentación		
Nombre	**Estereotipo**	**Descripcion**
Default.aspx	Componente	Componente de código asp.net de la página cliente.
Default.aspx.cs	Componente	Componente de código que se ejecuta en la pagina servidora.
Main.aspx	Componente	Componente de código asp.net de la página cliente.
Main.aspx.cs	Componente	Componente de código que se ejecuta en la pagina servidora.
Web.Config	Componente	Componente que tiene las configuraciones de la aplicación web.
Errores.cs	Componente	Componente encargado de manipular los errores.
Reportes.aspx	Componente	Componente de código asp.net de la página cliente.
Reportes.aspx.cs	Componente	Componente de código que se ejecuta en la pagina servidora
Ayuda	Paquete	Contiene el componente de ayuda.
Images	Paquete	Contiene el componente de las imágenes.
Bin	Paquete	Contiene el componente de las librerías de clases.

App_Code	Paquete	Contiene el componente de las clases auxiliares.
Modules	Paquete	Contiene los paquetes de componentes de los módulos y el paquete inicio.
App_WebReferences	Paquete	Contiene el componente de las interfaces para la conexión con la capa de Lógica de Negocio.
Reports	Paquete	Contiene el conjunto de componentes con extensión .rpt

Lógica de Negocio:

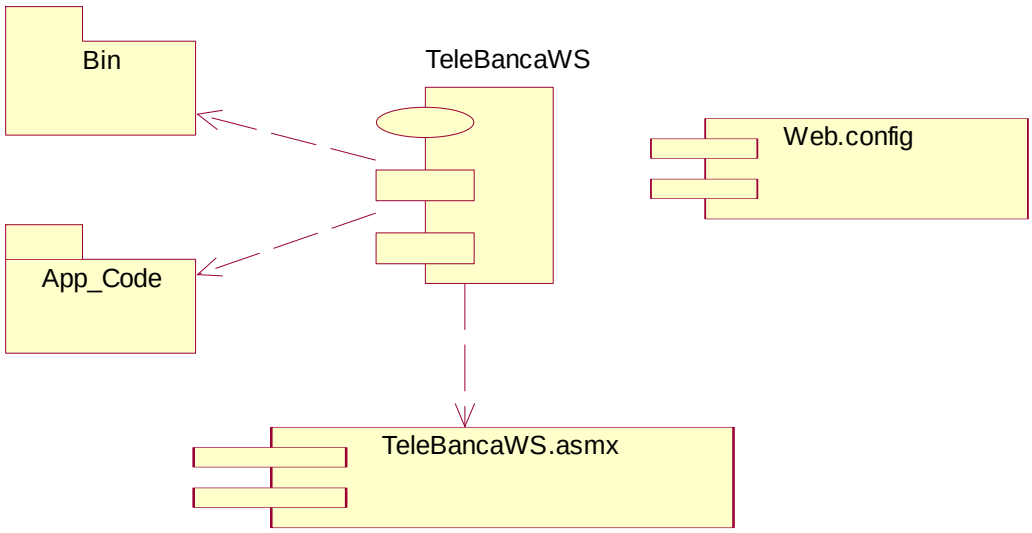

Fig.19 Componentes correspondientes a la Lógica de Negocio.

Capa de Lógica de Negocio		
Nombre	**Estereotipo**	**Descripción**
TeleBancaWS.asmx	Componente	Contiene la descripción HTML del webservice
Web.config	Componente	Contiene la información de configuración del sitio web.
Bin	Paquete	Contiene las librerías de clases externas de la capa.
App_Code	Paquete	Contiene los componentes de código de la capa.

Acceso a Datos:

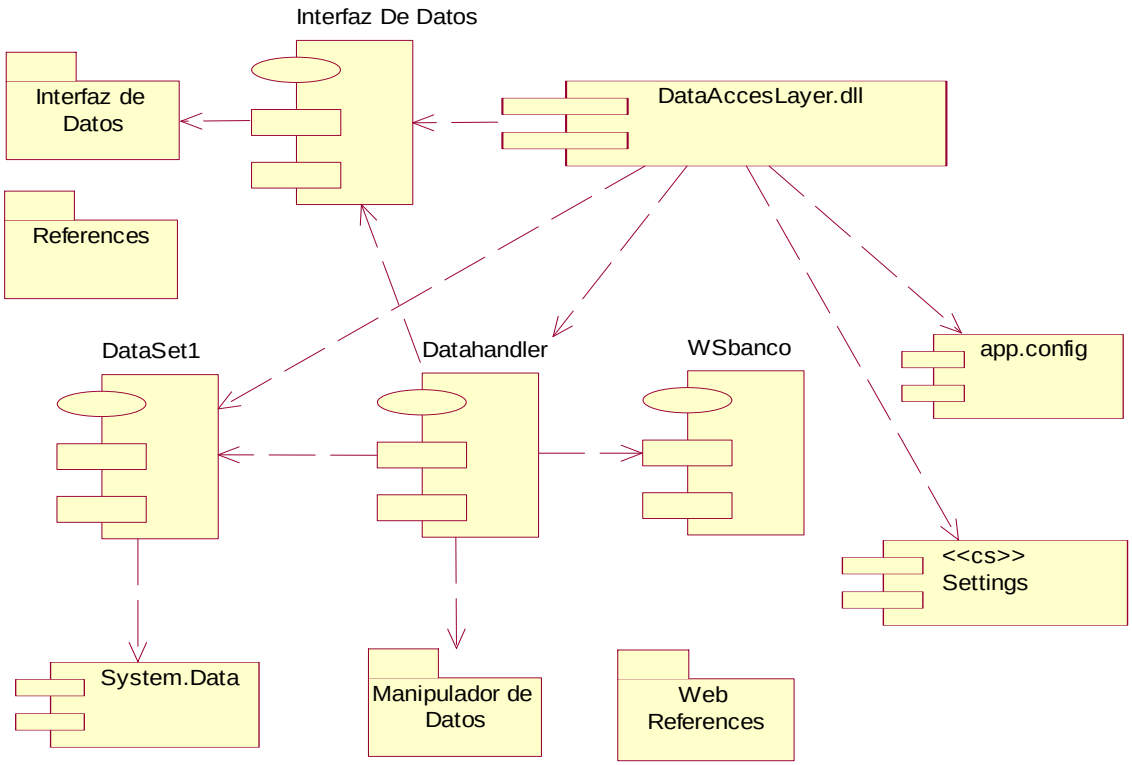

Fig.20 Componentes correspondientes al Acceso a Datos.

Capa de Acceso a Datos		
Nombre	**Estereotipo**	**Descripción**
Interfaz De Datos	Paquete	Contiene el componente de las clases persistentes que se utilizan en el sistema.
Manipulador de Datos	Paquete	Contiene el componente de las clases parciales Datahandler.
Web References	Paquete	Contiene el componente WSBanco.
References	Paquete	Contiene el componente de las librerías que utiliza el sistema.
System.Data	Componente	Componente de la librería de manipulación de datos del framework.
Settings.cs	Componente	Componente del fichero generado por el visual studio para manipular las propiedades del subsistema.
app.config	Componente	Componente de los ficheros de configuración de las .DLL.
DataAccesLayer.dll	Componente	Componente de librería de clases del subsistema.

Base de Datos:

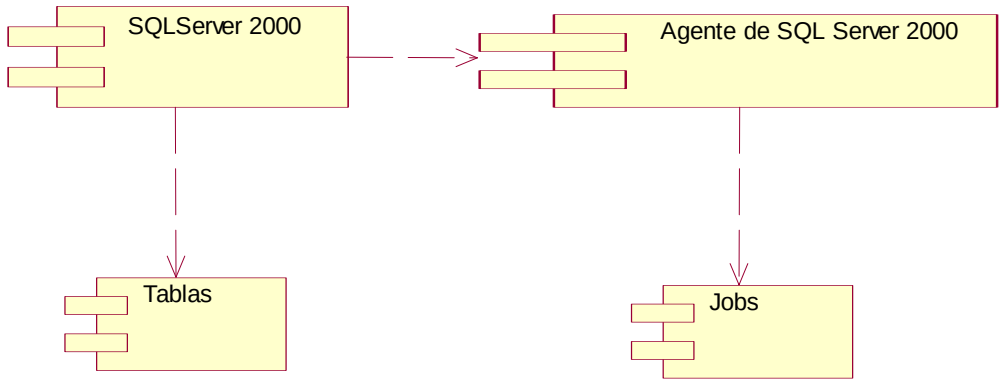

Fig.21 Componentes correspondientes a la Base de Datos.

Capa de Acceso a Datos		
Nombre	**Estereotipo**	**Descripción**
SQL Server 2000	Componente	Servidor de base de datos.
Agente de SQL Server 2000	Componente	Servicio encargado de realizar las actividades automatizadas en SQL Server 2000
Tablas	Componente	Contiene los atributos y registros para almacenar los datos del sistema.
Jobs	Componente	Objetos que se encargan de realizar tareas y se encuentran contenidos en el agente.

2.8 Vista de Despliegue

La vista de despliegue suministra una base para la comprensión de la distribución física de un sistema a través de nodos, y propone como se satisfacen los requisitos no funcionales de software y hardware e influye en el rendimiento.

MODELO DE DESPLIEGUE

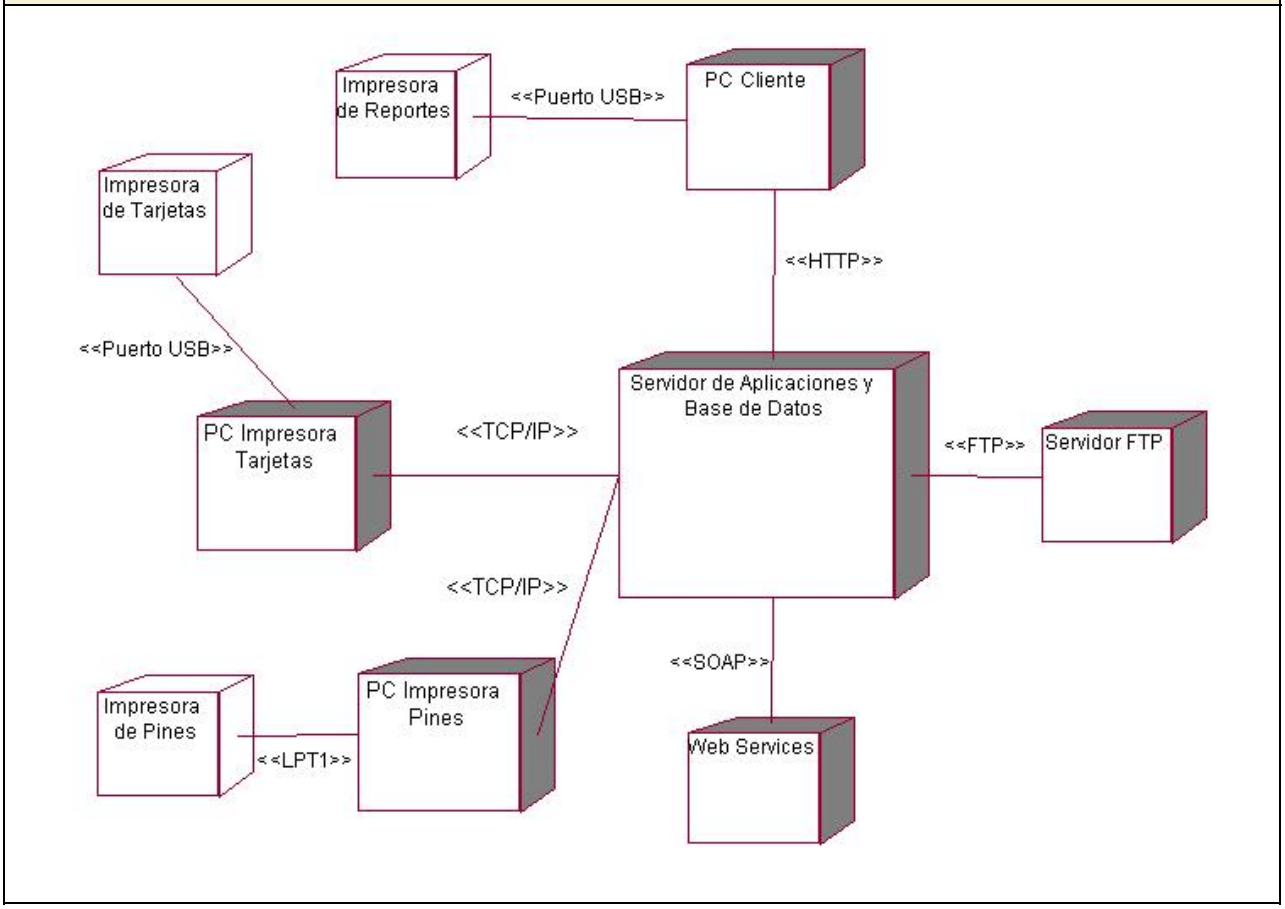

Fig.22 Vista de Despliegue

Nodo PC Cliente: Este ordenador corresponde con el puesto de trabajo de las operadoras de la empresa y del administrador del sistema los cuales serán los únicos trabajadores que van a tener contacto directo con la aplicación, desde aquí se puede mandar a imprimir reportes, pines o tarjetas.

Nodo PC Impresora Tarjetas: A través de esta PC se podrá efectuar únicamente la impresión de las tarjetas, no tiene que ser necesariamente directamente de desde la misma, puesto que este nodo no es un puesto de trabajo.

Nodo PC Impresora Pines: A través de esta PC se podrá efectuar solamente la impresión de pines, al igual que con la PC Impresora Tarjetas, este nodo no es un puesto de trabajo y la solicitud de impresión de pines puede ser remota.

Nodo Web Services: Este nodo contiene el web service propio de la empresa, el cual estará en constante intercambio de información con la aplicación, a través del web service de la misma.

Nodo Servidor de Aplicaciones y de Base de Datos: Constituye el nodo fundamental de la empresa, es importante tener presente que en él se encuentra la aplicación conjuntamente con la base de datos, así que todos los demás nodos ya sean puestos de trabajo o no deben poder conectarse al mismo. Este nodo garantizará el buen funcionamiento de la aplicación.

Dispositivos Impresora: Los dispositivos de scanner e impresora satisfacen los requisitos de los clientes de impresión de reportes generados por la aplicación, los dispositivos Impresora de Tarjetas e Impresora de Reportes son impresoras láser y van a estar conectados por USB, mientras que la Impresora de Pines es de matriz de punto y estará conectada a través del puerto LPT1.

Nodo Servidor FTP: En este nodo se guarda la información que brinda las empresas sobre los servicios complejos que faltan por pagar.

2.9 Conclusiones

En este capítulo se muestra la propuesta de solución del sistema mediante la línea base de la arquitectura y el documento de descripción de la arquitectura. Se muestran los patrones estudiados con su fundamentación. Se definen los requisitos no funcionales del sistema, es decir, las herramientas y tecnologías necesarias para el sistema. Se escoge la metodología RUP para llevar a cabo la definición de la arquitectura. Se exponen las cuatro vistas planteadas por RUP necesarias para la comprensión y el desarrollo del mismo.

CONCLUSIONES

El desarrollo de la arquitectura de software es una de las etapas fundamentales en el desarrollo de software, pues es aquí donde los profesionales aportan todos sus conocimientos, creatividad y experiencia para crear la mejor propuesta de solución que se dará al cliente que cumpla con los requerimientos funcionales y no funcionales establecidos para el sistema en desarrollo, así como sus preocupaciones principales de lo que esperan del sistema.

Es de vital importancia que todo sistema de software esté respaldado por una arquitectura sólida que facilite el entendimiento del mismo, que sea capaz de organizar el desarrollo, fomentar la reutilización y hacer evolucionar el sistema.

Se realizó un estudio profundo acerca de varios estilos y patrones arquitectónicos que existen en la actualidad, para hacer uso de las mejores técnicas de diseño arquitectónico.

Se desarrolló una arquitectura de cuatro capas que facilitó el desarrollo paralelo del sistema, el mantenimiento y soporte de la aplicación al tratarse cada capa de forma individual. Se estructuró el sistema en cuatro módulos divididos por funcionalidades específicas, para facilitar el trabajo del equipo de desarrollo.

Se definieron componentes y funcionalidades reusables que fueron empleados en cada uno de los módulos durante el ciclo de vida del sistema, esto permitió acortar el tiempo de desarrollo de la aplicación.

Se cumplieron todos los objetivos trazados en esta investigación y a la vez que se dio respuesta a la pregunta planteada en el problema científico a partir de definir un marco arquitectónico para el proyecto TeleBanca.

RECOMENDACIONES

- Se recomienda aplicar los métodos de evaluación de la arquitectura para identificar las posibles debilidades.
- Se recomienda mantener un constante refinamiento de la arquitectura a lo largo del ciclo de desarrollo.
- Se recomienda al equipo de desarrollo de la versión 2.0 que proponga un nuevo diseño de arquitectura que tenga en cuenta las características de una aplicación de escritorio.

REFERENCIAS BIBLIOGRÁFICAS

[1] Edsger Dijkstra. “The Structure of the THE Multiprogramming system.” Communications of the ACM, 26(1), pp. 49-52, Enero de 1983.

[2] Frederick Brooks Jr. The mythical man-month. Reading, Addison-Wesley, 1975.

[3] Dewayne E. Perry y Alexander L. Wolf. “Foundations for the study of software architecture”. ACM SIGSOFT Software Engineering Notes, 17(4), pp. 40–52, Octubre de 1992.

[4] Shaw, M. y Garlan, D. (1996). Software Architecture. Perspectives of an Emerging Discipline, Prentice Hall.

[5] Jacopbson, I. (1999). The Unified Software Development Process, Addison Wesley.

[6] Philippe Kruchten. “The 4+1 View Model of Architecture.” IEEE Software 12(6), pp. 42-50, Noviembre de 1995.

[7] Grady Booch, James Rumbaugh e Ivar Jacobson. El Lenguaje Unificado de Modelado. Madrid, Addison-Wesley, 1999.

[8] Mary Shaw. “Some Patterns for Software Architecture,” en Pattern Languages of Program Design, Vol. 2, J. Vlissides, J. Coplien, y N. Kerth (eds.), Reading, Addison-Wesley, pp. 255-269, 1996.

[9] Robert Monroe, Andrew Kompanek, Ralph Melton y David Garlan. “Stylized architecture, design patterns, and objects”. http://citeseer.nj.nec.com/monroe96stylized.html.

[10] Robert Monroe, Andrew Kompanek, Ralph Melton y David Garlan. “Architectural Styles, design patterns, and objects”. IEEE Software, pp. 43-52, Enero de 1997.

[11] Sharon White y Cuauhtémoc Lemus-Olalde. “The software architecture process”. http://nas.cl.uh.edu/whites/webpapers.dir/ETCE97pap.pdf, 1997.

[12] Gamma, E. (2002). Patrones de Diseño, Addison-Wesley

[13] Rational Software.

[14] I.Jacobson, G.Booch, J.Rumbaugh. “El Proceso Unificado de Desarrollo de Software”. Madrid, PEARSON EDUCACION, S.A, 2000.

[15] David Garlan y Mary Shaw. “An introduction to software architecture”. CMU Software Engineering Institute Technical Report, CMU/SEI-94-TR-21, ESC-TR-94-21, 1994.

BIBLIOGRAFÍA

Arquitecturas de Aplicaciones en N-capas con .NET. Noviembre 2002, Madrid.

Douglas Schmidt, Michael Stal, Hans Rohnert, Frank Buschmann, PATTERN-ORIENTED SOFTWARE ARCHITECTURE. Volumen 1.

Douglas Schmidt, Michael Stal, Hans Rohnert, Frank Buschmann, PATTERN-ORIENTED SOFTWARE ARCHITECTURE. Volumen 2.

César Colado Rodríguez, Diseño y desarrollo de aplicaciones web multidispositivo. Febrero de 2003.

Carlos E. Cuesta. Arquitecturas de Software. Ingeniería del Software I, Universidad Rey Juan Carlos.

Society, S. E. S. C. o. t. I. C. (2000). "IEEE Recommended Practice for Architectural Description of Software-Intensive Systems."

Bass, L., Clements, P., y Kazman, R. (1998) Software Architecture in Practice. Addison Wesley.

Booch, G. (1999). The Unified Modeling Language User Guide, Addison Wesley.

Bosch., J. (2000). Design & Use of Software Architectures”. Addison Wesley.

Brown, A. W. y. W., K. C. (1998) The Current State of CBSE. IEEE Software

Carlos Reynoso, N. K. (2004) Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.

Clements, P. (1996). A Survey of Architecture Description Languages. Proceedings of the International Workshop on Software Specification and Design.

Dewayne E. Perry, A. L. W. (1992). "Foundations for the study of software architecture."

Gamma, E. (2002). Patrones de Diseño, Addison-Wesley.

IEEE-1471 (2000) Recommended Practice for. Architectural Description of Software-Intensive Systems

ISO/ITU-T (1995) Reference Model for Open Distributed Processing. International Organization for Standarization.

Jacopbson, I. (1999). The Unified Software Development Process., Addison Wesley.

Kruchten, P. (1995). The 4+1 View Model of Architecture., IEEE Software.

Len Bass, P. C., Rick Kazman (2003). Software Architecture in Practice (2nd Ed.), Addison Wesley.

Mary Shaw, D. G. (1996). Software Architecture. Perspectives on an Emerging Discipline, Prentice Hall.

Medvidovic, N. y. R., D. S. (1997) Domains of Concern in Software Architectures and Architecture Description Languages. En USENIX Conf. on Domain-Specific Languages, Santa Barbara (Estados Unidos).

Robert Monroe, A. K., Ralph Melton y David Garlan (1997) Architectural Styles, design patterns, and objects. IEEE Software.

Rumbaugh, J. (1991). Object-Oriented Modeling and Design, Prentice Hall.

Rumbaugh, J. (1999). The Unified Modeling Language Reference Manual, Addison Wesley.

SEI (1990) Workshop on Domain-Specific Software Architectures. Software Engineering Institute.

Szyperski, C. (2000). Components and Architecture. Software Development Online: 10.

Toledo, Octubre 2002. .NET, servicios Web XML y la plataforma de desarrollo Microsoft.

Celestino Güemes Seoane. Nuevas Arquitecturas de Desarrollo Web. Director de Innovación Tecnológica, Mundivía.

GLOSARIO DE TÉRMINOS

Sistema Operativo: es un conjunto de programas destinados a permitir la comunicación del usuario con un computador y gestionar sus recursos de una forma eficaz.

IBM (**I**nternational **B**usiness **M**achines): es una empresa que fabrica y comercializa hardware, software y servicios relacionados con la informática.

Hardware (soporte físico): se utiliza generalmente para describir los artefactos físicos de una tecnología. Es el conjunto de elementos físicos que componen una computadora Disco Duro, CD-ROM, etc.

Software (soporte lógico): los componentes intangibles de una computadora, es decir, al conjunto de programas y procedimientos necesarios para hacer posible la realización de una tarea específica.

Informática: es la disciplina que estudia el tratamiento automático de la información utilizando dispositivos electrónicos y sistemas computacionales. Es la unión sinérgica del cómputo y las comunicaciones.

UTP (del inglés **U**nshielded **T**wisted **P**air, par trenzado no apantallado): es un tipo de conductor utilizado, principalmente para comunicaciones.

Mbps (**M**ega**b**it **p**or **s**egundo): es una unidad que se usa para cuantificar la velocidad de transmisión de información.

MB (**M**ega**B**yte): es una unidad de medida de cantidad de datos informáticos.

GB (**G**iga**B**yte): es una unidad de medida informática equivalente a mil millones de bytes.

RAM (**R**andom **A**ccess **M**emory): memoria de acceso aleatorio ó memoria de acceso directo.

TCP/IP: es un conjunto de protocolos de red que permiten la transmisión de datos entre redes de computadoras.

OS/360: es un sistema operativo producido por IBM.

IDE (Integrated Development Environment): Un entorno de desarrollo integrado, es un programa compuesto por un conjunto de herramientas para un programador. Un IDE es un

entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica GUI. Puede dedicarse en exclusiva a un sólo lenguaje de programación o bien, poder utilizarse para varios.

Estereotipo: Define un nuevo significado de la semántica para el elemento a modelar.

TeleBanca: Es un proyecto de la UCI, para la realización de un centro de llamadas, para efectuar pagos de servicios a la población, por ejemplo: pago de servicios Telefónicos, Eléctricos, Multas de Transito y Aguas de la Habana, etc., contra los saldos en sus cuentas de tarjetas red mediante la vía telefónica.

Servicios: es un conjunto de actividades que buscan responder a una o más necesidades de un cliente.

Cliente: es un ordenador que accede a recursos y servicios brindados por otro llamado Servidor, generalmente en forma remota.

Servidor: Una aplicación informática o programa que realiza algunas tareas en beneficio de otras aplicaciones llamadas clientes.

Cliente/Servidor: Esta arquitectura consiste básicamente en que un programa (cliente informático) realiza peticiones a otro programa (servidor) que les da respuesta.

Servicio WEB (en inglés *Web service*) es una colección de protocolos y estándares que sirven para intercambiar datos entre aplicaciones. Distintas aplicaciones de software desarrolladas en lenguajes de programación diferentes, y ejecutadas sobre cualquier plataforma, pueden utilizar los servicios web para intercambiar datos en redes de ordenadores.

POSA: **P**attern-**O**riented **S**oftware **A**rchitecture

SEI (**S**oftware **E**ngineering **I**nstitute) es un instituto federal estadounidense de investigación y desarrollo, fundado por el Congreso Estadounidense en 1984 para desarrollar modelos de evaluación y mejora en el desarrollo de software, que dieran respuesta a los problemas que generaba al ejército estadounidense la programación e integración de los sub-sistemas de software en la construcción de complejos sistemas militares. Financiado por el Departamento de Defensa estadounidense y administrado por la Universidad Carnegie Mellon.

IEEE corresponde a las siglas de The **I**nstitute of **E**lectrical and **E**lectronics **E**ngineers, el Instituto de Ingenieros Eléctricos y Electrónicos, una asociación técnico-profesional mundial dedicada a la estandarización, entre otras cosas. Es la mayor asociación internacional sin fines de lucro formada por profesionales de las nuevas tecnologías, como ingenieros eléctricos, ingenieros en electrónica, científicos de la computación e ingenieros en telecomunicación.

URIs (**U**niform **R**esource **L**ocator*):* es un localizador uniforme de recurso. Es una secuencia de caracteres, de acuerdo a un formato estándar, que se usa para nombrar recursos, como documentos e imágenes en Internet, por su localización.

HTTP (**H**yper**T**ext **T**ransfer **P**rotocol): protocolo de transferencia de hipertexto es el protocolo usado en cada transacción de la Web (WWW). El hipertexto es el contenido de las páginas web, y el protocolo de transferencia es el sistema mediante el cual se envían las peticiones de acceso a una página y la respuesta con el contenido.

Protocolo: Conjunto de normas y procedimientos útiles para la transmisión de datos, conocido por el emisor y el receptor.

WWW (**W**orld **W**ide **W**eb): es un sistema de documentos de hipertexto enlazados y accesibles a través de Internet. Con un navegador Web, un usuario visualiza páginas Web que pueden contener texto, imágenes u otros contenidos multimedia.

XML (sigla en inglés de e**X**tensible **M**arkup **L**anguage): Lenguaje de marcado extensible es un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo y casi cualquier cosa imaginable. Permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil.

SOAP (siglas de **S**imple **O**bject **A**ccess **P**rotocol): Protocolo de Acceso Simple a Objetos es un protocolo estándar define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML.

GUI (en inglés **G**raphical **U**ser **I**nterface): Interfaz Grafica de Usuario es un tipo de interfaz de usuario que utiliza un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz. Habitualmente las acciones se realizan mediante manipulación directa para facilitar la interacción del usuario con la computadora.

UDDI (**U**niversal **D**escription, **D**iscovery and **I**ntegration): es uno de los estándares básicos de los servicios Web cuyo objetivo es ser accedido por los mensajes SOAP y dar paso a documentos WSDL, en los que se describen.

ASP (**A**ctive **S**erver **P**ages): es una tecnología del lado servidor de Microsoft para páginas web generadas dinámicamente, que ha sido comercializada como un anexo a Internet Information Server (IIS).

IIS (**I**nternet **I**nformation **S**erver): es una serie de servicios para los ordenadores que funcionan con Windows. Convierte a un ordenador en un servidor, es decir que en las computadoras que tienen este servicio instalado se pueden publicar páginas web tanto local como remotamente (servidor web)

Smalltalk: es un sistema informático que permite realizar tareas de computación mediante la interacción con un entorno de objetos virtuales. Metafóricamente, se puede considerar que un Smalltalk es un mundo virtual donde viven objetos que se comunican mediante el envío de mensajes.

Centro de Llamadas: Lugar único donde se recibirán todas las llamadas telefónicas, las operadoras ejecutarán las transacciones a partir de las indicaciones recibidas por el cliente.