ESCUELA POLITÉCNICA NACIONAL
ESCUELA DE INGENIERIA

PROPUESTA DE COMUNICACIÓN .NET PARA CONEXIÓN CON MAIN FRAME OS/390 VIA TCP/IP y SNA.

PROYECTO PREVIO A LA OBTENCIÓN DEL TITULO DE INGENIERO INFORMATICO MENCION REDES DE INFORMACION

NELSON ENRIQUE PORRAS VASQUEZ
CRISTIAN RUBEN FLORES MOSQUERA

DIRECTOR: ING. MARCO JARRIN

Quito, Septiembre 2004

DECLARACIÓN

Nosotros, Nelson Enrique Porras Vásquez y Cristian Rubén Flores Mosquera, declaramos bajo juramento que el trabajo aquí descrito es nuestra autoría. Que no ha sido previamente presentada para ningún grado o calificación profesional; y, que hemos consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedemos nuestros derechos de propiedad intelectual correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normativa institucional vigente.

Nelson Porras **Cristian Flores**

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por Nelson Enrique Porras Vásquez y Cristian Rubén Flores Mosquera, bajo mi supervisión.

_____

Ing. Marco Jarrín
DIRECTOR DE PROYECTO

AGRADECIMIENTO

A Dios por darme la oportunidad de culminar una etapa más en mi vida y brindarme su apoyo y presencia en todas las metas fijadas.

A mi esposa Lorena y mi hija Samantha por el amor y apoyo incondicional que tuvieron conmigo durante las largas jornadas de estudio.

A mis padres por el amor y apoyo incondicional que siempre me han brindado.

A mis amigos y profesores con quienes he compartido largas jornadas de aprendizaje y enseñanza.

Al Ing. Marco Jarrín por su apoyo y dirección en este proyecto.

Nelson.

AGRADECIMIENTO

A Dios por darme la oportunidad de culminar una etapa más en mi vida y brindarme su apoyo y presencia en todas las metas fijadas.

A mis padres por el amor y apoyo incondicional que siempre me han brindado.

A mis amigos y profesores con quienes he compartido largas jornadas de aprendizaje y enseñanza.

Al Ing. Marco Jarrín por su apoyo y dirección en este proyecto.

Cristian.

DEDICATORIA

A Dios por cuidar y guiar mis pasos en todo momento.

A mi esposa Lorena por todo el amor, comprensión y dedicación a mi entregada.

A mi hija Samantha por su inocencia, ternura y amor, y por incentivar mi vida.

A mis padres Matilde y Salomón por su apoyo durante toda mi vida.

A mis hermanos y familiares por su contribución para alcanzar esta meta.

Y, de manera especial a mi Padre a quien le hubiera encantado verme finalizar

Una etapa más en mi vida profesional. Que Dios lo tenga a su lado.

Nelson

DEDICATORIA

A Dios por cuidar y guiar mis pasos en todo momento.

A mis hijos Jhony y Jhojan por su inocencia, ternura y amor, y por incentivar mi vida.

A mis padres Laura y Héctor por su apoyo durante toda mi vida.

A mis hermanos y familiares por su contribución para alcanzar esta meta.

Cristian

RESUMEN

Actualmente el creciente avance tecnológico dado en los últimos años ha provocado que innumerables empresas busquen renovar y mejorar tanto sus procesos como sus equipos con el fin de posicionarse de mejor manera en el mercado, brindando a sus clientes información rápida y precisa e inclusive proporcionando facilidades sin tener que moverse de su casa.

El acceder a esta nueva tecnología requiere de un minucioso análisis de costes y proveedores por parte de las empresas quienes quieren adquirir dicha tecnología; por tal motivo, el presente proyecto pretende apoyar a la empresa caso de estudio con una opción adicional de comunicación compitiendo con otros productos y apoyados en un estándar de desarrollo de aplicaciones.

Se ha utilizado herramientas de desarrollo que actualmente están en la punta de la tecnología como es el caso de .NET; esto tomando en cuenta que la empresa caso de estudio tiene licenciamiento de dicha herramienta y sus nuevas aplicaciones se basan en la arquitectura proporcionada por .NET.

PRESENTACION

Con el propósito de brindar al lector una rápida comprensión al contenido de este proyecto, el presente trabajo se lo ha estructurado de la siguiente manera:

El Capítulo 1 *"Introducción".* Contiene la situación actual de la empresa caso de estudio.

El Capítulo 2 "*Características de las herramientas Utilizadas".* Presenta las características básicas de cada una de las herramientas a utilizadas, así como también todo el sustento teórico para el desarrollo del presente trabajo.

El Capítulo 3 "*Construcción De La Herramienta Estándar De Comunicación Web-Os/390 Y Definición De La Metodología Base A Utilizar",* comprende la definición de estándares y reglas a seguir en el desarrollo de aplicaciones usando el modelo de comunicación planteado; además se muestra el código desarrollado de la librería de comunicación.

El Capítulo 4 *"Conclusiones y Recomendaciones",* obtenidas al final de la realización del proyecto.

INDICE RESUMIDO

INDICE GENERAL

INDICE DE FIGURAS

CAPITULO I.

INTRODUCCION

El avance tecnológico de los últimos años a presionado a las empresas a migrar o desarrollar sus aplicaciones en plataformas actuales, las mismas que para su total implementación requieren de una actualización similar en cuanto a Hardware; esto repercute obviamente en el presupuesto de cualquier empresa cuyas intenciones sea la de actualizar sus procesos y equipos tecnológicos.

Hoy por hoy es mucho mas apetecible el tener una propuesta tecnológica cuyos pre-requisitos de Hardware y Software no sobrepase el presupuesto de una empresa por mas grande que esta sea. Por lo tanto, la presente propuesta dará a conocer de alguna manera los beneficios que tendrá su implementación.

Este es el caso de la solución informática que actualmente se tiene en la Institución caso de estudio, la misma que involucra costos extremadamente altos, lo cual desvirtúa la razón misma de ser de la Institución. Es por esto, que nace esta propuesta para aplacar una pequeña parte del problema mediante el diseño y estandarización de clases y librerías de comunicación con las que funcionara el core tecnológico de la Institución desarrolladas con herramientas de punta.

1.1. DESCRIPCION DEL PROBLEMA

A partir del año 2000 la Institución caso de estudio, implementa una solución tecnológica de Hardware y Software Bancario adquirido a una empresa de software; dicha solución especificaba elevados costos principalmente en lo que se refiere a Hardware. El Banco en su necesidad de abaratar costos implementa paralelamente proyectos destinados a dicha causa, entre los cuales están migraciones masivas de herramientas de programación y base de datos y principalmente sustitución de los equipos principales que proporcionan el Sistema al Banco.

La solución informática que se implementaba en ese entonces requería de especificaciones muy costosas de Hardware, principalmente para el equipo MAINFRAME OS/390. Esta requería de por lo menos 16 procesadores para soportar la carga de clientes que tiene el Banco.

Por otra parte, el Front End estaba desarrollado en la plataforma Java versión 1.1.8 la cual requería de máquinas de por lo menos 128MB de memoria principal, que a pesar de ser una herramienta actual, el diseño en el cual estaba implementada no era de fácil migración para la WEB que era uno de los objetivos que perseguía el Banco.

Para solucionar este inconveniente el Banco inicia un proyecto cuya finalidad es cambiar el Front End de Java a un Front End WEB utilizando herramientas

tecnológicas actuales como es el caso del FrameWork .NET, en cuya implementación surge la necesidad de establecer una vía de comunicación hacia el OS / 390, y se llega a la conclusión de comunicarse vía COMTI's de Microsoft.

Pero, con algunas limitantes que describiremos en los capítulos posteriores a cerca del uso de los COMTI's, y principalmente el atarse a un proveedor específico hace que el desarrollo de la nueva aplicación propuesta por el Banco sea dependiente de aplicaciones comerciales; razón por la cual se pone a consideración la presente propuesta de comunicación .NET hacia el Host Central (OS / 390).

1.2. ALCANCE DEL PROYECTO

Como alcance del proyecto se plantea estandarizar la comunicación entre el Front End mediante librerías estándar de comunicación para interconectar la WEB con el MainFrame OS/390, además de especificar los estándares a ser utilizados en el desarrollo de las aplicaciones dentro de la Institución.

1.3. METODOLOGÍA

Apoyado en la decisión institucional en cuanto a implementar su Front End con el ambiente de programación .NET; utilizaremos para la presente propuesta la metodología conocida como Microsoft Solutions Framework versión 3.1 (MSF), cuyos procedimientos soportan ampliamente todo el ciclo de vida de un Proyecto.

Por tal motivo, se reseñará brevemente a continuación la metodología MSF:

Microsoft Solutions Framework

MSF ofrece una guía de cómo organizar los recursos alrededor de la planificación, desarrollo e implementación exitosa de soluciones tecnológicas.

Modelos y Disciplinas de MSF

MSF utiliza un marco de trabajo probado cuyo objetivo es la de apoyar a la culminación exitosa de un proyecto. Los modelos utilizados son:

Modelo de Equipo

Este modelo organiza a las personas con el fin de que estas realicen el trabajo del proyecto asegurándose el cumplimiento de todas las metas al enlazar cada rol del equipo con una responsabilidad importante del proyecto. Define como máximo seis roles o personas cuyas responsabilidades se definirán al inicio del proyecto. Estos roles a su vez pueden abarcar varios recursos con los que se contará para cumplir sus actividades y responsabilidades.

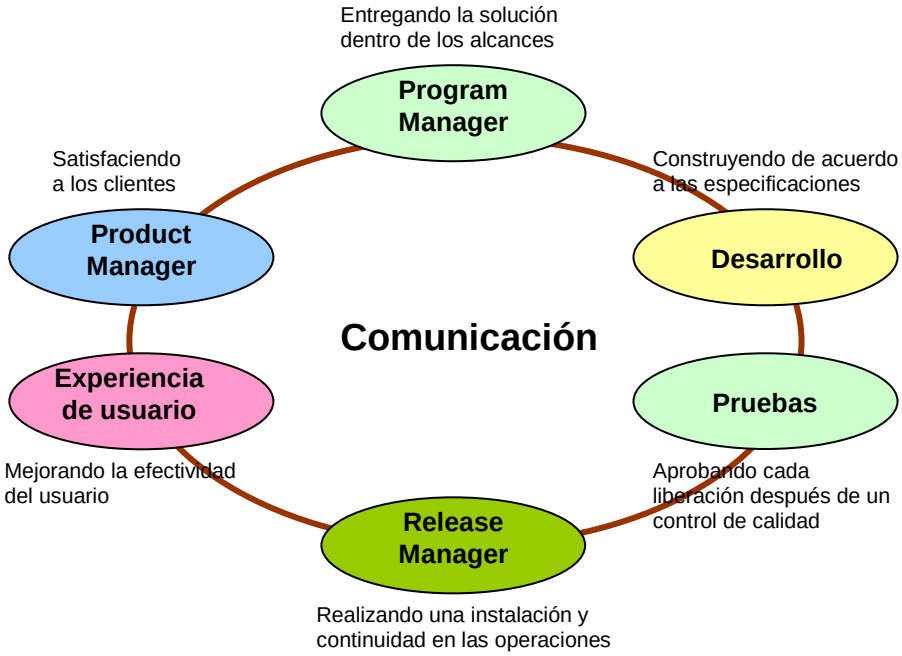


Gráfico 1. Modelo de Equipos de MSF

Modelo de Procesos

Este modelo organiza los procesos necesarios para crear y entregar una solución al ordenarla en relación al tiempo, para lo cual, la solución es dividida en diferentes fases marcadas por hitos.

Gráfico 2. Hitos y Fases del Modelo de Procesos de MSF

- **Visión**
  En esta fase se crea una visión de alto nivel sobre las metas, limitantes y de la solución del proyecto.

- **Planificación**
  En esta fase se crea la arquitectura y diseño de la solución, así como los planes del proyecto con su agenda.

- **Desarrollo**
  En esta fase se contruye las funcionalidades, componentes y otros elementos descritos en las especificaciones.

- **Estabilización**
  En esta fase se incluye el mejoramiento de la calidad de la solución para alcanzar los criterios de aceptación para el paso a producción.

- **Implantación**
  En esta fase se implemente la solución en el ambiente de producción.

Disciplina de manejo de proyectos

Asegura que las actividades del proyecto sean racionalizadas y que apoyen al éxito del equipo en lugar de entorpecerlo.

Disciplina de manejo de riesgos

Maneja proactivamente los riesgos del proyecto con el fin de minimizar las sorpresas y actividades costosas.

Disciplina de manejo de conocimiento

Utiliza los proyectos como oportunidades de aprendizaje, para lo cual identifica proactivamente las destrezas de los miembros del equipo que trabajará en el proyecto.

A continuación se describe los principios fundamentales de MSF aplicados al Modelo de Equipos:

- Trabajar en función a una visión compartida.
- Enfoque en el valor del negocio.
- Mantenerse ágil, esperar cambios.
- Reforzar a los miembros del equipo.
- Adoptar comunicaciones abiertas.
- Establecer responsabilidades claras y compartidas.

El desarrollo de esta metodología para el presente proyecto se encuentra distribuido en la sección de Anexos correspondientes a los siguientes Apéndices:

APENDICE B **– METODOLOGIA MSF – DOCUMENTO DE ANTEPROYECTO**
APENDICE C **– METODOLOGIA MSF – ESTRUCTURA DEL PROYECTO**
APENDICE D **– METODOLOGIA MSF – VISION Y ALCANCE DEL PROYECTO**
APENDICE E **– METODOLOGIA MSF – DOCUMENTO DE REQUERIMIENTOS**
APENDICE F **– METODOLOGIA MSF – DOCUMENTO DE ESPECIFICACIONES FUNCIONALES**
APENDICE G **– METODOLOGIA MSF – DOCUMENTO MAESTRO**
APENDICE H **– METODOLOGIA MSF – DOCUMENTO DE PLAN DE PRUEBAS**

En cada uno de estos apéndices se anexos los diferentes entregables obtenidos en cada fase de la metodología MSF.

CAPITULO II

CARACTERISTICAS DE LAS HERRAMIENTAS UTILIZADAS

1. ARQUITECTURA .NET

1.1. PROGRAMACION ORIENTADA A OBJETOS

La orientación a objetos

En la actualidad, el cómputo distribuido ocupa un lugar preponderante tanto en las ciencias de la computación como en la industria, debido a que muchos de los problemas a los que se enfrentan son inherentemente distribuidos. De la misma manera, las tecnologías orientadas a objetos se han consolidado como una de las herramientas más eficaces en el desarrollo de software, debido principalmente a su capacidad de describir los problemas en el dominio del problema, más que en el dominio de la solución.

La programación orientada a objetos trata de reflejar las características de los objetos reales, haciendo que un programa se divida en entidades que mantienen juntos los datos y el código que los manipula.

Dichas entidades reciben, como sus contrapartidas del mundo real, el nombre de objetos. Un objeto se forma de un conjunto de variables, que generalmente son accedidos mediante los denominados métodos (para variables privadas) que son funciones ligadas al objeto que permiten actuar sobre el estado de una variable. Este modo de disponer código y acceso de datos de se lo denomina encapsulación.

La encapsulación proporciona modularidad en su implementación, es decir, no afecta a ningún otro objeto del sistema al momento de implementar sus métodos. Los servicios que proporciona el objeto pueden ser utilizados sin saber cuáles son los detalles de su implementación (ocultamiento de la información).

Por otra parte, la abstracción de datos no es más que la definición de los datos a través de las operaciones que es posible realizar con ellos, en vez de por su representación en memoria.

La comunicación entre objetos se basa principalmente en la invocación de métodos que posee el objeto con los respectivos parámetros y cuyo resultado dependerá de la implementación del método para que efectue la acción solicitada.

En programación orientada a objetos, se denomina instanciación a la creación de un objeto a partir de una clase, y es posible referirse al objeto como una instancia de esa clase.

Una clase puede heredar parte de la funcionalidad de otra, para lo cual añade métodos a la clase original o los redefine para adaptarlos a una nueva necesidad. También se definen métodos que deliberadamente no se los implementan, de tal manera que las clases que heredan de ella deben definirlos e implementarlos. A estos métodos se los denomina abstractos o virtuales. Las clases que contienen métodos virtuales se llaman clases abstractas sin que se los pueda usar de forma directa, sino exclusivamente mediante la creación de clases que heredan de ellas y definan e implementen sus métodos.

Otra ventaja importante de la programación orientada a objetos es la facilidad en la reutilización de código para otras aplicaciones.

Los objetos son la representación en un lenguaje de programación de los conceptos que el ser humano es capaz de abstraer y generalizar. La programación orientada a objetos trata de amoldarse al modo de pensar del hombre, y no al de la máquina.

Clases y objetos

Una clase no es más que la base, modelo o plantilla para la creación de objetos. Antes de poder utilizar un objeto se debe indicar el tipo de objeto del cual estamos hablando, esto se consigue definiendo la clase de la cual el objeto que pretendemos manejar es un representante particular.

Si establecemos una analogía con el cerebro humano, una clase sería la representación abstracta de un concepto. Por tanto, una clase simplemente es una definición de un tipo de dato, por lo que no es directamente utilizable.

Atributos o propiedades

Los atributos son los datos específicos de una clase, en cuya definición únicamente se indica cuáles son los atributos y con qué nombre se va a designar a cada uno de ellos. Cada uno de los objetos de esa clase manejará un conjunto de atributos cuyos valores son independientes de los demás objetos de esa misma clase.

Los atributos pueden relacionarse al concepto de variable utilizado en otros lenguajes. Sin embargo, a diferencia de éstas, no tienen sentido fuera de la definición de una clase. Por este motivo, también suelen denominarse variables miembro de la clase, para indicar expresamente que se trata de variables contenidas en dicha clase; es decir sus datos son miembros integrantes de la misma.

Métodos

Al igual que los atributos, los métodos forman parte de la definición y son funciones propias de la misma. En general, la invocación a los métodos de un objeto producirá como resultado de su ejecución un dato de cierto tipo, que podrá recogerse y evaluarse en el punto donde se realice la llamada al método.

Herencia

El mecanismo de herencia permite definir nuevas clases partiendo de otras ya existentes. De este modo se consiguen reflejar todos los niveles de generalización que se deseen.

La reutilización de código

La reutilización de objetos se basa en la utilización de otros objetos ya definidos, proporcionando la capacidad de construir o adquirir una biblioteca de objetos desde la cual se componen los programas con mayor rapidez y facilidad. Esto evita el tener que empezar desde el principio con cada programa que se quiera desarrollar.

Encapsulamiento y ocultación de datos

La encapsulación es la combinación de datos y del código que manipula dichos datos en un solo componente llamado objeto. La encapsulación también hace referencia al control del acceso a los detalles de la implementación de un objeto. El acceso a un objeto queda limitado a una interfaz controlada y bien definida. Esto permite que los objetos sean autónomos y también los protege de una mala utilización accidental, dos aspectos muy importantes para un fiable diseño de software.

1.2. LA ARQUITECTURA .NET

.Net posee una arquitectura que agrupa componentes con funciones similares en capas lógicas, esto permite que el diseño y el desarrollo de aplicaciones sea más fácil de construir y modificar, aparte de que facilita la interconexión de tecnologías diferentes.

El siguiente gráfico muestra como se distribuye la arquitectura de aplicaciones .NET

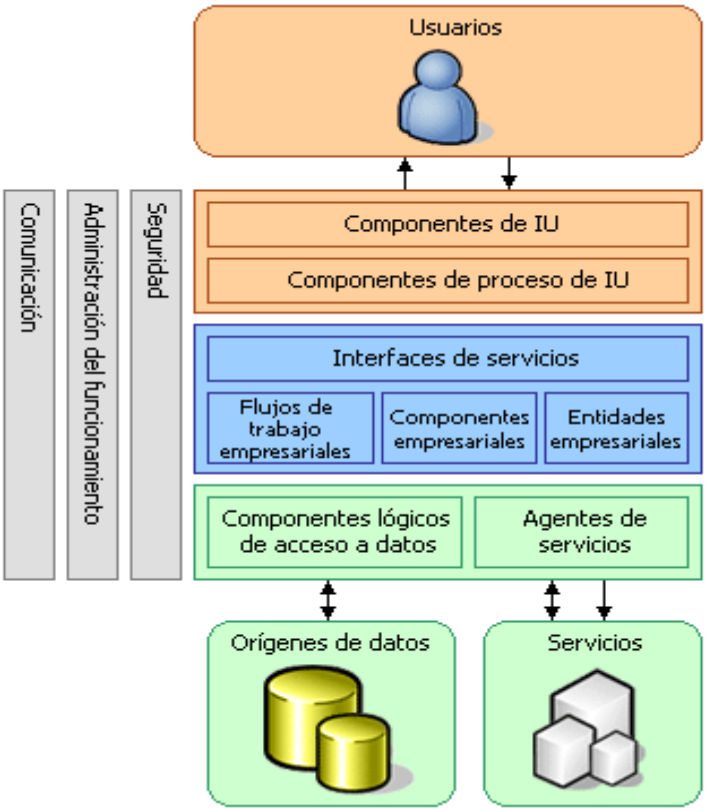


Gráfico 3. Arquitectura .NET

Capa de Presentación

La capa de presentación contiene los componentes necesarios para habilitar la interacción del usuario con la aplicación

- **Componentes de Interfaz de Usuario (UI)**

  Son el medio directo de interacción con el usuario. Se implementan utilizando formularios, páginas Web, controles u otro tipo de tecnología que permita validar los datos de entrada y salida al usuario.

- **Componentes de Proceso de Usuario (UI)**

Para aplicaciones más complejas los componentes de proceso encapsulan lógica de navegación y control de eventos de la interfase, este mecanismo permitiría que varias interfases de usuario utilicen el mismo “motor” de interacción básica.

Capa de Lógica de Negocio

Los servicios que proporciona esta capa se encapsulan en tres tipos de componentes:

- **Interfases de Servicios**

Para exponer lógica empresarial como un servicio, es necesario crear interfaces de servicios que admitan los contratos de comunicación (comunicación basada en mensajes, formatos, protocolos, seguridad y excepciones, entre otros) que requieren los clientes.

A estas interfases también se las conoce como “fachadas empresariales” y se implementan principalmente utilizando Web Services desarrollados con ASP.Net.

- **Flujos de Trabajo Empresariales (de negocio)**

Los flujos de trabajo empresariales definen y coordinan los procesos de negocios que están conformados de varios pasos de ejecución larga y se pueden implementar utilizando herramientas de administración de procesos empresariales, como BizTalk Server Orchestration.

- **Componentes Empresariales (de negocio)**

Encapsulan la lógica de negocio de la aplicación, se benefician de COM+ (Enterprise Services) para el manejo transaccional y manejo de escalabilidad.

Estos componentes son invocados por la capa de proceso de usuario, interfases de servicios y otros procesos de negocio. Además, llaman a los componentes lógicos de acceso a datos para recuperar y actualizar datos de la aplicación, así como también a servicios externos a través de agentes de servicios.

- **Entidades Empresariales (de Negocio)**

Las entidades empresariales o de Negocio, representan objetos que serán manejados o consumidos por toda la aplicación, estos pueden ser un modelo de objetos, xml, DataSets con tipo o estructuras de datos, que permitan representar objetos del mundo real que han sido identificados durante el modelamiento.

Los componentes de negocio no deben tener acceso directo a la base de datos, en su lugar, deben utilizar componentes lógicos de acceso a datos para realizar procedimientos funcionales con ellos a medida que se invocan sus métodos.

Las entidades empresariales no deben inicializar ningún tipo de transacciones ni utilizar API de acceso a datos; son meramente una representación de datos, potencialmente con comportamiento.

Capa de acceso a datos

La capa de acceso a datos contiene clases que interactúan con la base de datos. Estas clases surgen como una necesidad de mantener la cohesión o clases altamente especializadas que ayuden a reducir la dependencia entre las clases y capas. Aquí podemos encontrar también una clase con métodos estáticos que permiten uniformizar las operaciones de acceso a datos a través de un único conjunto de métodos.

- **Componentes Lógicos de Acceso a Datos**

Permiten centralizar la funcionalidad de acceso a datos y se facilita la configuración y el mantenimiento de la misma.

Los componentes lógicos de acceso a datos exponen métodos para insertar, eliminar, actualizar y recuperar datos, incluyendo la provisión de funcionalidad de paginación al recuperar grandes cantidades de datos.

Puede utilizar un componente de ayuda de acceso a datos para centralizar la administración de la conexión y todo el código relacionado con un origen de datos específico.

Puede utilizar un componente de ayuda de acceso a datos para centralizar la administración de la conexión y todo el código relacionado con un origen de datos específico.

Se recomienda implementar las consultas y operaciones de datos como procedimientos almacenados (si es compatible con el origen de datos) para mejorar el rendimiento y la facilidad de mantenimiento.

- **Agentes de Servicios**

Se utilizan cuando una aplicación requiere el uso de la funcionalidad proporcionada por un servicio externo. El agente de servicio encapsula el acceso a un servicio externo, permite aislar la implementación de los procesos empresariales de la implementación de formato de datos o cambios de esquema.

Componentes de seguridad, administración operativa y comunicación

Toda aplicación generalmente utiliza componentes para la administración de excepciones, autorizar a los usuarios a que realicen tareas determinadas y comunicarse con otros servicios y aplicaciones.

2. PROTOCOLO SNA Y TCP/IP (SOCKETS)

2.1. PROTOCOLO SNA (Systems Network Architecture)

En esta arquitectura se considera la existencia de varios tipos de nodos: terminales (T), controladores (C) (supervisan varios terminales), front-ends (FE) (descargan a los hosts de las labores de comunicaciones) y finalmente los ordenadores principales o hosts.

Arquitectura SNA

Los componentes del modelo SNA de IBM corresponden muy de cerca con el modelo de referencia OSI. Las descripciones que siguen delinean el rol de cada componente SNA que provee conectividad entre las entidades SNA.

- **Control de enlace de datos "Data link control" (DLC):** Define varios protocolos, incluyendo el protocolo Control de Enlace de Datos Sincrónicos "Synchronous Data Link Control (SDLC)" para comunicación jerárquica, y el protocolo de comunicación de redes Token Ring para comunicación LAN entre pares.
- **Control de ruta**: Realiza muchas de las funciones de la capa de red del modelo OSI, incluyendo el ruteo, la segmentación de datagramas y el reensamblado "segmentation and reassembly" (SAR)
- **Control de transmisión:** Provee un servicio de conexión confiable extremo a extremo, así como también servicios de encriptación y de desencriptado.
- **Control del flujo de datos:** Administra procesamiento de pedidos y respuestas, determina de quien es el turno de comunicarse, mensajes de grupo e interrumpe el flujo de datos bajo pedido.
- **Servicios de presentación:** Especifican algoritmos de transformación de datos que traducen datos desde un formato a otro, coordinan compartición de recursos y sincronizan la transacción de operaciones.
- **Servicios de transacción**: Provee servicios de aplicación en la forma de programas que implementan el procesamiento distribuido o servicios de administración.

Una pieza clave definida dentro del modelo de red SNA es la red de control de ruta, la cual es responsable de mover información entre nodos SNA y facilitar la comunicación entre nodos de diferentes redes. La red de control de ruta usa funciones provistas por el control de ruta y el control del enlace de datos "data link control (DLC)".

Entidades Físicas SNA

Las entidades físicas de la SNA tradicional asume una de las siguientes cuatro formas: hosts, controles de comunicaciones, controles de establecimiento y terminales. Los hosts en una SNA controlan toda o parte de una red y típicamente proveen cálculos, ejecución de programas, acceso a bases de datos, servicio de directorio, y administración de red. Los controladores de comunicaciones, también llamados procesadores de front-end "front end

proccessors" (FEPs), administran la red física y controlan los enlaces de comunicaciones.

Los controladores de establecimiento se los llama comúnmente controladores de grupo. Estos dispositivos controlan operaciones de entrada y salida de los dispositivos conectados, tales como las terminales. Las terminales, también conocidas como estaciones de trabajo, proveen la interfase de usuario hacia la red.

El siguiente gráfico ilustra cada uno de estas entidades físicas en el contexto de un diagrama generalizado de una red SNA.)

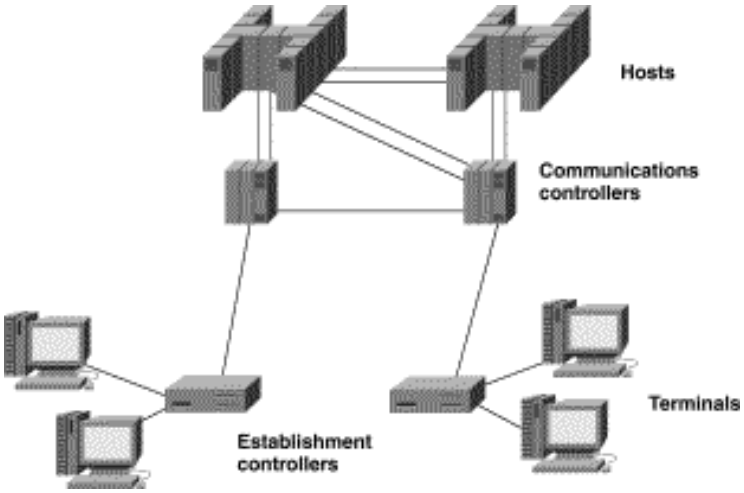


Gráfico 4. Entidades del SNA

Control de Enlace de Datos SNA

La capa de control de enlace de datos SNA "*SNA data link control (DLC)*" soporta medios tales como canales de mainframe, SDLC, X25, Token Ring, IEEE 802.3/Ethernet, FDDI y Frame Relay, cada uno de los cuales esta diseñado para proveer acceso a los dispositivos y usuarios con diferentes requerimientos.

Una conexión a un canal de mainframe SNA estándar provee un canal de datos paralelo que usa técnicas de movimiento de datos mediante memoria de acceso directo "direct memory access (DMA)". Un canal de mainframe estándar puede transferir datos a una velocidad de 3 a 4.5 Mbps.

La siguiente figura ilustra como los distintos medios calzan en la red SNA.

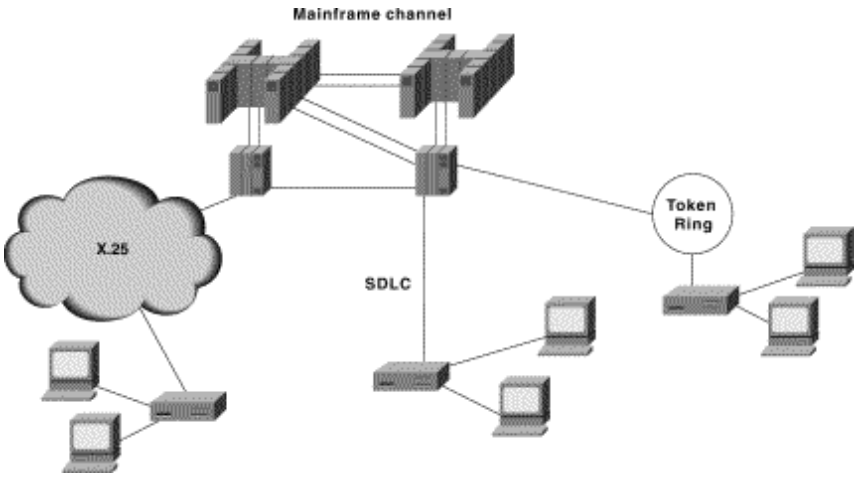


Gráfico 5. Compatibilidad de medios con SNA

Unidades direccionables de red SNA

SNA define tres unidades direccionables de red "network addressable units (NAUs)": unidades lógicas, unidades físicas y puntos de control. Cada uno juega un rol importante en el establecimiento de conexiones entre sistemas en una red SNA.

- ***Unidades Lógicas*** *"Logical units" (LUs)* funcionan como puertos de acceso a usuarios finales en una red SNA. Las LUs proveen a los usuarios acceso a recursos de red y administran la transmisión de información entre usuarios finales.

- ***Unidades físicas*** *"Physical units" (PUs)* son usadas para monitorear y controlar enlaces de red conectados y otros recursos de red asociados con un nodo particular. Las PUs son implementadas en hosts por métodos de acceso SNA, tales como el método virtual de acceso de telecomunicación "virtual telecommunication access method (VTAM)". Las PUs también son implementadas dentro de los controladores de comunicaciones por programas de control de red "network control programs (NCPs)."
- ***Puntos de Control*** *"Control points (CPs)"* administran nodos SNA y sus recursos. Los CPs generalmente se diferencian de las PUs en que los CPs determinan que acciones deben ser tomadas, mientras que los Pus causan que ocurran acciones.

Formato de la Unidad Básica de Información

Las NAUs SNA emplean las unidades básicas de información "basic information units (BIUs)" para intercambiar pedidos y respuestas. La siguiente figura ilustra el formato del BIU.

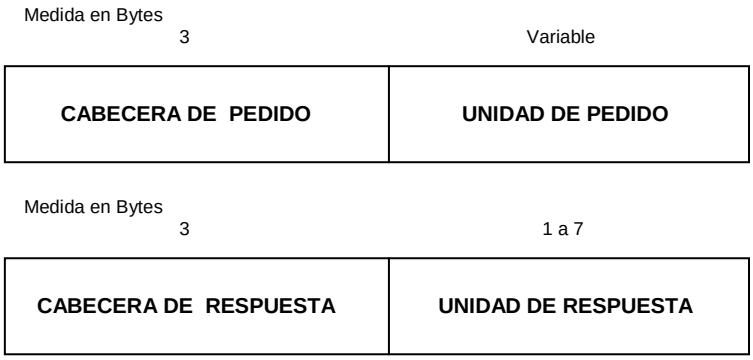


Gráfico 6. Unidades básicas de Información del SNA

Una unidad Básica de Información puede ser o un pedido o una respuesta.

- **Cabecera de pedido:** Identifica el tipo de datos en las unidades de pedido asociadas. Esta cabecera provee información acerca del formato de los datos y especifica protocolos para la sesión. Solo NAUs usan información de cabeceras de pedido.

- **Unidad de pedido:** Contiene ya sea datos de usuario Terminal o comandos SNA. Los datos de usuario Terminal se envían en unidades de pedidos de datos. Los comandos SNA se envían en unidades de pedido de comandos que controlan la información de red y contiene información intercambiada entre usuarios terminales.

- **Cabecera de respuesta:** Identifica el tipo de datos asociado con la unidad de respuesta. El bit indicador de pedido/respuesta distingue una cabecera de respuesta de una cabecera de pedido. Una NAU que recibe indica si la respuesta siendo devuelta al emisor solicitante es positiva o negativa estableciendo el bit identificador de tipo de respuesta “response type indicator (RTI)” en la cabecera de respuesta.

- **Unidad de respuesta:** Contiene información acerca del pedido indicando ya sea una respuesta positiva o negativa. Las respuestas positivas a pedidos de comandos usualmente contiene una unidad de respuesta de 1 a 3 bytes que identifica el comando de pedido. Las respuestas positivas a pedidos de datos contiene cabeceras de respuesta pero no unidades de respuesta. Las unidades de respuesta negativas son de 4 a 7 bytes de largo y siempre son devueltas con una respuesta negativa. Una NAU que recibe devuelve una respuesta negativa a la NAU solicitante bajo una de las tres condiciones:

    - Emisor viola el protocolo SNA
    - Receptor no entiende la transmisión
    - Condición inusual, tal como una falla de ruta, ocurre cuando una respuesta negativa es transmitida, los primero 4 bytes de una unidad de respuesta contiene datos que explican porque el pedido es inaceptable. La NAU receptora envía hasta 3 bytes adicionales que identifican el pedido rechazado.

Formato de la Unidad de información de ruta

La unidad de información de ruta “path information unit (PIU)” es una unidad de mensaje SNA formada por elementos de control de ruta, que se añade una cabecera de transmisión a un BIU. La siguiente figura ilustra el formato PIU.

Los pedidos y respuestas de unidades de información de ruta “Path Information Unit (PIU)” consisten de tres campos:

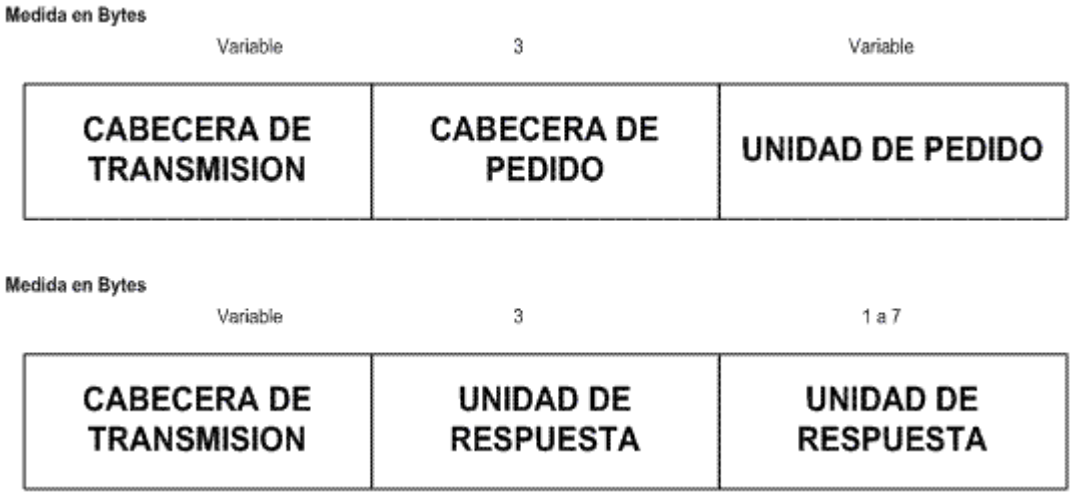


Gráfico 7. Rutas de unidades de información del SNA

- Cabecera de transmisión: Rutea unidades de mensaje a través de la red. Esta cabecera contiene información de ruteo para redes SNA tradicionales. Los formatos de cabecera de transmisión se diferencian por el tipo de identificación de formato “format identification (FID)”. El control de ruta utiliza los tipos de FID para rutear datos entre nodos SNA.

  Tres tipos de FID son implementados en PIUs:

  - FID0 es usada para rutear datos entre nodos de subarea adyacentes para dispositivos no SNA. FID0 generalmente es dejado obsoleto por el bit FID4 bit seteado para indicar si un dispositivo es SNA o no.
  - FID1 es usado para rutear datos entre nodos de subarea adyacentes cuando uno o ambos nodos no soportan protocolos de rutas explicitas y virtuales. FID2 es usado para rutear datos entre nodos de límite de subarea y un nodo periférico adyacente, o entre nodos adyacente tipo 2.1.

  En general, la cabecera de transmisión es usada para rutear datos entre nodos de subarea adyacente cuando ambos nodos de subarea soportan protocolos de ruteo explícitos y virtuales

2.2. PROTOCOLO TCP/IP

Una familia de Protocolos

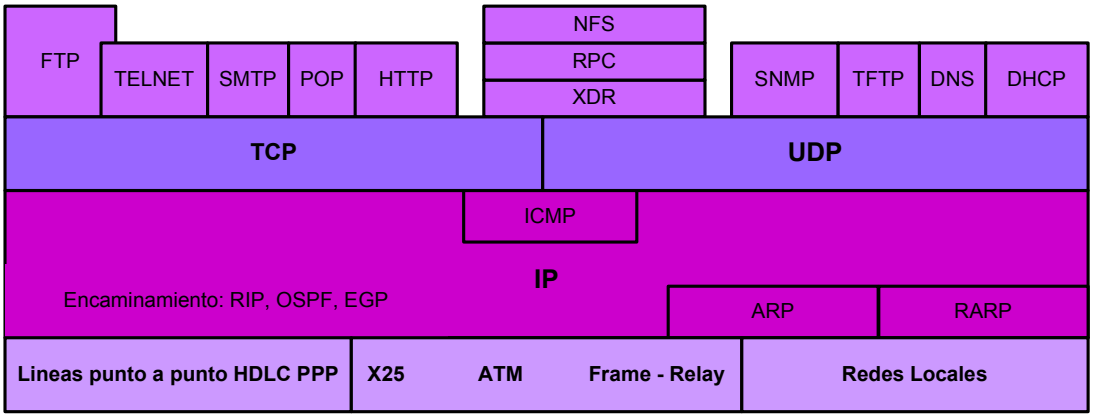


Gráfico 8. Familia de protocolos TCP/IP

Los protocolos que se utilizan en las comunicaciones son una serie de normas que deben aportar las siguientes funcionalidades:

- Permitir localizar un ordenador de forma inequívoca.
- Permitir realizar una conexión con otro ordenador.
- Permitir intercambiar información entre ordenadores de forma segura, independiente del tipo de maquinas que estén conectadas (PC, Mac, AS-400.).
- Abstraer a los usuarios de los enlaces utilizados (red telefónica, radio enlaces, satélite...) para el intercambio de información.
- Permitir liberar la conexión de forma ordenada.

Debido a la gran complejidad que conlleva la interconexión de ordenadores, se ha tenido que dividir todos los procesos necesarios para realizar las conexiones en diferentes niveles. Cada nivel se ha creado para dar una solución a un tipo de problema particular dentro de la conexión. Cada nivel tendrá asociado un protocolo, el cual entenderá todo lo que forme parte de la conexión.

Qué es TCP / IP

Cuando se habla de TCP/IP, se relaciona automáticamente como el protocolo sobre el que funciona la red Internet. Este nombre viene dado por los dos protocolos estrella de esta familia:

- El protocolo TCP, funciona en el nivel de transporte del modelo de referencia OSI, proporcionando un transporte fiable de datos.

- El protocolo IP, funciona en el nivel de red del modelo OSI, que nos permite encaminar nuestros datos hacia otras maquinas.

Pero un protocolo de comunicaciones debe solucionar una serie de problemas relacionados con la comunicación entre ordenadores, además de los que proporciona los protocolos TCP e IP.

Arquitectura de Protocolos TCP / IP

Para poder solucionar los problemas que van ligados a la comunicación de ordenadores dentro de la red Internet, se tienen que tener en cuenta una serie de particularidades sobre las que ha sido diseñada TCP/IP:

- Los programas de aplicación no tienen conocimiento del hardware que se utilizara para realizar la comunicación (módem, tarjeta de red...)
- La comunicación no esta orientada a la conexión de dos maquinas, eso quiere decir que cada paquete de información es independiente, y puede viajar por caminos diferentes entre dos maquinas.
- La interfaz de usuario debe ser independiente del sistema, así los programas no necesitan saber sobre que tipo de red trabajan.
- El uso de la red no impone ninguna topología en especial (distribución de los distintos ordenadores).

De esta forma, podremos decir, que dos redes están interconectadas, si hay una maquina común que pase información de una red a otra. Además, también podremos decir que una red Internet virtual realizara conexiones entre redes, que ha cambio de pertenecer a la gran red, colaboraran en el trafico de información procedente de una red cualquiera, que necesite de ella para acceder a una red remota. Todo esto independiente de las maquinas que implementen estas funciones, y de los sistemas operativos que estas utilicen.

Nivel de Red

El propósito de la capa de Internet es enviar mensajes desde el origen de cualquier red, y que estos mensajes lleguen a su destino, independientemente de la ruta y de las redes que se utilizaron para llegar hasta allí. El protocolo específico que rige esta capa se denomina Protocolo Internet (IP). En esta capa se produce la determinación de la mejor ruta. Esto se puede comparar con el sistema postal. Cuando envía una carta por correo, usted no sabe cómo llega a destino (existen varias rutas posibles); lo que le interesa es que la carta llegue.

Nivel de Enlace

Este nivel se limita a recibir datagramas del nivel superior (nivel de red) y transmitirlo al hardware de la red. Pueden usarse diversos protocolos: DLC (IEEE 802.2), Frame Relay, X.25, etc.

La interconexión de diferentes redes genera una red virtual en la que las maquinas se identifican mediante una dirección de red lógica. Sin embargo a la

hora de transmitir información por un medio físico se envía y se recibe información de direcciones físicas. Un diseño eficiente implica que una dirección lógica sea independiente de una dirección física, por lo tanto es necesario un mecanismo que relacione las direcciones lógicas con las direcciones físicas. De esta forma podremos cambiar nuestra dirección lógica IP conservando el mismo hardware, del mismo modo podremos cambiar una tarjeta de red, la cual contiene una dirección física, sin tener que cambiar nuestra dirección lógica IP.

El Protocolo IP

El protocolo IP (*Internet Protocol*) define la unidad básica de transmisión de datos cuando viajan por una red TCP/IP. Además, el protocolo IP incluye una serie de reglas que especifican como procesar los paquetes y como manejar los errores. El protocolo IP se basa en la idea de que los datos se transmiten con un mecanismo no fiable y sin conexión. Un mecanismo no fiable quiere decir que un paquete puede perderse, duplicarse o enviarse a un destino diferente del deseado. El mecanismo es sin conexión porque cada paquete se trata independientemente de los otros. Paquetes de una secuencia enviados de una máquina a otra pueden ir por distintos caminos, e incluso unos pueden alcanzar su destino mientras que otros no. El protocolo incluye también la idea del encaminamiento (*o routing*) de paquetes.

El datagrama IP

El datagrama es la unidad básica de transmisión de datos en la red Internet. El datagrama, al igual que las tramas en las redes físicas, se divide en encabezamiento y campo de datos. El encabezamiento contiene las direcciones IP de la fuente y el destino.

La longitud máxima de un datagrama es de 65.536 octetos (64 Kbytes). Sin embargo, para viajar de una máquina a otra, los datagramas lo hacen en el campo de datos de tramas de enlace, por lo tanto los datagramas de longitud excesiva deben ser divididos en fragmentos que quepan en las tramas. Cada fragmento perteneciente a un mismo datagrama tiene el mismo número de identificación que el datagrama original, con lo que es posible su reconstrucción.

El datagrama IP es encapsulado en la trama ethernet, como se muestra en la figura:

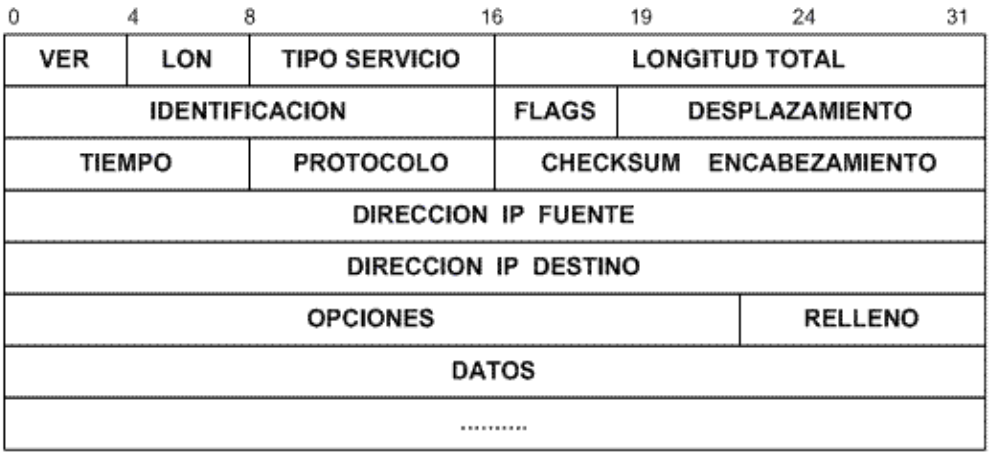


Gráfico 9. Datagrama IP

El Protocolo TCP

A bajo nivel, el protocolo IP proporciona un servicio de distribución no fiable de paquetes. Cuando la red física falla, los paquetes pueden perderse, llegar con errores, duplicados o fuera de secuencia.

Al nivel más alto, los programas de aplicación necesitan, frecuentemente, enviar grandes volúmenes de datos, por lo cual se utiliza el protocolo de transporte TCP, que permite la transmisión fiable de datos mediante el establecimiento y liberación de conexiones, de manera que los datos que llegan a los programas de aplicación lo hagan sin errores y ordenados, pudiendo ser utilizados directamente, sin necesidad de escribir algoritmos de detección y recuperación de errores.

TCP permite a varios programas de aplicación en una misma máquina comunicarse simultáneamente y demultiplexar el tráfico TCP que recibe entre las distintas aplicaciones. El protocolo TCP incorpora los denominados puertos, que identifican el último destino dentro de una máquina y el programa de aplicación que está haciendo usos de ese puerto. Cada puerto tiene asociado un entero para identificarlo. Dado que el número de puerto es único dentro de una máquina, el destino último para el tráfico TCP queda perfectamente determinado por la dirección IP del “host” y el número de puerto TCP en ese “host”.

El protocolo TCP combina una adjudicación de números de puertos TCP dinámica y estática, usando una serie de asignaciones de puertos conocida para un conjunto de programas que se utilizan comúnmente (por ejemplo, correo electrónico). Sin embargo, la mayoría de los números de puerto están disponibles para que el sistema operativo los utilice a medida que los programas de aplicación los necesiten.

Formato de segmento TCP

El segmento TCP es la unidad de transferencia de datos de este protocolo. Los segmentos se intercambian para establecer y liberar conexiones, transferir datos, enviar acuses de recibo e informar sobre tamaños de ventana. Un acuse de recibo que vaya de una máquina A a otra B puede viajar en el mismo segmento que lleve datos de la máquina A a la máquina B (piggybacking).

Los segmentos TCP viajan en la porción de datos de los datagramas IP, como se muestra en la figura:

Encabezamiento IP	Segmento TCP tratado como datos

El protocolo TCP entrega al servicio IP el segmento que contiene los datos, que es el que se transmite realmente, y una *pseudo-cabecera* que no se transmite, pero que permite al protocolo IP completar los datos del o los datagramas que va a generar. El formato del segmento y de la pseudo-cabecera son los que se muestran en la figura siguiente.

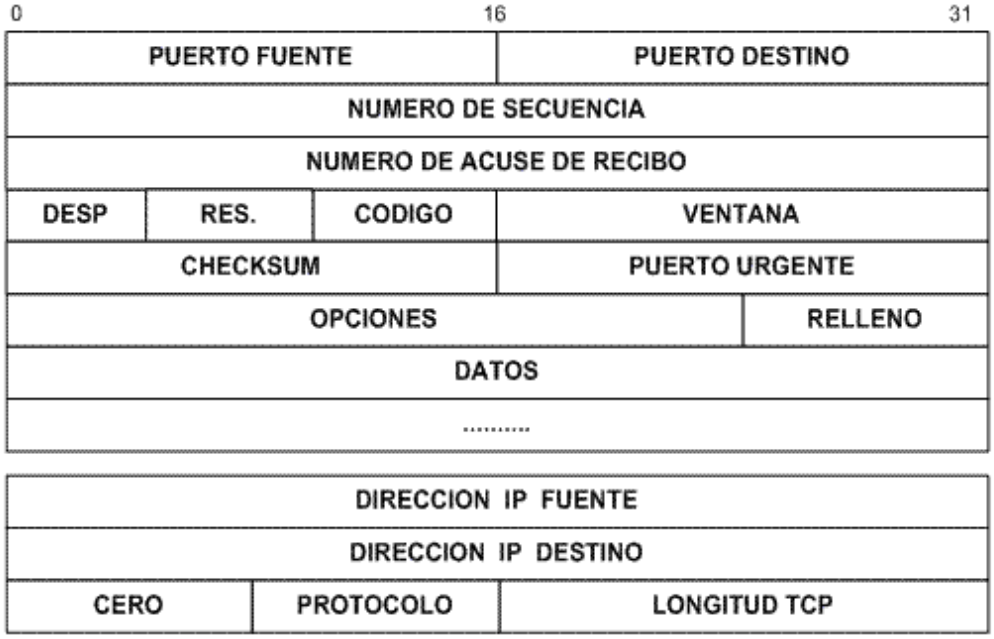


Gráfico 10. Segmento TCP.

2.3. SOCKETS

El socket es un punto de comunicación por el cual un proceso puede emitir o recibir información. En el interior de un proceso, un socket se identifica por un descriptor de la misma naturaleza que los que identifican los archivos, al igual que todos los procesos del sistema UNIX de Berkeley.

La comunicación mediante sockets es una interfaz (o servicio) con la capa de transporte (nivel 4) de la jerarquía OSI. La filosofía de la división por capas de un sistema es encapsular, dentro de cada una de ellas, detalles que conciernen sólo a cada capa, y presentársela al usuario de tal forma que este pueda trabajar con ella sin necesidad de conocer sus detalles de implementación. La interfaz de acceso a la capa de transporte del sistema UNIX de Berkeley no está totalmente aislada de las capas inferiores, por lo que a la hora de trabajar con sockets, es necesario conocer algunos detalles sobre esas capas. En concreto, a la hora de establecer una conexión mediante sockets, es necesario conocer la familia o dominio de la conexión, y el tipo de conexión.

La creación de un socket se realizará por la primitiva socket cuyo valor de vuelta es un descriptor sobre el cual es posible realizar operaciones de escritura y lectura. Un socket permite la comunicación en los dos sentidos (conexión full-dúplex).

A continuación se muestra una tabla descriptora de ficheros de un proceso.

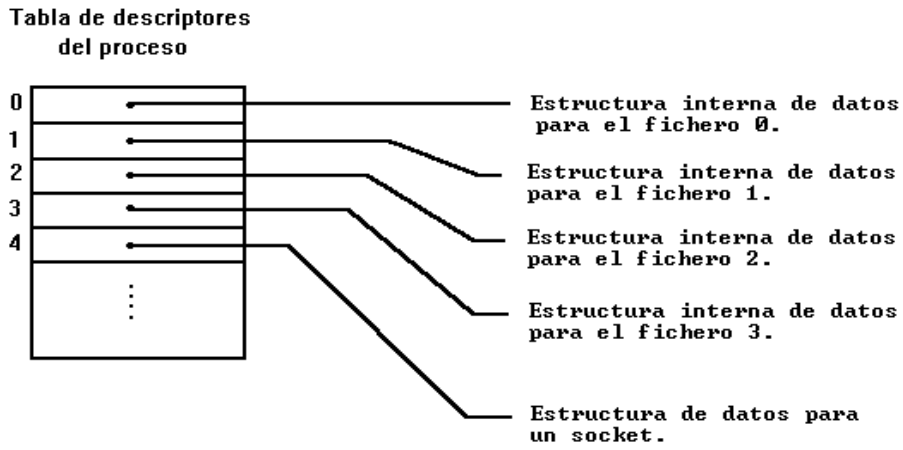


Gráfico 11. Descriptores de proceso (Sockets)

Una diferencia esencial con los archivos es que el nombrado de los sockets es una operación distinta de su creación (apertura y creación van a la par, pero después es posible fijar una dirección de su dominio al objeto creado por medio de la primitiva bind).

Dominio de un Socket

Una familia, o dominio de la conexión, agrupa todos aquellos sockets que comparten características comunes. Especifica el formato de las direcciones que se podrán dar al socket y los diferentes protocolos soportados por las comunicaciones vía los sockets de este dominio.

Cada protocolo, a la hora de referirse a un nodo de la red, implementa un mecanismo de direccionamiento. La dirección distingue de forma inequívoca a cada nodo u ordenador, y es utilizada para encaminar los datos desde el nodo origen hasta el nodo destino. Hay muchas llamadas al sistema que necesitan un puntero a una estructura de dirección de socket para trabajar. La siguiente estructura genérica, se utiliza para describir las diferentes primitivas.

struct sockaddr {

u_short sa_family; /*familia de sockets; se emplean constantes de la forma AF_xxx */

char sa_data[14]; /*14 bytes que contienen la dirección; su significado depende de la familia de sockets que se emplee */

};

Pero en el caso de que estemos tratando con una aplicación particular, esta estructura se deberá reemplazar por la estructura correspondiente del dominio de comunicaciones utilizado, ya que por desgracia, no todas las familias de direcciones se ajustan a la estructura genérica descrita.

No obstante, esta estructura genérica encaja en la estructura definida para la familia AF_INET, perteneciente al dominio Internet, lo que hace que el software de TCP/IP trabaje correctamente, puesto que ambas tienen el mismo número de bytes. A continuación, pasamos a describir esta estructura del dominio Internet.

struct sockaddr_in {

short sin_family; /* la familia de la dirección AF_INET*/

u_short sin_port; /* 16 bits con el número del puerto */

u_long sin_addr; /* 32 bits con la dirección Internet (identificación de la red y del host)*/

char sin _zero[8]; /* 8 bytes no usados */

}

Se puede observar que en la dirección Internet el campo sin_family es equivalente al campo sa_family de la dirección genérica y que los campos sin_port, sin_addr y sin_zero cumplen la misma función que el campo sa_addr.

También podemos ver el dominio UNIX (AF_UNIX), donde los sockets son locales al sistema en el cual han sido definidos. Permiten la comunicación interna de procesos, y su designación se realiza por medio de una referencia UNIX.

```
struct sockaddr_un {
        short sun_family;       /* dominio UNIX: AF_UNIX */
        char sun_data[108];     /* path */
};
```

Estas direcciones se corresponden en realidad con paths de ficheros, y su longitud (110 bytes) es superior a los 16 bytes que de forma estándar tienen las direcciones del resto de familias. Esto es posible debido a que esta familia se usa para comunicar procesos ejecutados bajo control de la misma máquina, no teniendo así que acceder a la red. Otros dominios son:

```
AF_NS           /* protocolos XEROX NS */
AF_CCITT        /* protocolos CCITT, protocolos X.25, etc. */
AF_SNA          /* IBM SNA */
AF_DECnet       /* DECnet */
```

Tipos de Sockets

Cada tipo de socket va a definir una serie de propiedades en función de las comunicaciones en las cuales está implicado:

a) La fiabilidad de la transmisión. Ningún dato transmitido se pierde.

b) La conservación del orden de los datos. Los datos llegan en el orden en el que han sido emitidos.

c) La no duplicación de datos. Sólo llega a destino un ejemplar de cada dato emitido.

d) La comunicación en modo conectado. Se establece una conexión entre dos puntos antes del principio de la comunicación (es decir, se establece un circuito virtual). A partir de entonces, una emisión desde un extremo está implícitamente destinada al otro extremo conectado.

e) La conservación de los límites de los mensajes. Los límites de los mensajes emitidos se pueden encontrar en el destino.

f) El envío de mensajes (urgentes). Corresponde a la posibilidad de enviar datos fuera del flujo normal, y por consecuencia accesibles inmediatamente (datos fuera de flujo).

Cabe reseñar que un cauce de comunicación normal tiene las cuatro primeras propiedades, pero no las dos últimas.

En cuanto a los tipos de sockets disponibles, se pueden considerar:

Sockets Stream

Son un servicio orientado a conexión donde los datos se transfieren sin encuadrarlos en registros o bloques. Si se rompe la conexión entre los procesos, éstos serán informados.

El protocolo de comunicaciones con streams es un protocolo orientado a conexión, ya que para establecer una comunicación utilizando el protocolo TCP, hay que establecer en primer lugar una conexión entre un par de sockets. Mientras uno de los sockets atiende peticiones de conexión (servidor), el otro solicita una conexión (cliente). Una vez que los dos sockets estén conectados, se pueden utilizar para transmitir datos en ambas direcciones.

Sockets Datagrama

Son un servicio de transporte sin conexión. Son más eficientes que TCP, pero no está garantizada la fiabilidad. Los datos se envían y reciben en paquetes, cuya entrega no está garantizada. Los paquetes pueden ser duplicados, perdidos o llegar en un orden diferente al que se envió.

El protocolo de comunicaciones con datagramas es un protocolo sin conexión, es decir, cada vez que se envíen datagramas es necesario enviar el descriptor del socket local y la dirección del socket que debe recibir el datagrama. Como se puede ver, hay que enviar datos adicionales cada vez que se realice una comunicación.

Sockets Raw

Son sockets que dan acceso directo a la capa de software de red subyacente o a protocolos de más bajo nivel. Se utilizan sobre todo para la depuración del código de los protocolos.

Sumario de primitivas

Hasta ahora, hemos visto las primitivas más importantes usadas con sockets. Pero hay muchas más, por lo que pasamos a hacer un pequeño esquema en el que se muestran todas las primitivas existentes usadas con TCP:

Primitiva	Función
socket	*Crea un descriptor para que sea usado en la comunicación.*
connect	*Conexión con un cliente remoto.*
write	*Manda datos a través de una conexión.*
read	*Lee los datos entrantes de una conexión.*
close	*Termina la conexión y elimina el descriptor.*
bind	*Vincula una dirección local IP y un puerto de protocolo a un socket.*
listen	*Pone el socket en modo pasivo y establece el número de conexiones TCP entrantes que el sistema puede encolar.*
accept	*Acepta la siguiente conexión entrante.*
recv	*Recibe el siguiente datagrama entrante.*
recvmsg	*Recibe el siguiente datagrama entrante (variación de recv).*
recvfrom	*Recibe el siguiente datagrama entrante y graba la dirección fuente.*
send	*Envía un datagrama.*
select	*Informa de sockets listos para escribir o leer de ellos.*
sendmsg	*Envía un datagrama (variación de send).*
sendto	*Envía un datagrama, usualmente a un dirección previamente grabada.*
shutdown	*Termina un conexión TCP en una o ambas direcciones.*
getpeername	*Después de la llegada de una conexión, obtiene la dirección completa de la máquina remota desde un socket.*
getsockopt	*Obtiene las opciones actuales de un socket.*
setsockopt	*Cambia la opción de un socket.*

El siguiente gráfico muestra el esquema general de interacción cliente - servidor:

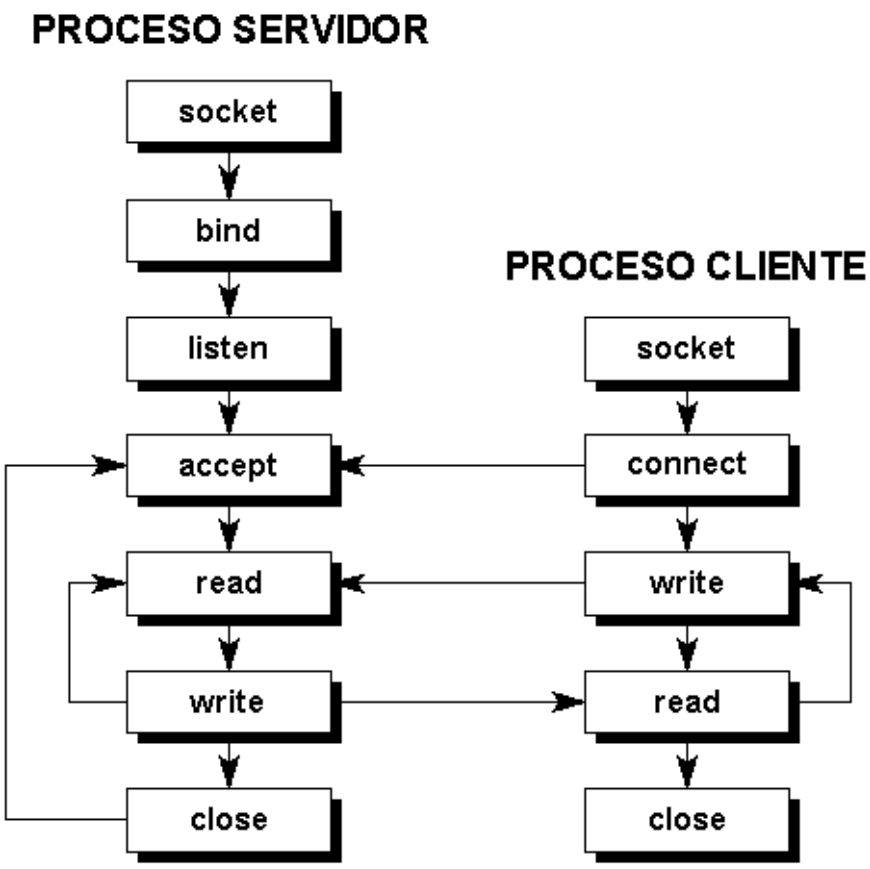


Gráfico 12. Interacción cliente-servidor con sockets

3. EQUIVALENCIA ENTRE TIPO DE DATOS COBOL Y C#.

Conceptos Básicos

La presente propuesta se liga a varias herramientas de programación como por ejemplo COBOL y .NET (C#), por lo que se hace necesario aclarar algunos conceptos referentes a cada herramienta:

3.1. Tipos de Datos Cobol

Como todo lenguaje de programación, Cobol define dos tipos de datos: Numérico y Alfanumérico con sus respectivas variaciones en cada uno, que definimos a continuación:

Cláusula USAGE

Su función es la de indicar como se representaran los datos en la memoria. Existen varios tipos, de los cuales describiremos lo siguientes:

- DISPLAY
- COMPUTATIONAL (COMP)
- COMPUTATIONAL-1 (COMP-1)
- COMPUTATIONAL-2 (COMP-2)
- COMPUTATIONAL-3 (COMP-3)
- INDEX

DISPLAY

Esta opción almacena en la memoria un octeto por cada carácter, por ejemplo:

```
01      CAMPO1      PIC X(6)  USAGE DISPLAY
01      CAMPO2      PIC 9,999.99
```

Se transfiere los valores “RICOSA” y 1.248,73 respectivamente; su representación en memoria sería

R	I	C	O	S	A
11011001	11001001	11000011	11010110	11100010	11000001

1	,	2	4	8	.	7	3
11110001	01101011	11110010	11110100	11111000	01001011	11110111	11110011

COMPUTACIONAL (COMP)

Este tipo de almacenamiento para datos numéricos, almacena en formato binario puro los números positivos y en formato de complemento A2 los números negativos.

En binario puro, el número de octetos que reserva el compilador en la memoria para almacenar cualquier número expresado en el sistema decimal es:

2 octetos para los números que contienen de 1 a 4 dígitos.

4 octetos para los números que contienen de 5 a 9 dígitos.

8 octetos para los números que contienen de 10 a 18 dígitos.

Esta forma de almacenamiento permite ahorrar memoria, por ejemplo, al poder colocar un numero de tres cifras en dos octetos, lo que en el caso anterior lo haría en tres.

A los campos numéricos en que se especifica USAGE COMPUTATIONAL o COMP se les llama campos BINARIOS.

El empleo más frecuente de esta opción, es en campos que van a servir de índices para la búsqueda en TABLAS.

COMPUTATIONAL-3 (COMP-3).

Un campo descrito con PICTURE numérica y con la opción COMPUTATIONAL-3, almacena dos dígitos en cada octeto (formato empaquetado), utilizando los cuatro bits del extremo derecho del campo para representar el signo del número: (1100 para los positivos, 1101 para los negativos y 1111 para los números sin signo), A estos campos se les denomina decimales INTERNOS.

COMPUTATIONAL-1, COMPUTATIONAL-2 (COMP-1, COMP-2).

Las opciones COMPUTATIONAL-1 y COMPUTATIONAL-2 se emplean cuando quieren almacenar números muy grandes o muy pequeños, es decir, números mayores de 5.4 x 10 exp-79 y menores de 72x10exp76.

Son también necesarias para realizar la potenciación en números con exponentes fraccionarios.

INDEX

Es usado para búsqueda en Tablas.

3.2. Tipos de Datos C#

C# como lenguaje .NET define una variedad de datos intrínsecos o nativos del lenguaje, que a continuación se describen:

Tipo	Valor	Tamaño (bits)
sbyte	Números enteros con signo	8
byte	Números enteros sin signo	8
short	Números enteros con signo	16
ushort	Números enteros sin signo	16
int	Números enteros con signo	32
uint	Números enteros sin signo	32
long	Números enteros con signo	64
ulong	Números enteros sin signo	64
float	Números en punto flotante	32
double	Números en punto flotante	64
decimal	Números con punto fijo	128
bool	true o false	
char	Un carácter Unicode	16
string	Cadena de caracteres	
object	Cualquier tipo de valor	

La presente propuesta utilizara las siguientes definiciones:

COBOL	**EQUIVALENCIA .NET**
X(10)	string
9(3)	int
9(8) COMP-3	long
9(12)V9(2)	double

4. COMTI

El Component Object Model (COM) introducido por Microsoft se ha convertido en una importante tecnología para desarrollos desde cliente/servidor con orientación a objetos y bajo redes Windows NT y superiores.

COM acorta el ciclo del desarrollo de aplicaciones reutilizando los objetos de componentes distribuidos en diferentes servidores.

Debido a tal crecimiento de la tecnología COM, Microsoft ha desarrollado varias herramientas útiles para integrar la tecnología COM con los productos y sistemas de tercera generación. Una de estas herramientas es el **COM Transaction Integration COMTI** para IBM CICS e Information Management System (IMS), pudiendo integrar de esta manera el Microsoft Transaction Server (MTS) con programas de Mainframe (Transaction Programs TPs); es decir, es capaz de ejecutar transacciones y programas que corren sobre CICS en IBM mainframes.

EL COMTI sería básicamente la respuesta a Instituciones cuyos sistemas son soportados por Mainframes; y que por exigencias tecnológicas en cuanto a renovar sus equipos y sistemas son inducidas a integrar nuevas tecnologías que incluyen sistemas clientes/servidor y Web bajo nuevos protocolos de comunicación.

Esto resultaría evidentemente muy costoso, razón por la cual, dichas Instituciones buscan Integrar de mejor manera las nuevas tecnologías con la que actualmente disponen.

En esencia, el COMTI proporciona una integración transparente entre el front end con el Mainframe (IBM OS/390), es decir, proporcionara una vía de comunicación entre el cliente y el Host. Comúnmente en Mainframe se utiliza herramientas de programación como es el caso de Cobol, AS 400, etc., estos lenguajes al igual que otros, necesitan datos de entrada y retorna datos de salida, es aquí, donde el COMTI actúa, facilitando o transportando dichos datos entre el front end y el Mainframe, además de ejecutar la transacción o programa respectivo mediante verbos APPC transportados con el protocolo SNA.

Esto facilita para conservar la lógica del negocio en el Mainframe, probablemente sin tener que modificar nada, y proporcionar interfases de usuario desarrolladas en plataformas modernas.

5. MIGRACIÓN DEL MVS CON CICS FOR OS/390 A HP-UX CON TX-SERIES.

Con el afán de contar con herramientas actuales de programación y a causa del creciente auge tecnológico; el Banco paulatinamente ha ido renovándose tecnológicamente tanto en hardware como en software.

La primera muestra de esto fue la adquisición de un sistema bancario extranjero denominado Siglo21 (Sistema de Información Global release 2.1), cuyas características eran las siguientes:

Características de hardware y software a migrar:

- S/390 Desarrollo: Partición LPAR (Logical Partitioning for iSeries) que contendrá los entornos de Desarrollo y Test-Qa.

- S/390 Producción: Partición LPAR que contendrá el entorno de producción.

- Cada entorno contiene su propia base de datos DB2 accesado por QMF (Query Management Facility) y el CICS (Customer Information Control System) comparte el ambiente de trabajo TSO (Time Sharying Options).

- Toda la transaccionalidad y lógica del negocio se ejecuta en un mainframe de IBM o compatible con el sistema operativo OS/390, estos son los requisitos básicos, que para una maquina de producción debe ser analizado de acuerdo al numero de clientes y transacciones que se realizan día a día en la institución:

  - 0.5 MIPS (million instructions per second) por persona que se conecte
  - 5 volúmenes 3390-3 (aproximadamente 15 Gb) de espacio en disco, para el soporte de la aplicación y una base de datos con una carga mínima.
  - Entre 256 Mb y 512 Mb de memoria real dependiendo de los subsistemas paralelos que se ejecuten.
  - Consta de un total de 35 Bases de datos diferenciadas por aplicación. Usa el motor DB2 en versión 5.1, ocupando un espacio aproximado de 4 volúmenes.
  - El monitor de transacciones es CICS/ESA (Enterprise System Architecture) en versión 4.10 o CICS Transaction Server en versión 1.3.
  - Las comunicaciones están soportadas por el subsistema VTAM (Virtual Telecommunication Access Method).
  - La seguridad se gestiona mediante RACF (Resource Access Control Facility) para garantizar el control adecuado a la aplicación.

- Los compiladores usados son propios de Cobol bajo MVS (Multiple Virtual Storage) a mas de pasar por un precompilador propio de la Aplicación Siglo21.

- La interfase con el usuario o Front End requiere de varias entidades para acoplarse con el mainframe, tales como red LAN, servidores, estaciones de trabajo, etc.; las caracteristicas se exponen a continuación:

  - Dos servidores para Desarrollo y Test los cuales contendrán el código fuente de la aplicación, sus ejecutables y demás aplicaciones de ofimática que use la aplicación.
  - Un servidor por cada Agencia para Producción utilizado como servidor de aplicación y servidor de comunicación.
  - Cada servidor tendrá su conexión dedicada al CICS correspondiente.
  - Las características básicas de cada servidor son: Pentium III a 500 Mhz con 256 Mb de RAM; la controladora de discos duros usará una interfaz Wide SCSI, y cada disco dispondrá de 25 Gb de capacidad.
  - El sistema operativo recomendado para los servidores es Windows NT 4.0 o superiores.
  - El software encargado de suministrar los servicios de comunicación entre los servidores y el mainframe es SNA Server (Microsoft) con el Service Pack 3.0 para SNA Server, mediante protocolo SDLC.
  - El protocolo de comunicación entre las estaciones de trabajo y los servidores es TCP/IP.
  - Las características básicas de una estación de trabajo son: Pentium II en adelante, 500 Mhz con 128 Mb de RAM y el espacio en disco no es prioridad, pero actualmente se puede tener discos con gran volumen de información para Pentium II.

Estas constituyen las principales características del hardware y software adquirido y en etapa de implementación; sin embargo, debido al tamaño de la institución, estas características constituyen un costo sumamente elevado tomando en cuenta el propósito de ser de un Banco.

Debido a la necesidad imperiosa de implementar Siglo21, el área de Tecnología del Banco se sumerge en una serie de investigaciones con el fin específico de implementar Siglo21 con herramientas de más fácil acceso económico y con rendimientos iguales o superiores a los descritos por los fabricantes de la aplicación Siglo21 con respecto al hardware y software usado para implementarlo.

Después de un año de investigaciones, pruebas de concepto, de carga, etc. se llega a la conclusión que Siglo21 puede ser montado en Plataformas Unix (HP-UX) con TX-Series como monitor transaccional, reemplazando de esta manera al MVS y todo el equipo mainframe de IBM, provocando un ahorro abismal.

Proceso de migración

Después de obtenido todos los pormenores necesarios para llevar a cabo una migración masiva de Siglo21 bajo plataformas y sistemas operativos diferentes a los recomendados por los fabricantes, se inicia formalmente dicha migración con las siguientes características:

Alcance del proceso de migración

Las tareas a desarrollar fueron las siguientes:

- Instalación de software requerido en Servidores HP-UX
- Compilación de programas COBOL en la nueva plataforma.
- Desarrollo de programas precompiladores COBOL
- Desarrollo de programas precompiladores ORACLE 9i.
- Ejecución de programas de conversión de datos desde DB2 a ORACLE.
- Carga de las especificaciones de ejecución batch (equivalente al Job Control Language) en la nueva plataforma.
- Prueba de las aplicaciones en la nueva plataforma.
- Modificación de programas de comunicación entre el Servidor SNA y el UNIX (convertidores de ASCII a EBCDIC ya no es necesario para la nueva plataforma) .
- Capacitación soporte y migración transparente hacia la nueva plataforma.

Todas las tareas especificadas dieron como resultado una migración e implementación transparente para el usuario final, y proporcionó estándares a seguir para el desarrollo de posteriores módulos dentro de lo que ahora es Siglo21.

Herramientas de Migración.

A continuación se describe las características de hardware y software usados en la migración:

- TXSeries versión 4.2 que substituye al MVS/ESA y proporciona opciones de servidor y gateway, siendo una evolución de aplicaciones existentes como Transaction Server, CICS y Encina.
- MF Cobol (Microfocus Cobol Server Express), soporta alto volumen de aplicaciones transaccionales con multiples opciones de acceso a datos, compiladores y debugger, substituye el Cobol para mainframe antes utilizado.
- Oracle Database Enterprise Edition que reemplaza al DB2 para mainframe
- El sistema operativo es HP-UX versión 11i para servidores RISC HP.
- Servidores UNIX, para Producción denominado Superdome 64 way configurado en dos particiones, en la cual la partición de producción consta de 16 procesadores en modalidad iCOD del tipo PA-RISC 8700+ de 875 MHz. y 64 GB de memoria. La partición de contingencia denominada FailOver cuenta con 16 procesadores activos del mismo tipo de la partición de producción y 32 GB de memoria.

Ambas particiones poseen redundancia a nivel de conectividad con la red de usuarios, con la solución de almacenamiento propuesto y con los dispositivos de disco interno utilizados para el arranque del sistema, manejo del sistema operativo y área de swap (cada unidad DS2300 consta de 8 discos de 36 GB cada uno, para un total de 288 GB disponibles).

CAPITULO III

CONSTRUCCIÓN DE LA HERRAMIENTA ESTANDAR DE COMUNICACION WEB-OS/390 Y DEFINICIÓN DE LA METODOLOGÍA BASE A UTILIZAR

Antes de empezar a describir principalmente el diseño de la librería de comunicación, es importante hacer un breve análisis de la situación actual, y asi mismo una breve introducción a las características de la propuesta planteada:

Análisis del sistema actual

El sistema actual instalado en la empresa caso de estudio, dispone de varios componentes que lo conforman tanto en hardware como en software; conforme avancemos en el análisis inicial, describiremos cada uno de ellos.

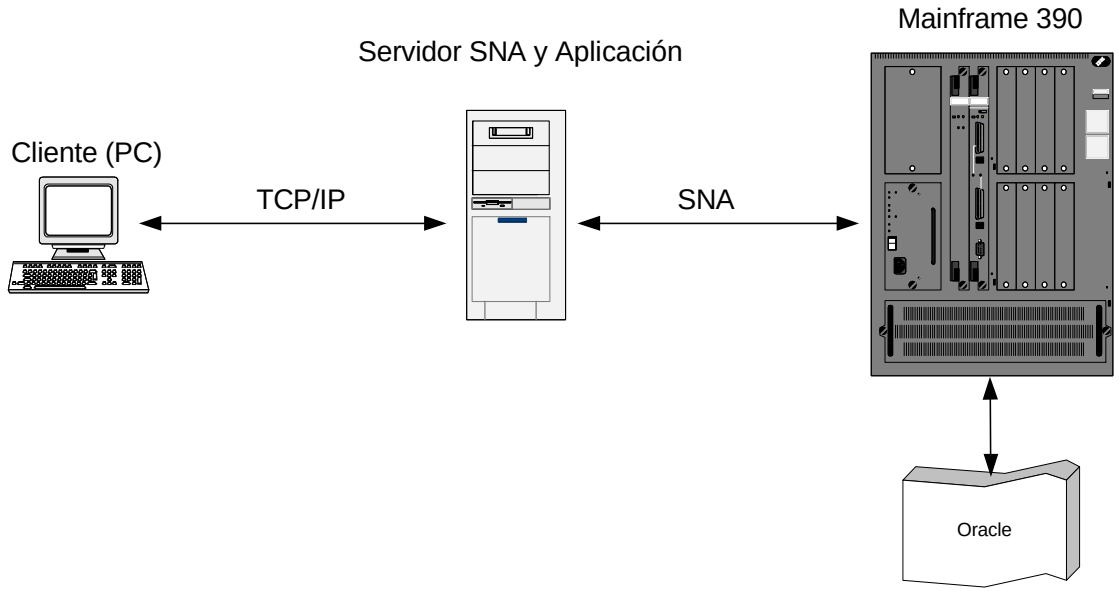


Gráfico 13. Componentes del Sistema Actual.

El Gráfico 13 muestra los componentes en cuanto a la infraestructura que forman parte de la aplicación; a continuación describiremos con mayor detalle cada uno de estos componentes.

Cliente

Se lo define como Cliente al ordenador desde el cual el usuario final solicitará información al sistema; su característica principal es la memoria principal, pues requiere de por lo menos 128 Mb de RAM.

Su función es la de levantar la máquina virtual de Java (1.1.8) para que el procesador ejecute los .class residentes en el servidor de aplicación para poder interactuar con el usuario final, quien constantemente solicitara requerimientos al servidor.

El protocolo de comunicación entre el cliente y el servidor es TCP/IP, mediante el uso de sokets; mas adelante se explicará el proceso que realiza tanto el cliente como el servidor para establecer comunicación.

Servidor

En este esquema, al Servidor se lo utiliza como servidor de aplicación en el cual residirá todo el software de la aplicación como tal y el software para que ejecute dicha aplicación; y como servidor de comunicación en el que se instalará el SNA y los protocolos necesarios (DLC) para establecer comunicación con el Mainframe 390.

El servidor será el encargado de proporcionar los ejecutables de la aplicación a las maquinas clientes, y al mismo tiempo servirá de gateway entre el cliente y el Mainframe, transportando la información entre estos dos componentes de hardware.

Mainframe

Es una máquina de IBM compatible con el sistema operativo OS/390; tiene una gran capacidad de procesamiento, y en el cual reside todos los servicios o programas Cobol que interactúan con la base de datos Oracle para ejecutar lógica definida por el negocio.

Este elemento, se encargará de enviar y recibir los datos procesados hacia el servidor SNA.

En capítulos posteriores se dará mas detalle de las características de estos elementos.

A más de los elementos arriba descritos, a continuación se describirá los elementos de software que conforma el sistema instalado en la empresa caso de estudio.

El siguiente gráfico muestra en general los principales componentes de software usado para la operatividad del sistema.

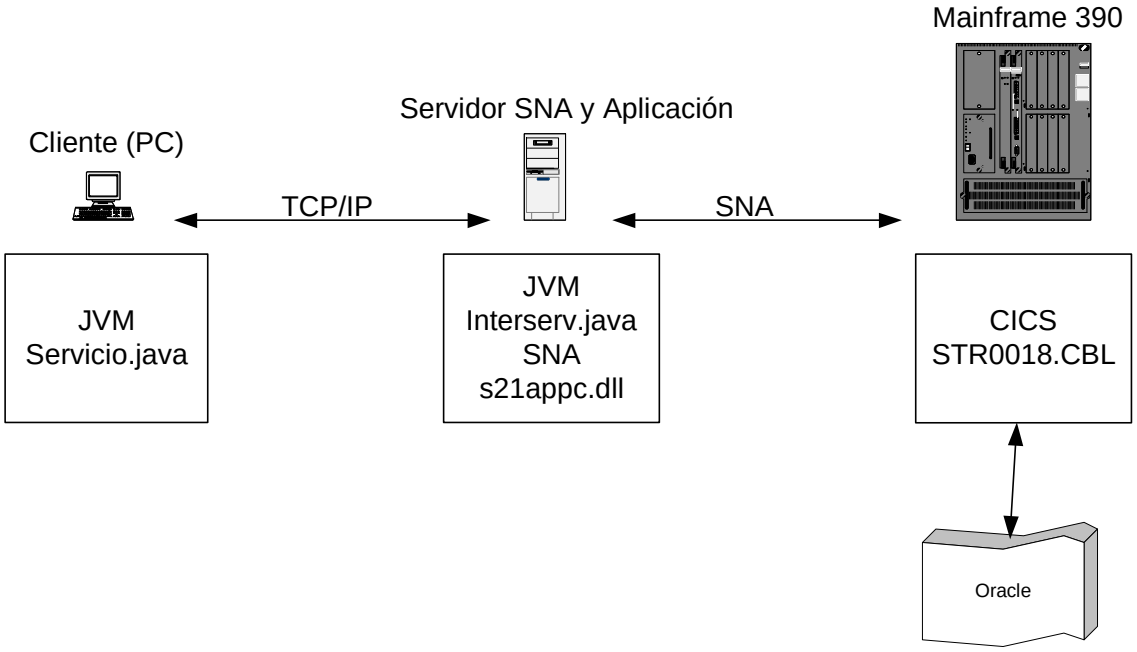


Gráfico 14. Componentes de Software del Sistema Actual.

En el grafico de componentes de software se aprecia las principales piezas que forman parte del sistema actual; a continuación se describirá el funcionamiento de cada componente de software dentro de su respectivo componente de hardware.

Como se indicó anteriormente, el cliente procesará los ejecutables que le proporciona el servidor de aplicación; para esto necesita antes levantar el Java Virtual Machine para poder luego ejecutar las clases proporcionadas por el servidor.

Una ves levantada el JVM y al mismo tiempo la aplicación, el usuario final necesitará entonces interactuar y procesar los datos que enviará y recibirá del Mainframe; para esto, en el lado del cliente existe una clase denominada Servicio, la misma que se encarga de enviar y recibir tramas de datos en bytes mediante sockets hacia el servidor; este por su parte tiene una clase Servidor denominada Interserv la misma que proporciona un multiprocesamiento para atender simultánea y concurrentemente a los clientes que lo invoquen, escuchando en el puerto 10500, esto lo hace mediante theads y sockets de java.

Cada petición que llega desde el cliente, es procesado por un hilo diferente de la clase Interserv, este hilo interactúa con una dll hecha en C++ denominada S21appc.dll, la misma que mediante verbos APPC de SNA capturará sesiones LU 6.2 y transmitirá la información al Mainframe.

El Mainframe por su parte dispondrá de definiciones de transacciones y programas en el CICS, los mismos que serán llamados por un programa cobol

denominado STR0018, quien se ata a una secuencia de transacciones de acuerdo a la funcionalidad de cada módulo de la aplicación.

El programa STR0018 es considerado como un programa “distribuidor”, debido a que a este programa le llegará una cabecera y un cuerpo de mensaje; en la cabecera, uno de los campos es el nombre del programa cobol que será ejecutado; entonces el STR0018 validará la cabecera de la transacción, y si todo está correcto, realiza un call dinámico al programa que se indica en la cabecera, enviándole como parámetros la información de la cabecera considerados como datos de entorno y a continuación el cuerpo mismo de la trama.

El programa cobol ejecutado generalmente contiene en su procedimiento sentencias SQL; es decir, estos programas son precompilados para aceptar sentencias SQL; por lo tanto, interactúan con Oracle para procesar datos en las tablas respectivas; los mismos que son procesados de acuerdo a la lógica establecida por el negocio y retornados al programa STR0018 para que los devuelva al servidor SNA y este a su vez al cliente.

Generalidades de la propuesta

Uno de los objetivos de la empresa caso de estudio, es el de migrar del esquema descrito anteriormente a un esquema WEB tratando en lo posible que los cambios únicamente se concentren en el front end; es decir, se reemplazará el front end de java por un font en hecho en web y codificado en C# de .NET.

Una de las definiciones que falta en la empresa caso de estudio para continuar con este objetivo es precisamente como será la comunicación entre los servidores WEB y el Mainframe; y una de las opciones es mediante objetos COMTIs; que se los verá mas adelante.

Es por esto, que el presente trabajo consolida una propuesta adicional en cuanto a la comunicación entre servidores WEB y el Mainframe, para posteriormente ser discutida e implementada.

Esta propuesta, parte específicamente de los recursos que dispone actualmente la empresa caso de estudio; es decir, se trabajará exclusivamente con lo que se tiene del sistema actual para luego ser plasmado en un ambiente WEB,, consiguiendo llegar a un esquema parecido al del siguiente gráfico:

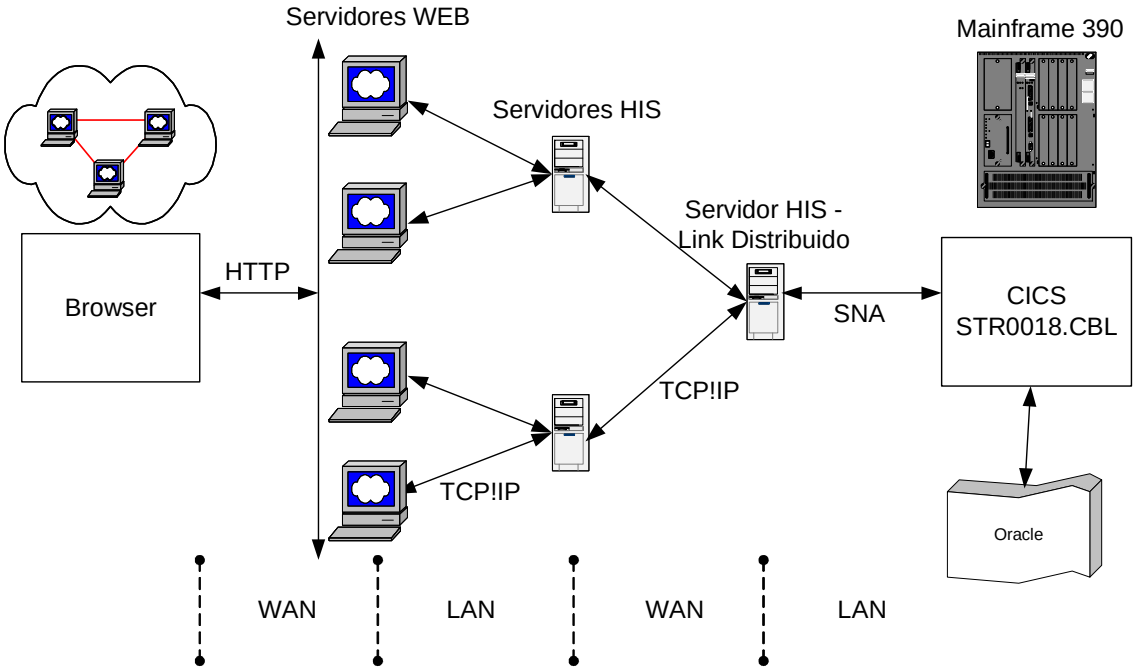


Gráfico 15. Componentes del Sistema propuesto.

Como se ve en el gráfico 15, la diferencia con el sistema actual es básicamente los componentes WEB introducidos; el resto de componentes difieren en infraestructura de equipos, pero la esencia es la misma que la del sistema actual.

El presente trabajo se centra en el componente de comunicación que de acuerdo al gráfico 3 está entre los servidores WEB y los servidores HIS.

Vale la pena mencionar que el diseño de la presente propuesta, se basa, como se ha dicho anteriormente, en los recursos existentes; por tal motivo se tiene las siguientes consideraciones de diseño para esta propuesta:

1. El componente de comunicación propuesto, tomará como base las definiciones del negocio implementado en el sistema actual.
2. El componente de comunicación trabajará con tramas de datos fijas y variables, lo cual no dispone el sistema actual.
3. Cada trama que pase por el componente de comunicación tendrá una parte de cabecera y otra de cuerpo mismo de la trama; la cabecera será idéntica a la del sistema actual conteniendo los siguientes campos:

    - Versión, de dos caracteres, indica la versión de los programas que se ejecutarán
    - Usuario, de ocho caracteres, indica el usuario conectado al sistema; este dato también es usado para auditoria.
    - Dominio, de cuatro caracteres, indica desde que terminal se realizo una determinada transacción; este dato es complemento del Nodo y viceversa.

- Nodo, de cuatro caracteres, indica desde que terminal se realizo una determinada transacción; este dato es complemento del Dominio y viceversa.
- Programa, de ocho caracteres, indica que programa invocar en el Mainframe.
- Longitud de entrada, de 6 caracteres, indica la longitud de la trama de datos que será enviada desde el front end.
- Longitud de salida, de 6 caracteres, indica la longitud de la trama de datos de salida que esperará el front end.
- Código de empresa, de cuatro caracteres, indica con que empresa se esta ejecutando la transacción.
- Código de centro, de cuatro caracteres, indica con que centro de la empresa se esta ejecutando la transacción.
- Entorno, de dos caracteres, indica en que entorno se ejecutará la transacción, puede ser Desarrollo (DA), Test (PE), Explotación (EX).
- Idioma, de un carácter, indica el idioma por defecto con el que se está ejecutando la transacción.
- CRC del programa, de 8 caracteres, indica si el CRC calculado de la trama de datos desde el front end es correcto o no.
- Autorización, de un carácter, indica si la transacción requiere de autorización o no.

4. El cuerpo del mensaje dependerá exclusivamente del diseño del programa o servicio cobol, es decir, dependerá de la lógica o funcionalidad que el negocio haya proporcionado para luego plasmarlo en el programa cobol.
5. Se creará una librería de comunicación que use la estructura actual de comunicación en cuanto a cabeceras y cuerpos del mensaje; esta librería será capaz de entender tipo de datos cobol y C#, traduciéndolos de bytes a objetos y viceversa.

Estas definiciones se detallan con mayor presición en la sección de Anexos donde está el desarrollo de la metodología MSF para la presente propuesta correspondiénte a los siguientes apéndices:

APENDICE B – METODOLOGIA MSF – DOCUMENTO DE ANTEPROYECTO
APENDICE C – METODOLOGIA MSF – ESTRUCTURA DEL PROYECTO
APENDICE D – METODOLOGIA MSF – VISION Y ALCANCE DEL PROYECTO

1. *DISEÑO DE LIBRERIAS Y CLASES DE COMUNICACIÓN*

Se ha tomado como base la arquitectura de componentes que usa .Net, es decir, se ha implementado el uso de capas para cada función específica del proyecto, con lo cual se tiene básicamente las siguientes capas:

- Capa de Presentación.
- Capa de Negocio
- Capa de Comunicación o persistencia.

Gráficamente se tiene la siguiente estructura, aunque el presente proyecto se enfoca a la capa de acceso a datos:

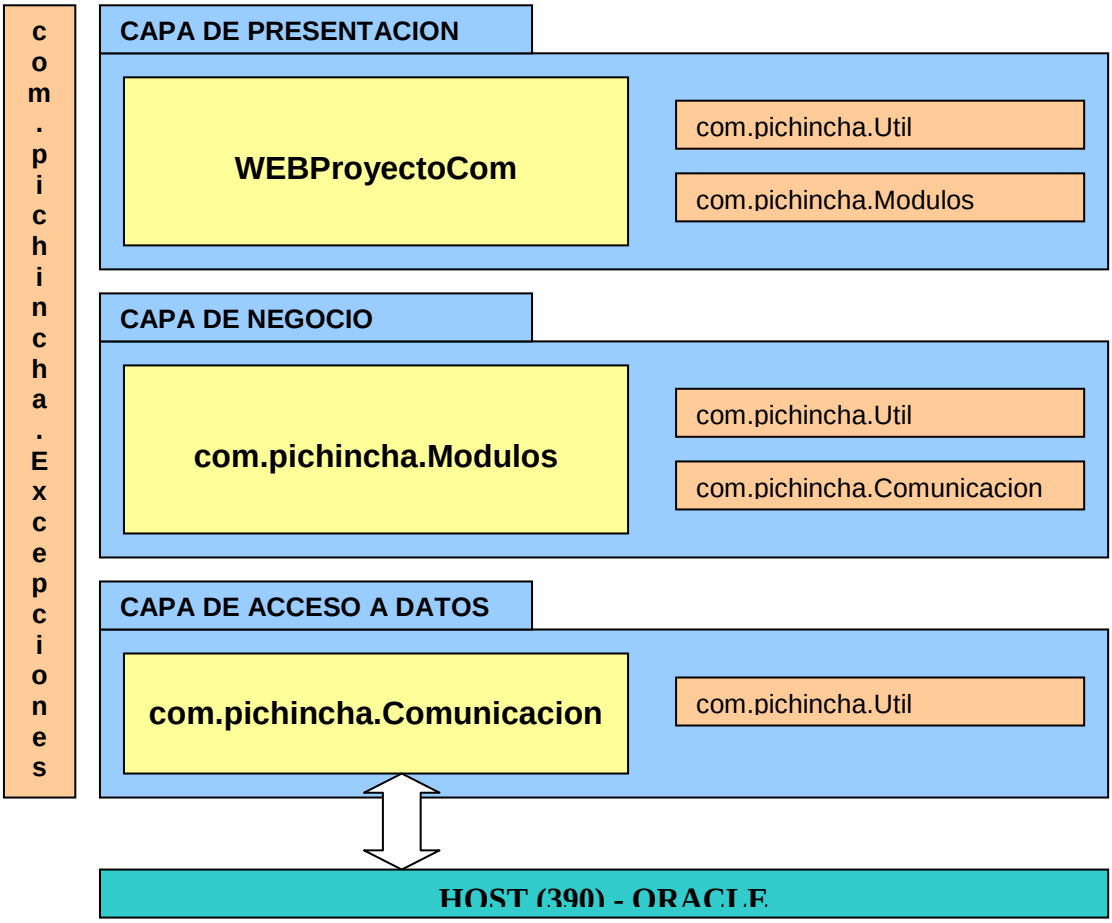


Gráfico 16. Componentes del Sistema propuesto.

1.1. Capa de Presentación

En esta capa ira todos los proyectos o aplicaciones Web. Básicamente los proyectos a aplicaciones dentro de esta capa se organizaran de acuerdo a la funcionalidad de cada modulo; por ejemplo, para el modulo de Pasivos,

se tendrá una estructura similar a: WEBProyectoCom/Pasivos/Principal.aspx.

Esta capa exclusivamente hará referencia a la capa de Negocio; no podrá hacerlo directamente con la capa de Comunicación.

1.2. Capa de Negocio

Contiene todas las clases en las que se deba implementar lógica de negocio. Esta capa es la única encargada de referenciar a la capa de comunicación, es decir, se encargara de enviar, recibir y procesar los datos del Host.

Igual que la capa de presentación, esta capa se dividirá en módulos de acuerdo a la funcionalidad que se tenga. Cada uno de estos módulos se organizará de la siguiente manera:

- **Modulo**
  - **datos**
  - **negocio**
  - **servicios**

Donde:

- **Modulo** significa el nombre de la funcionalidad que representa, por ejemplo, Activo, Pasivo, Contabilidad, etc.
- **datos** corresponde a clases con estado en las que se almacenaran lo que se interactúe con la capa de persistencia
- **negocio** corresponde a las clases que invocara a los servicios host mediante la capa de comunicación.
- **servicios** corresponde a las clases que contiene datos y heredan de las clases de comunicación y servirán de agentes de traslación de datos entre el Front y el Host.

1.3. Capa de Comunicación o acceso a datos.

Será la encargada de establecer comunicación mediante sockets hacia los servidores HIS (Host Integration Server) quienes a su vez se comunicaran con el Host mediante SNA.

A mas de establecer la vía de comunicación, mapeara los datos serializados que intercambia el .Net con el OS/390 a cadenas de bytes con objetos .Net y viceversa.

Esta capas tendrán como respaldo los siguiente componentes: com.pichincha.Util y com.pichincha.Excepciones, quienes se encargaran respectivamente de proporcionar utilidades básicas como lectura de

ficheros, constructores de ArrayList, etc. y de manejar las excepciones que se den en cualquiera de las capas.

En la capa de comunicación se tiene varias clases, todas igual de importantes pero las que sobresalen son las siguientes:

- **DatoRegistro.cs**

  Es una clase Abstracta que define métodos para facilitarle a otras clases el poder convertir bytes a objetos y viceversa. Además implementa un método para instanciar en tiempo de ejecución un arreglo de objetos.

  Esta clase será la clase padre del cual heredarán las clases cuyo objetivo es la de almacenar los datos de entrada y salida hacia el Host.

- **GestorHost.cs**

  Es una clase Abstracta con estado cuya principal función es la de comunicarse con el Host, y la de convertir objetos a una cadena serializada de bytes y viceversa. Implementa en su código sentencias de comunicación con sockets, abrirá enlace con los servidores HIS en el puerto 10500 para enviar y recibir la mensajería de cada programa.

  Esta clase es la más importante, pues es la clase de la cual extenderan todos los servicios ubicados en la clase de negocio. En esta clase se organizará la cabecera y el cuerpo del mensaje para luego ser enviada a los servidores HIS.

  Define dos métodos de tipo DatoRegistro para acceder a los tipos de datos de las clases que almacenarán los datos de entrada y salida hacia el Host.

También se maneja otras clases cuya funcionalidad es la de hacer compatible los tipos de datos que maneja Cobol y .Net; es decir, estas clases serán los tipos de datos finales para la capa de Negocio, y encapsularan internamente los tipos de datos primitivos de .Net. Estas clases son las siguientes:

S21String, maneja la compatibilidad entre datos de tipo PIC X(?) y string.
S21Integer, maneja la compatibilidad entre datos de tipo PIC 9(1 a 9) con integer.
S21Long, maneja la compatibilidad entre datos de tipo PIC 9(10 a 15) con long.
S21BigDecimal, maneja la compatibilidad entre datos de tipo PIC 9(?) V 9(?) con double.
S21Boolean, maneja la compatibilidad entre datos de tipo “booleano” en cobol con bool.

A continuación se expone el diseño lógico de la propuesta. Cabe destacar que este diseño se lo realizó en la etapa de Visión de la metodología MSF, referirse al "APENDICE E – METODOLOGIA MSF – DOCUMENTO DE REQUERIMIENTOS" de este documento:

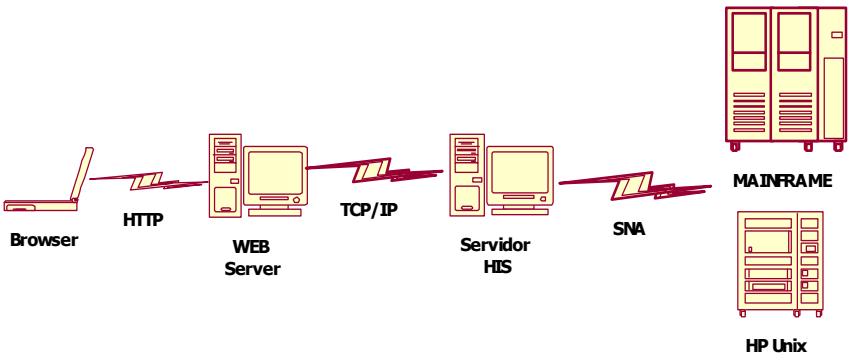


Gráfico 17. Componentes del Sistema propuesto.

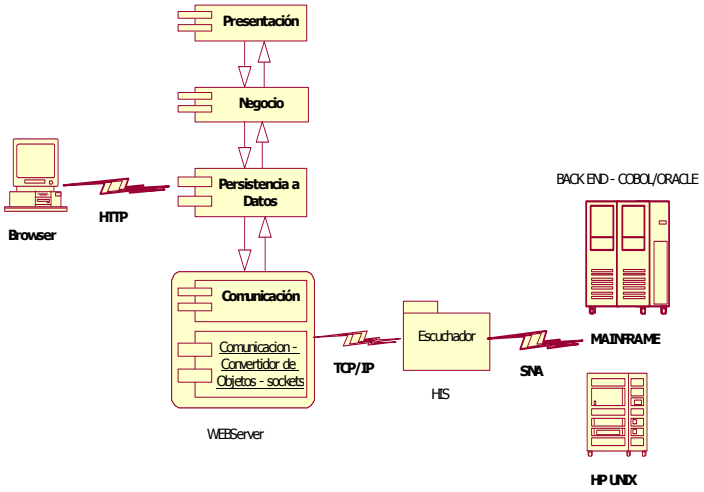


Gráfico 18. Diagrama Lógico de Componentes.

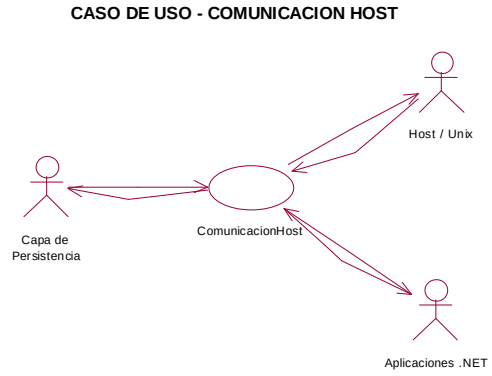


Gráfico 19. Caso de Uso Principal.

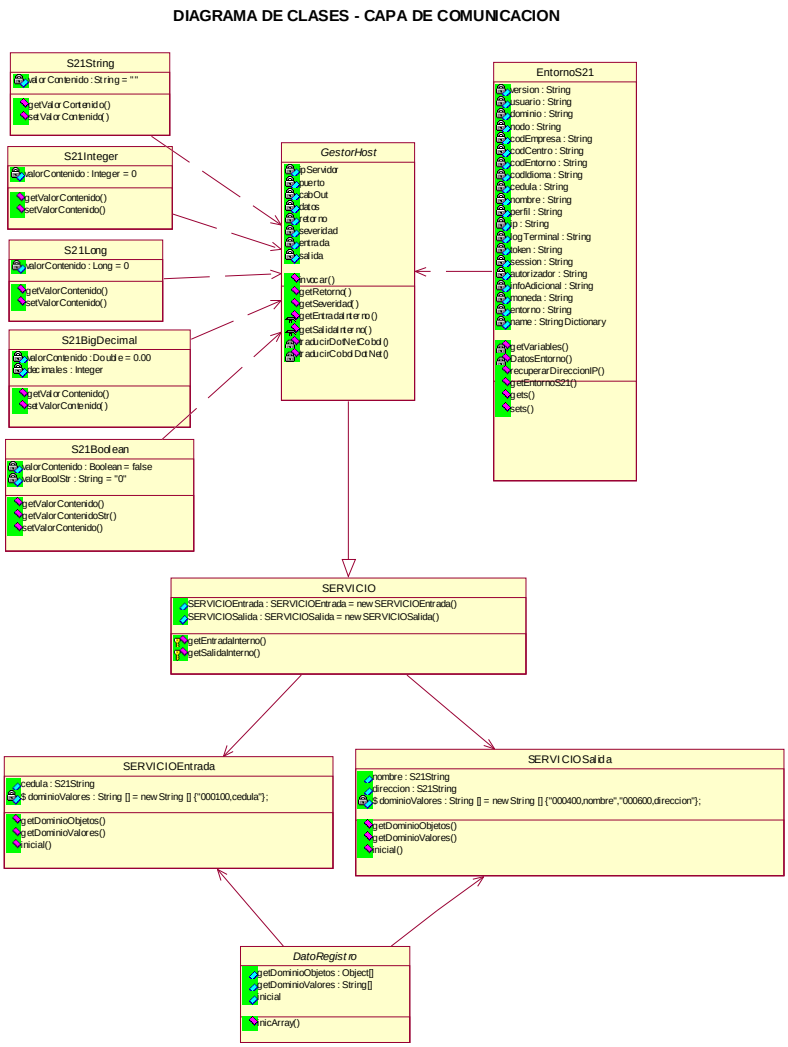


Gráfico 20. Diagrama de Clases, Capa de Comunicación.

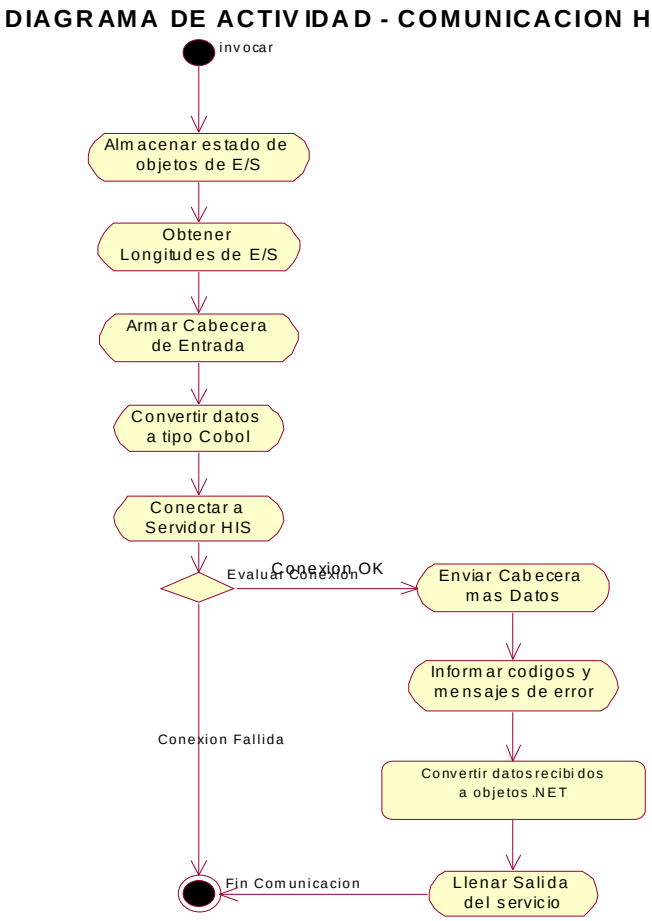


Gráfico 21. Diagrama Principal de Actividad – Comunicación Host.

2. IMPLEMENTACION DE CLASES DE COMUNICACIÓN WEB-OS/390

A continuación se muestra gráficamente la estructura planteada del componente de comunicación:

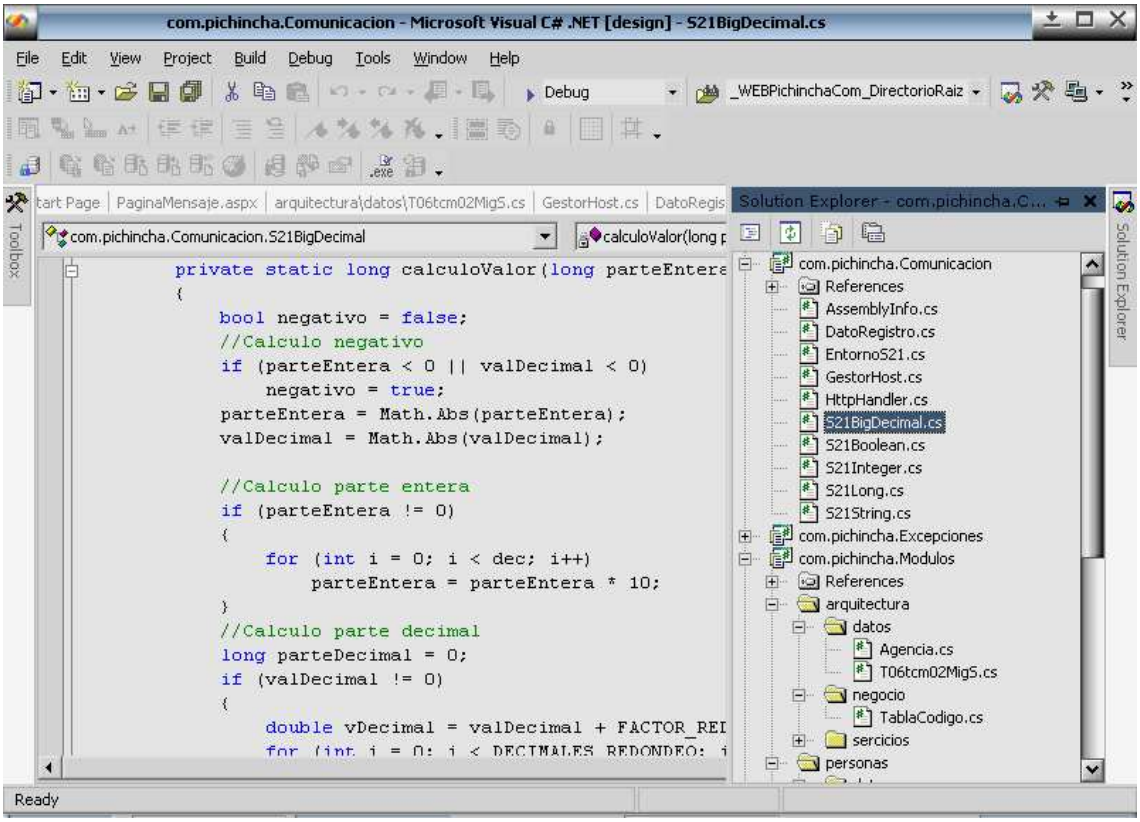


Gráfico 22. Componente de comunicación planteado.

Como se menciono anteriormente cada clase tiene su propia funcionalidad, a continuación se muestra el código fuente respectivo.

DatoRegistro.cs	(Ver Apéndice B).
GestorHost.cs	(Ver Apéndice C).
S21String.cs	(Ver Apéndice D).
S21Integer.cs	(Ver Apéndice E).
S21Long.cs	(Ver Apéndice F).
S21BigDecimal.cs	(Ver Apéndice G).
S21Boolean.cs	(Ver Apéndice H).

El objetivo de las seis ultimas clases es básicamente la de proporcionar objetos con estado para poder asignar y tomar el valor de cada propiedad pertenecientes a los servicios que se usarán para la transportación o comunicación de los datos hacia y desde el Host.

La siguiente pantalla muestra una prueba inicial de concepto, aunque no es el objetivo del presente proyecto:

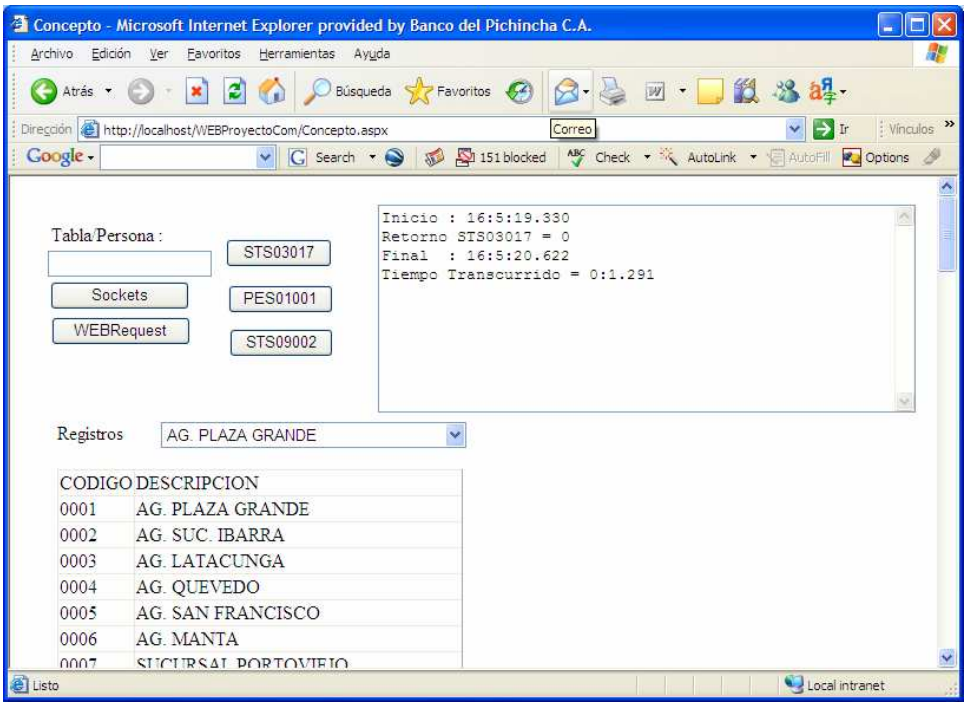


Gráfico 23. Prueba de concepto del componente planteado.

En esta pantalla se esta consultado en la base de datos Oracle las Agencias Migradas. El servicio para esta causa es el STS03017.

Las pruebas realizadas sobre la funcionalidad y operatividad de esta prueba de concepto se describen a continuación (Referencia "**APENDICE G** – METODOLOGIA MSF – DOCUMENTO MAESTRO, y **APENDICE H** – METODOLOGIA MSF – DOCUMENTO DE PLAN DE PRUEBAS"):

Fecha:	08/03/2005

PLAN DE PRUEBAS

Producto / Solución:	**COMUNICACIÓN HOST**
Proceso en Prueba:	**CONSISTENCIA DE DATOS**

Horario:	**15:00**	**Usuario:**	**Santiago Bustillos**	**Perfil:**	Usuario Final

Reporte del Usuario						Reporte del Proveedor		
ITEM	**FUNCIONALIDAD**	**PRE-REQUISITO**	**RESULTADOS ESPERADOS**	**RESULTADOS OBTENIDOS**	**CALIFICACION DE LA PRUEBA**	**FECHA COMPROM. DE SOLUCIÓN**	**FECHA EFECTIVA DE SOLUCIÓN**	**JUSTIFICACION AL FALLO PRESENTADO**
1	Verificar que los datos retornados sean los correctos	Crear el usuario para prueba asi como tambien tener acceso al PL/SQL para verificar los datos	Que los codigos y descripciones sean exactamente los que arroje la base de datos al consultar con la siguiente sentencia: SELECT CODIGO_REGISTRO, DEL_REGISTRO FROM T06TC002 WHERE COD_TABLA_ID = 'nombre_tabla_consultada'	Los resultados son los esperados, se comprobo la consistencia entre lo retornado por el servicio y la base de datos, y todo es correcto.	Exitosa			
2	Verificar que sucede cuando no existe un dato no encontrado	Crear el usuario para prueba asi como tambien tener acceso al PL/SQL para verificar los datos	Debe aparecer datos en blanco	Los resultados son los esperados	Exitosa			
3	Verificar rendimiento		Debe igualar o superar un tiempo de 4 segundos	Se hicieron varias pruebas: Tiempo: min:seg:miliseg : Tiempo 1 = 0:4.246 Tiempo 2 = 0:0.260 Tiempo 3 = 0:6.409 Tiempo 4 = 0:1.261 Tiempo 5 = 0:3.565 Tiempo 6 = 0:3.515 Tiempo 7 = 0:0.290 Tiempo 8 = 0:6.489 Tiempo 9 = 0:1.241 Tiempo 10 = 0:2.273 Promedio = 2:6	Exitosa			

Fecha:	05/05/2006

PLAN DE PRUEBAS

Producto / Solución:	**COMUNICACIÓN HOST**
Proceso en Prueba:	**CONCURRENCIA**

Horario:	**15:36**	**Usuario:**	**Equipo de Trabajo**	**Perfil:**	Usuario Final

Reporte del Usuario						Reporte del Proveedor		
ITEM	**FUNCIONALIDAD**	**PRE-REQUISITO**	**RESULTADOS ESPERADOS**	**RESULTADOS OBTENIDOS**	**CALIFICACION DE LA PRUEBA**	**FECHA COMPROM. DE SOLUCIÓN**	**FECHA EFECTIVA DE SOLUCIÓN**	**JUSTIFICACION AL FALLO PRESENTADO**
1	Verificar rendimiento con 10 hilos de ejecución concurrentemente durante 1 minuto	Disponer de la Herramienta Microsoft Applicaction Center Test	Debe igualar o superar un tiempo de 4 segundos	15:32:34.647 11 15:32:36.662 12 15:32:36.662 13 15:32:36.678 14 15:32:37.530 15 15:32:37.553 16 15:32:38.530 17 15:32:38.350 18 15:32:38.553 19 15:32:39.162 20 15:32:39.553 21	Exitosa			

Beneficios de la solución planteada.

Vale la pena especificar los beneficios que se tiene al acoplar el desarrollo de aplicaciones a la presente propuesta de comunicación:

- Como se dijo anteriormente, la presente propuesta proporciona una opción más de comunicación entre Servidores Web y el IBM 390.
- Su diseño está basado en la infraestructura actual tanto de hardware como de software de la empresa caso de estudio.
- Compite con otras tecnologías que se encuentran ya en el mercado cuyos costos son representativos.
- El código fuente esta disponible para modificaciones, analisis y depuraciones en cualquier momento, a diferencia de otros productos que básicamente son una “caja negra”.
- Su diseño implica el uso de estandares que se acoplen a la capa de comunicación o persistencia; esto ayuda a estandarizar el desarrollo de aplicaciones.
- La migración para comunicarse con HP Unix en lugar del IBM 390 implica únicamente la no conversión de ASCII a EBCDIC y viceversa; a diferencia de los objetos COMTIs para cuya migración involucraría modificaciones en los programas distribuidores; esto es debido a que el IBM 390 entiende tramas EBCDIC, a diferencia del HP Unix que entiende tramas ASCII.
- Permite compatibilidad de tipos de datos entre el Cobol y el C# bajo el IDE de .NET.
- Proporciona una facil comprensión del codigo fuente y de los utilitarios usados para el desarrollo de aplicaciones.

3. DEFINICIÓN DE METODOLOGÍAS Y ESTANDARES DE PROGRAMACIÓN PARA EL USO DE LA PLATAFORMA PLANTEADA.

En todo proyecto resulta imperativo definir estándares a seguir con la finalidad de establecer normas claras y básicas encaminadas a proporcionar un alto grado de escalabilidad, detección de errores, depuraciones, mantenimiento, etc. Por tal motivo, se presenta a continuación los estándares respectivos para el uso del componente de comunicación propuesto en el presente proyecto.

En esta sección se describirá en detalle los estándares a seguir en la capa de negocio; por lo tanto, se deberá tomar muy en cuenta los siguientes estándares:

3.1. Metodología.

Toda aplicación debe acoplarse a la siguiente metodología de trabajo con el fin de tener el código mas ordenado de tal manera que su mantenimiento sea los mas fácil posible.

- En todo desarrollo se empleara el Editor de Servicios .Net para crear las interfases de entrada y salida hacia el Host.
- Toda interfase será ubicada en su respectivo NameSapace de acuerdo a su funcionalidad.
- Todas las interfases generadas irán en la carpeta **servicios** del NameSpace respectivo.
- El código generado para las interfases no debe ser manipulado.

3.2. Estándares.

- **Estructura de Directorios**

Todo módulo residirá en el proyecto com.pichincha.Modulos; este proyecto se dividirá en módulos de acuerdo a la funcionalidad que se tenga. Cada uno de estos módulos se organizará de la siguiente manera:

- **Modulo**
  - **datos**
  - **negocio**
  - **servicios**

Únicamente en el subdirectorio negocio se podrá hacer referencia a la capa de comunicación, además, se podrá tener lógica de negocio que interactúa con los datos proporcionados con los servicios.

- **Archivos Fuentes.**

  El desarrollador únicamente tendrá acceso a la capa de presentación y a la capa de negocio; es decir, solo podrá modificar código en paginas (.aspx) y clases ubicadas en la siguiente estructura **Modulo.negocio,** esto es debido a que en el resto de capas la arquitectura esta ya planteada y el desarrollador no se debe preocupar mas allá del negocio y presentación.

  Es preciso tener un método en lo posible siempre con el mismo nombre en todas las clases de negocio llamado "**buscarObjeto",** en el cual se instanciara los servicios respectivos para enviar y recibir datos del Host.

  Es recomendable no sobrepasar en este tipo de objetos las 1000 líneas de programación, a menos que sea realmente necesario.

- **Codificación**

  A continuación se especifica algunas reglas bastante útiles para la codificación de programas:

- **Longitud y división de líneas**

  Las líneas de código deben tener visualización total, es decir, no deben exceder los 80 caracteres, en caso de que se sobrepase esta longitud, deben ser divididas de acuerdo a los siguientes criterios:

  - Después de una coma
  - Después de un operador
  - La nueva línea debe alinearse con el comienzo de la expresión al mismo nivel de la línea previa. Ejemplo:

```
buscarObjeto( expr1 , expr2 ,
              expr3 , expr4 , expr5 );
```

- **Identación**

  Es recomendable usar un TAB en la identacion de código

- **Líneas en blanco**

Las líneas en blanco facilitan la lectura. Siempre deben utilizarse una línea en blanco entre:

- Métodos
- Propiedades

- Las variables locales y el inicio de la primera sentencia en un método. Las variables locales se declaran en el momento en que se van a usar, no necesariamente en la cabecera del método
- Secciones lógicas dentro de un método para facilitar la lectura. Estas secciones no requieren de regiones. Pues el abuso de regiones impide la legibilidad.

- **Espaciado**

Tratar en lo posible de mantener un espacio después de una coma ó dos puntos y después de un paréntesis.

```
metodo( a , b , c );
```

Utilizar espacios simples antes y después de los operadores, excepto los operadores unarios (++,--) y el not lógico (!).

Ejemplo incorrecto:

```
a=b;
for(int i=0;i<10;++i)
if((a&&b)||(c&&(d==null)))
```

Ejemplo correcto:

```
a = b;
for ( int i = 0 ; i < 10 ; ++i )
if ( ( a && b ) || ( c && ( d == null ) ) )
```

- **Comentarios**

- Utilizar los símbolos ***//***, tanto para comentarios de una línea, como para bloques de varias líneas.
- Los comentarios se deben indentar al mismo nivel de identación que el código que se está documentando.
- Evite comentarios que expliquen lo obvio.
- El código debe ser auto-explicativo. El código de calidad con variables y métodos que puedan ser leídos con facilidad no debería requerir comentarios. Los nombres de variables legibles ayudan a este cometido.
- Evite documentación extensa a nivel de métodos. Los comentarios a nivel de métodos deben servir sólo como referencia para otros desarrolladores.

- **Declaraciones**

  Número de declaraciones por línea

  Se recomienda una declaración por línea, ya que de esta forma se puede añadir un comentario a la derecha de la declaración. Ejemplo:

```
int level; // indentation level
int size;  // size of table
```

  Inicialización

  Inicializar las variables tan pronto como sean declaradas. Ejemplo:

```
string name = myObject.Name;
int val = time.Hours;
```

- **Declaración de Clases e Interfases**

  Deben seguirse las siguientes reglas:
  - No deben haber espacios entre el nombre del método y el paréntesis "**(**" que empieza la lista de parámetros
  - La llave "{" debe aparecer en la siguiente línea después de la sentencia de declaración
  - La llave de cierre "}" debe aparecer en una nueva línea y con identación que coincida con la de apertura.
  - Nótese que se requieren espacios antes y después de los : de la declaración de herencia y antes de la coma de la lista de herencias
  - Ejemplo:

```
class MySample : MyClass , IMyInterface
{
   int _myInt;

   public MySample( int myInt )
   {
_myInt = myInt ;
   }

   void Inc()
   {
      ++_myInt;
   }

   void EmptyMethod()
   {
   }
}
```

- ***Prácticas de Programación***

    - Evite métodos con más de 25 líneas.
    - Evite métodos con más de 5 parámetros, utilice estructuras o clases para pasar múltiples argumentos.
    - Nunca utilice ”código duro”, en lugar de eso, declare una constante o use una enumeración.
    - Utilice siempre arreglos basados en 0
    - Únicamente añada excepciones catch cuando se maneje expresamente dicha excepción.
    - No utilice variables públicas ó protegidas, en lugar de eso utilice propiedades. Excepto cuando la clase requiere de rendimiento o es una clase que JAMAS va a cambiar la implementación de su atributo.
    - Evite métodos abstractos, utilice interfases.
    - Implemente siempre un case default en una sentencia switch

4. CREACIÓN DE HERRAMIENTAS UTILES PARA EL DESARROLLO DE APLICACIONES SOBRE LA PLATAFORMA PLANTEADA.

En esta sección explicaremos la utilidad del Editor de Servicios creado para proporcionar al desarrollador de aplicaciones una herramienta para la generación de interfases para comunicación con el Host que cumpla con los estándares requeridos.

El Editor de Servicios esta desarrollado en .Net con Windows Form; básicamente genera Servicios divididos en tres:

Servicio (Contiene los datos de entrada y salida)
Entrada
Salida

Ejemplo:

- TCS01002.cs
- TCS01002Entrada.cs
- TCS01002Salida.cs

Y, registros, es decir, objetos de tipo registro que contienen varias propiedades de tipo primitivas para la aplicación.

El Editor de Registros básicamente genera código estándar que el desarrollador no debe modificar; este código se basa en los métodos abstractos heredados de las dos principales clases: GestorHost y DatoRegistro; además de proporcionar la metadata para la conversión respectiva de objetos a bytes y viceversa.

Su uso es simple, pues proporciona un ambiente visual amigable y entendible que acopla en definitiva el área de comunicación Cobol con los respectivos objetos .Net.

Su diseño completo se especifica en el "**APENDICE E** – METODOLOGIA MSF – DOCUMENTO DE REQUERIMIENTOS y,
APENDICE F – METODOLOGIA MSF – DOCUMENTO DE ESPECIFICACIONES FUNCIONALES" correspondientes a la metodología MSF, sin embargo, a continuación se muestra la parte principal del diseño del utilitario:

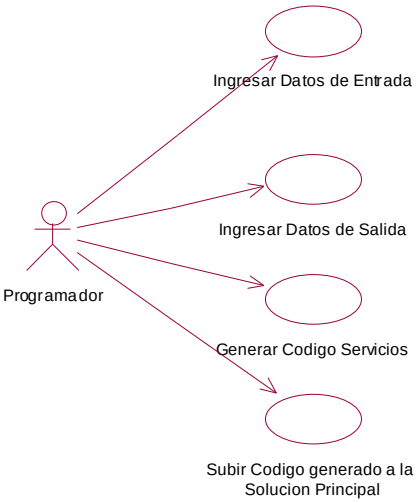


Gráfico 24. Caso de Uso Principal - Utilitarios.

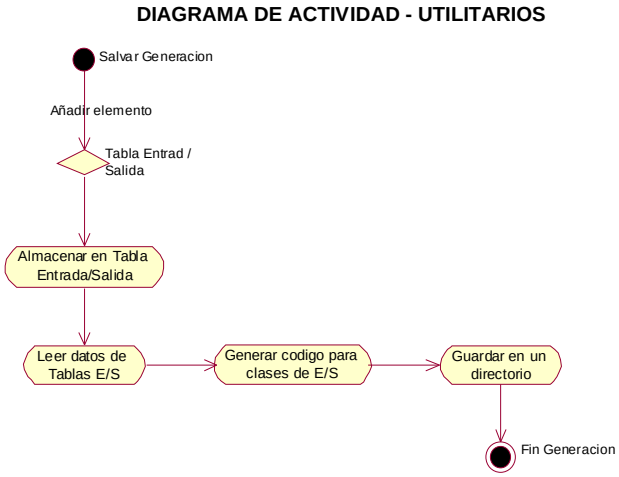


Gráfico 25. Diagrama Principal de Actividad - Utilitarios.

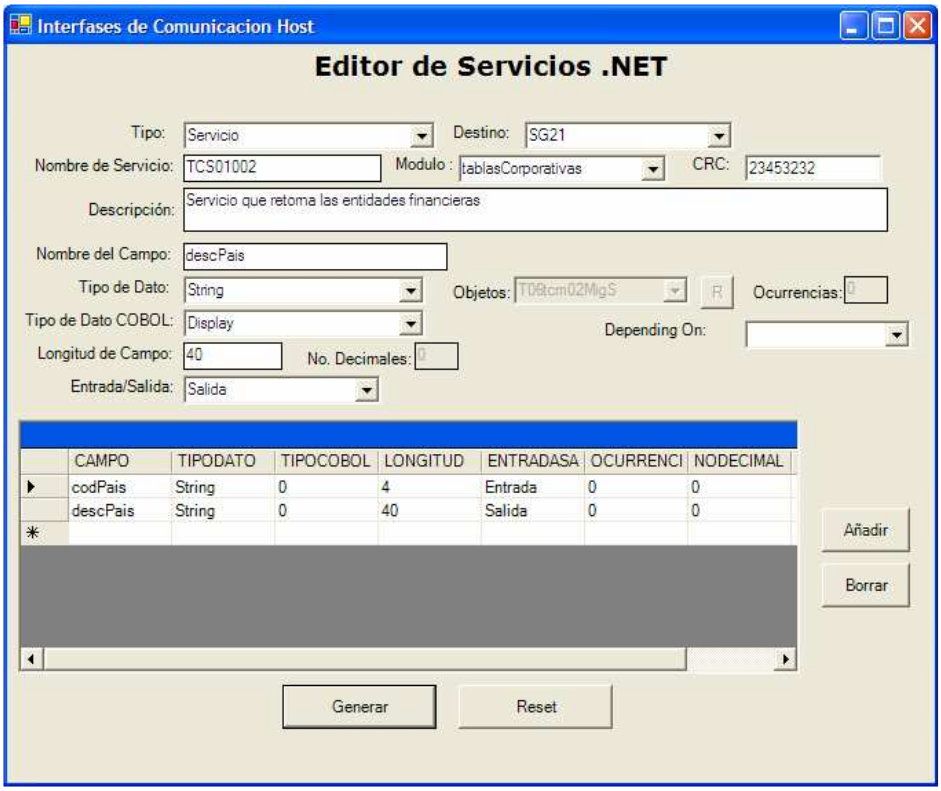


Gráfico 26. Herramienta para generar servicios .NET

Las pruebas realizadas sobre la funcionalidad y operatividad de esta prueba de concepto se describen a continuación (Referencia "**APENDICE G** – METODOLOGIA MSF – DOCUMENTO MAESTRO, y **APENDICE H** – METODOLOGIA MSF – DOCUMENTO DE PLAN DE PRUEBAS"):

						Fecha:	08/03/2005	
PLAN DE PRUEBAS								
Producto / Solución:	Utilitarios del proyecto HOST							
Proceso en Prueba:	Usabilidad							
Horario:	**9:30**	**Usuario:**	**Bustillos Santiago**		**Perfil:**	Usuario Final		

Reporte del Usuario						**Reporte del Proveedor**		
ITEM	**FUNCIONALIDAD**	**PRE-REQUISITOS**	**RESULTADOS ESPERADOS**	**RESULTADOS OBTENIDOS**	**CALIFICACION DE LA PRUEBA**	**FECHA COMPROM. DE SOLUCIÓN**	**FECHA EFECTIVA DE SOLUCIÓN**	**JUSTIFICACION AL FALLO PRESENTADO**
1	Verificar la usabilidad	Framework .NET	La aplicación correra independientemente de donde recida	Los resultados son los esperados	Exitosa			
2	Ingresar datos del area de comunicación	Conocer las estructuras de datos de Cobol	Aparecera los datos ingresados en una tabla ubicado debajo de la pantalla del editor de servicios .NET	Los resultados son los esperados	Exitosa			
3	Generar el codigo correspondiente a las areas de entrada y salida de comunicación		Verificar la existencia de los archivos generados en el directorio de trabajo	El código es almacenado en el directorio C:\EditorNET	Exitosa			

5. ESTRUCTURA DE RED PLANTEADA DESDE UNA AGENCIA

De acuerdo al esquema de comunicación planteado en el presente proyecto, definiremos a continuación gráficamente cual sería el esquema de red para una Agencia piloto.

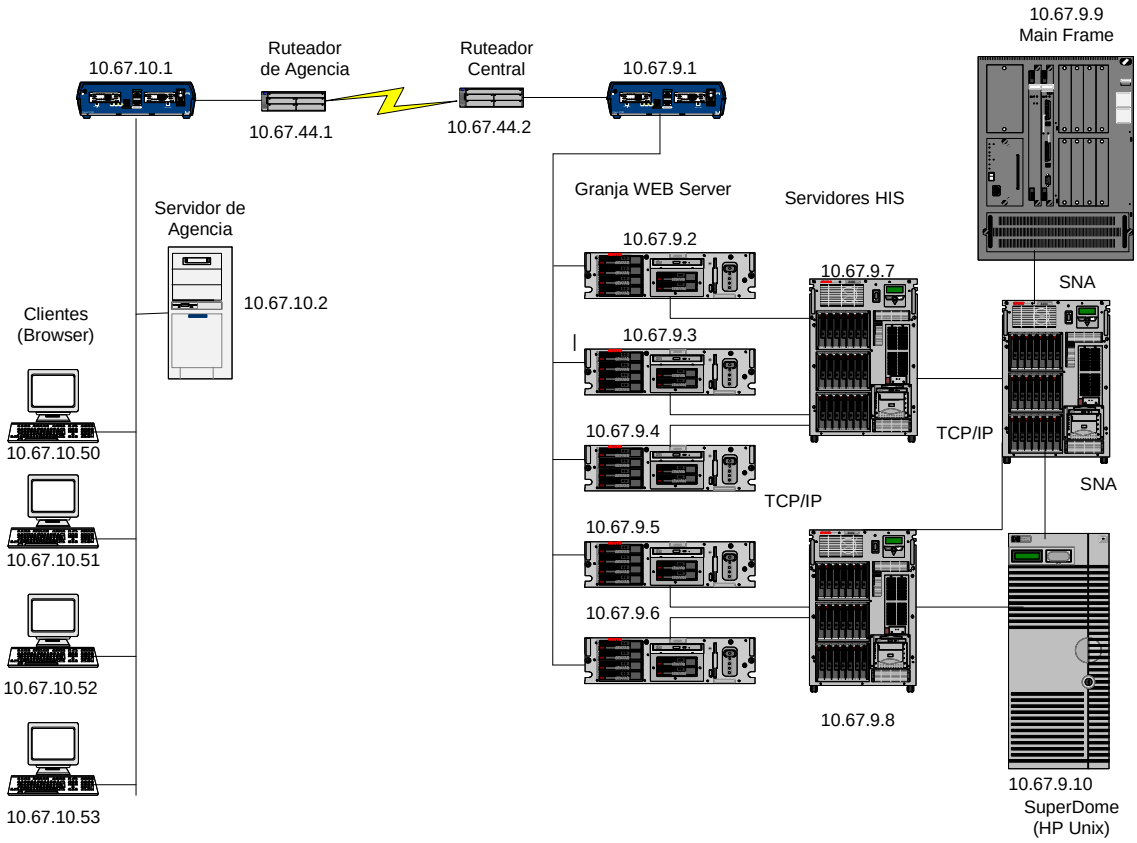


Gráfico 27. Elementos de la propuesta planteada.

Como se ve en el gráfico, el enlace WAN utiliza protocolos TCP/IP; por otra parte, para establecer comunicación entre los servidores HIS con el MainFrame o SuperDome, se utiliza el Link Distribuido que nos proporciona el SNA (referirse al Apéndice A de Anexos); lo cual básicamente quiere decir SNA sobre TCP/IP, proporcionando de esta manera mayor rendimiento a demás de no utilizar ciertos protocolos adicionales como el DLC.

Cada Agencia dispondra de su propia subred; la misma que se interconectará mediante ruteadores a la red principal; en esta red se encuentran los servidores WEB los mismos que interactuan con el Browser de los clientes y además transmitirán toda la mensajería hacia y desde el Host; para esto se conecta via sockets al o los puertos destinados para atender las peticiones de los servidores WEB en los servidores HIS.

En los Servidores SNA (HIS), por cada solicitud enviada por los Servidores WEB, se dispondra de una sesion SNA para disparar la petición hacia el Host,

en donde sera recibido por un programa denominado “distribuidor” hecho en Cobol, cuya lógica dispondrá, de acuerdo a la cabecera de la transacción, a qué programa (denominado “servicio”) le pasará el control con la respectiva área de comunicación

El programa Cobol (servicio), al cual el programa distribuidor le paso el control, tiene la capacidad de realizar accesos a la base de datos Oracle; con dichos datos implementara su lógica de negocio, y el resultado es transmitido al programa distribuidor quien realizara el respectivo commit o rollback en caso de presentar errores, y finalmente transmitirá la información hacia los servidores SNA quienes por su lado también transmitirán via sockets a los servidores Web.

CAPITULO IV

CONCLUSIONES Y RECOMENDACIONES

CONCLUSIONES

- El presente documento establece pautas básicas para desarrollar en un ambiente actual como es el FrameWork .NET.

- Se consideró varias prácticas de programación utilizadas en las diferentes herramientas como son Pascal, C++, Java, etc. para acoplarlo en una sola y plasmarlo en el presente documento.

- El presente documento esta orientado a todo tipo de programador, pues el diseño de comunicación tiene escalabilidad desde el punto de vista de lenguaje de programación.

- El diseño de la librería de comunicación se basa en la orientación a objetos, parte esencial para dicho componente, pues su herramienta de desarrollo es C# de .NET.

- .Net facilita el desarrollo de aplicaciones mediante su IDE, proporcionando un ambiente integrado de programación de fácil entender y navegación.

- El diseño de comunicación consta de dos partes: la parte de comunicación propiamente dicha (mediante sockets), y la parte de conversión de datos desde .NET a Cobol y viceversa.

- .Net contiene librerías propias que proporcionan al programador múltiples funciones; en este caso, funciones de comunicación como es el caso de los sockets.

- El diseño de desarrollo se basa en la arquitectura que propone .NET, es decir, una arquitectura basada en capas, cada una con funciones específicas; por ejemplo: Comunicación, Negocio, Presentación, etc.

- La presente propuesta se centra en la capa de Comunicación.

- El factor que mas incide en la elaboración de la presente propuesta es la de dar una alternativa de comunicación entre la WEB y el Mainframe reutilizando recursos sin ligarnos a herramientas de terceros que no se tiene control de los fuentes.

- Su diseño se basa en la infraestructura actual tanto de hardware como de software que posee actualmente la empresa caso de estudio.

- El código fuente propuesto para esta proyecto es de fácil entendimiento y no causa complejidad para su mantenimiento, proporcionando una facil comprensión del codigo fuente y de los utilitarios usados para el desarrollo de aplicaciones.

- La implementación y uso de la presente propuesta implica el uso de estandares que se acoplen a la capa de comunicación o persistencia; lo cual ayuda a estandarizar el desarrollo de aplicaciones.

- La implementación y uso de la presente propuesta compite con tecnologías que se encuentran ya en el mercado cuyos costos son representativos.

- El código fuente esta disponible para modificaciones, analisis y depuraciones en cualquier momento, a diferencia de otros productos que básicamente son una “caja negra”.

- La implementacion y uso de la presente propuesta es de fácil migración en cuanto a la comunicación con el HP Unix en lugar del IBM 390, lo cual únicamente implica la no conversión de ASCII a EBCDIC y viceversa en la clase de convesión de tipo de datos; a diferencia de los objetos COMTIs para cuya migración involucraría modificaciones en los programas distribuidores; esto se debe a que el IBM 390 entiende tramas EBCDIC, a diferencia del HP Unix que entiende tramas ASCII.

- El diseño e implementación de la presente propuesta, permite compatibilidad de tipos de datos entre el Cobol y el C# bajo el IDE de .NET.

- A pesar de que la presente propuesta se basa en los recursos que posee la empresa caso de estudio; no esta por demás decir que puede ser usado como base para otras empresas que posean una infraestructura parecida a la empresa analizada.

- Técnicamente la capa de comunicación propuesta, tiene una mensajería basada en cabecera y cuerpo de mensaje; la cabecera por su parte, se basa en la infraestructura actualmente montada de los programas distribuidores Cobol residentes en el mainframe; y el cuerpo del mensaje se basa en el área de comunicación que tendra cada programa o servicio Cobol disparado por el distribuidor.

RECOMENDACIONES

- Es importante destacar que quienes participen en el desarrollo de aplicaciones usando e implementando la presente propuesta, tengan conocimientos de Cobol y .Net; al igual que conocimientos de la parte del Negocio.

- Utilizar la estandarización en cuanto al desarrollo de aplicaciones basados en la presente propuesta es imperativo, pues las reglas propuestas son estrictamente obligatorias de seguirlas, ya que el no acatar dichos estándares puede conllevar a errores tanto de compilación como de ejecución.

- El uso de los utilitarios anexos a la presente propuesta es importante; pues a pesar de que dichos utilitarios generan codigo de fácil comprensión y de fácil creación, el uso de estos aplicativos lo es más fácil aún, pues se basa en pantallas amigables y opciones que ayudan a la creación de los servicios y areas de comunicación que se transmitirá y recibirá desde el mainframe.

- Diseñar areas de comunicación tomando en cuenta el tamaño de la misma; cuidar en lo posible de no exceder los 32 KB.

- Se recomienda la implementación de la presente propuesta pues sus costos son mínimos ya que su desarrollo se basa en los recursos actuales de la empresa caso de estudio; además de proporcionar un alto grado de migración hacia herramientas web sin tener que emprender proyectos dedicados a modificar toda la estructura de componentes especialmente de software que posee la empresa caso de estudio.

- Estructurar e implementar las otras capas que no fueron analizadas en la presente propuesta, principalmente la capa de Excepciones.

GLOSARIO DE TERMINOS

CICS (Customer Information Control System). Sistema transaccional bajo el cual se ejecuta transacciones y programas dentro de un mainframe.

Código duro, Programación cuya codificación de variables se las inicializa con valores predefinidos en donde dicha variable se convierte en constante.

COM (Component Object Model). Introducido por Microsoft se ha convertido en una importante tecnología para desarrollos desde cliente/servidor con orientación a objetos y bajo redes Windows NT y superiores.

COMTI (COM Transaction Integration). Herramienta desarrollada por Microsoft para integrar la tecnología COM con los productos y sistemas de tercera generación.

LAN (Local Area Network). Red de área local, red de transmisión de datos de alto volumen que conecta varios dispositivos intercomunicados dentro de una misma habitación edificio o complejo u otra área geográfica limitada

LPAR (Logical Partitioning). Partición lógica para equipos iSeries de IBM.

MAINFRAME (Equipo). Usado para definir equipos de gran tamaño, pueden llegar a ocupar un cuarto. Es un término comúnmente usado para definir los los grandes equipos de IBM.

MIPS (million instructions per second). Termino correspondiente a Millones de transacciones por segundo.

.NET (Dot Net). Arquitectura de Microsoft para aplicaciones basadas en Internet.

OSI (Open System Interconnection Model). Modelo desarrollado por la ISO que consta de 7 capas de red de comunicación.

Protocolo (Protocol). Conjunto formal de convenciones que gobiernan el formato y las señales de reloj relativas del intercambio de mensajes entre dos sistemas que se comunican.

Protocolo Internet (IP – Internet Protocol). Protocolo de nivel de red del conjunto de protocolos TCO/IP, Internet.

Red (Network). Conjunto de nodos interconectados.

TCP/IP (Transmisión Control Protocol/Internet Protocol). Protocolo de control de transmisión / Protocolo Internet; es un conjunto de protocolos utilizado en internet generalizándose su uso para la interconexión de redes heterogéneas

SNA (System Network Architecture). Arquitectura de comunicación de datos para equipos IBM, que definen los niveles de protocolos para comunicar terminales y aplicaciones y entre programas.

TX-Series. Monitor transaccional (Transaction Server) equivalente equivalente al CICS and Transaction Server de IBM.

WAN (Wide Area Network). Red de área extendida, red de transmisión de datos de medio volumen que conecta varios dispositivos intercomunicados dentro de una un área geográfica muy amplia.

ANEXOS

APENDICE A - SNA Distribuido

El siguiente es un artículo tomado de la siguiente URL:

http://support.microsoft.com/default.aspx?scid=http://www.microsoft.com/intlkb/spain/E10/2/84.asp#appliesto

Uso de los enlaces distribuidos de SNA Server

Se aplica a

Este artículo se publicó anteriormente con el número E10284

Resumen

En este artículo se describe el uso de los enlaces distribuidos de SNA Server.

Síntomas

Un caso muy común en instalaciones de SNA Server es el siguiente:

Una compañía dispone de una instalación operativa de SNA Server 2.11 en una red local con acceso a un AS/400 (o Mainframe IBM) vía DLC 802.2 (el enlace puede ser cualquier otro soportado por SNA Server, como SDLC, X.25, DFT o Twinax).

Por razones de expansión de la compañía, se desea instalar una nueva red local que se comunicará con la anterior vía routers y protocolo TCP/IP, usando X.25, RDSI o Frame Relay, conformando así una WAN. Por supuesto, requerimiento principal es tener conectividad con el AS/400 remoto desde la nueva red local. Debido a que las tramas DLC 802.2 no son encapsuladas en los paquetes TCP/IP, no podemos usar un enlace DLC 802.2 desde la nueva red local. Además, tampoco disponemos de adaptadores X.25 o SDLC para una conexión directa al AS/400.En definitiva, lo más óptimo sería aprovechar de alguna forma el enlace DLC existente en la red remota.

Solución

En primer lugar necesitaremos el Service Pack 1 de SNA Server 2.11. Dicho Service Pack puede conseguirse desde Internet (WWW.Microsoft.Com), mediante suscripción al servicio Technet o solicitándolo al departamento de soporte técnico de Microsoft.

Pasos a realizar en el servidor SNA original:

Instalar el Service Pack 1.
Ejecutar el programa "SETUP" del grupo de programas de SNA Server.
En la opción "Links" señalar el enlace DLC 802.2 que tenemos hacia el AS/400 y pulsar el botón "Distribute". Al finalizar esta operación nos aparecerá "(D)" delante del enlace DLC 802.2, indicando que ya está distribuido.

Pasos a realizar en el nuevo servidor SNA:

En primer lugar debemos verificar que la conexión al servidor Windows NT con SNA Server funciona correctamente. Por ejemplo, si usamos protocolo TCP/IP podemos comprobarlo ejecutando PING <dirección IP del servidor remoto> desde una ventana de comandos de Windows NT.

Instalar SNA Server 2.11 y el Service Pack 1 de SNA Server 2.11.

Ejecutar el programa "SETUP" del grupo de programas de SNA Server.

En la opción "Links"/ "Add", elegir "Distributed Link Service".

De la lista desplegable "Link Type" elegir el tipo adecuado (normalmente Ethernet 802.2).

En el campo "Remote Link Services" deberemos indicar:
\\<servidor nt remoto>\<servicio distribuido>

Donde:

<servidor nt remoto> puede ser el nombre NetBIOS del servidor NT con SNA Server en el cual hemos distribuido el enlace DLC 802.2 y opcionalmente, si usamos protocolo TCP/IP, la dirección IP de dicho servidor (la misma que usamos con el comando PING para probar la conexión).

<servicio distribuido> es el nombre que SNA Server dio en el momento de su instalación al enlace distribuido en el servidor remoto, es decir, el nombre que aparece en "Service Name: " por ejemplo: "SnaDLC1".
Una vez hecho esto, podemos usar el enlace distribuido remoto (en nuestro ejemplo de tipo DLC 802.2) como si fuera un enlace local a nuestro servidor, consiguiendo así la conectividad deseada en el planteamiento.

Ante-proyecto

[Comunicación Host]

CONFIDENCIAL
[Comunicación HOST - .NET]
[EPN-Tesis]
28 de Enero de 2005
Versión [1.0]

Contenido

(El contenido del documento original es el siguiente)

Antecedentes

A partir del año 2000 el Banco del Pichincha implementa una solución tecnológica de Hardware y Software Bancario adquirido a una empresa de software; dicha solución especificaba elevados costos principalmente en lo que se refiere a Hardware. El Banco en su necesidad de abaratar costos implementa paralelamente proyectos destinados a dicha causa, entre los cuales están migraciones masivas de herramientas de programación y base de datos y principalmente sustitución de los equipos principales que proporcionan el Sistema al Banco.

La solución informática que se implementaba en ese entonces requería de especificaciones muy costosas de Hardware, específicamente en cuanto a la renovación del equipo central en el que se especificaba migrar de un IBM S/390 Modelo Y36 G5 con 3 procesadores de 139 mips (ver características : http://www.ibm.com/br/products/servers/zseries/g5g6/g5g6_glance.phtml) a un IBM S/390 Modelo Y56 G5 cuyo procesamiento alcance mínimo los 700 mips (ver características : http://www.ibm.com/ec/products/servers/zseries/g5g6/g5g6_glance.phtml); adicionalmente, dicha renovación impactaba directamente en los costos de licenciamiento y renta de software de IBM, tales como CICS, MVS, VTAM, etc.; haciendo inalcanzable económicamente la renovación de dicho equipo; a mas de los costos de la implementación del software bancario adquirido; que por razones de confidencialidad no se puede incluir como anexo de costos del proyecto inicial.

Por otra parte, el Front End estaba desarrollado en la plataforma Java versión 1.1.8 la cual requería de máquinas de por lo menos 128MB de memoria principal, que a pesar de ser una herramienta actual, el diseño en el cual estaba implementada no era de fácil migración para la WEB que era uno de los objetivos que perseguía el Banco.

Para solucionar este inconveniente el Banco inicia un proyecto cuya finalidad es cambiar el Front End de Java a un Front End WEB utilizando herramientas tecnológicas actuales como es el caso del FrameWork .NET, en cuya implementación surge la necesidad de establecer una vía de comunicación hacia el S/390, y se llega a la conclusión de comunicarse vía COMTI's de Microsoft.

Pero, con algunas limitantes que describiremos en otras secciones a cerca del uso de los COMTI's, y principalmente el atarse a un proveedor específico hace que el desarrollo de la nueva aplicación propuesta por el Banco sea dependiente de aplicaciones comerciales; razón por la cual se pone a consideración la presente propuesta de comunicación .NET hacia el Host Central (S/390).

Visión del Proyecto

Contar con una propuesta alternativa de comunicación desde el Front End hacia el Back End garantizando los niveles de servicios deseados a mínimos costos.

Objetivos del Proyecto

Objetivos		
Objetivos de Negocio	**Proporcionar estándares de codificación para el desarrollo de aplicaciones**	
Objetivos Técnicos/Procesos	**1**	Recopilar información de reglas y normas de codificación en diferentes lenguajes de programación.
	2	Analizar reglas y normas recopiladas.
	3	Definir las reglas, normas y estándares a utilizar en el desarrollo de aplicaciones.
	4	Documentar y registrar los avances.

Objetivos de Negocio	**Disponer de una alternativa mas de comunicación contra el OS/390 y Unix.**	
Objetivos Técnicos/Procesos	**1**	Diseñar una propuesta de comunicación que se acopla a las necesidades de la Organización.
	2	Utilizar las herramientas necesarias para la elaboración de la propuesta.
	3	Enfocar la propuesta al desarrollo de aplicaciones WEB con .NET enfatizando el uso de los estándares definidos.
	4	Documentar y registrar los avances.

Objetivos de Negocio	**Utilizar plataformas actuales de desarrollo de aplicaciones.**	
Objetivos Técnicos/Procesos	**1**	Utilizar la herramienta de programación C# de .NET.
	2	Explotar las librerías correspondientes a sockets en .NET.
	3	Identificar los utilitarios para el desarrollo de aplicaciones acoplados a la arquitectura planteada.
	4	Documentar y registrar los avances.

Objetivos de Negocio	**Especificar el esquema de red de una agencia acoplada a la arquitectura planteada.**	
Objetivos Técnicos/Procesos	**1**	Definir los componentes que forman parte de las comunicaciones desde una Agencia al S/390 y Unix.
	2	Graficar y explicar el esquema de red planteado.
	3	Documentar y registrar los avances.

Concepto de la Solución

Definir claramente los componentes que intervienen en la propuesta de comunicación hacia el S/390 y Unix. Esta propuesta debe ser consistente con los recursos y requerimientos de la Organización en cuanto a Hardware, software.

Se trata entonces, de proporcionar un entregable en el que se definan las características principales de la propuesta, sus estándares, herramientas de soporte para desarrollo de aplicaciones acopladas a la propuesta y el esquema arquitectónico y de red para una Agencia piloto; de tal manera que, la propuesta se enmarca en las siguientes características:

- Definiciones claras de cuál es el modelo de la propuesta de comunicación.
- Definición de reglas y estándares para el desarrollo e implementación de la propuesta.
- Descripción de ventajas y desventajas para la implementación de la propuesta.
- Definición y diseño de la arquitectura de componentes para la comunicación con el Host.
- Descripción de las herramientas utilizadas para el desarrollo de las librerías de comunicación.
- Definición y diseño de los utilitarios que darán soporte al desarrollo de aplicaciones basadas en la propuesta dada.
- Estructurar pruebas iniciales de concepto de las librerías de comunicación.
- Uso de recursos de hardware y software de la organización.

Alcances

Alcances de las Opciones

Tópico	Prioridad	Concepto Funcional
Comunicación .NET con el S/390 y su equivalente al UNIX	I	Análisis y diseño de librerías de comunicación hacia servidores HIS.
Conversión de tipos de datos .NET-Cobol	II	Análisis y diseño de conversión de tipo de datos: objetos C# a Cobol y viceversa.
Capa de persistencia	III	Análisis y diseño de la capa de persistencia mediante herramientas útiles para el desarrollo de aplicaciones bajo

		el esquema planteado

Costos Estimados.

Tipo	Recurso	Costo (USD/hora)	Tiempo (horas)	Total (USD)	Descripción
Recurso Humano	Nelson Porras	10	240 horas (Estimado dos horas no laborables diarias)	2.400	Tomado en base a la siguiente formula: Sm / (4s * 5dl * 8hl) Donde: Sm = Sueldo mensual s = semanas dl = días laborables hl = horas laborables
Recurso Humano	Christian Flores	7	240 horas (Estimado dos horas no laborables diarias)	1.680	Tomado en base a la siguiente formula: Sm / (4s * 5dl * 8hl) Donde: Sm = Sueldo mensual s = semanas dl = días laborables hl = horas laborables
Recurso Tecnológico	Un Equipo de Trabajo	1	240	240	Proporcionado por el Banco, incluye uso de energía eléctrica.
Recurso Tecnológico	Una Lapto	0.10	240	24	De propiedad de Christian Flores, incluye uso de energía eléctrica e Internet.
Recurso Tecnológico	Un Equipo de Trabajo	0.10	240	24	De propiedad de Nelson Porras; incluye uso de energía eléctrica e Internet

Beneficios estimados o justificación

La implementación de la presente propuesta proporcionara a la Organización los siguientes beneficios:

- Administración total del código generado para la comunicación contra el OS/390 o Unix.
- Contar con reglas y estándares de codificación para programadores Front End.
- Contar con herramientas útiles para el soporte de desarrollo de aplicaciones Front End.
- Participación activa de programadores Host o Back End en el diseño y la comprensión de clases Front End.

- Contar con un único componente de comunicación hacia el OS/390 y Unix.

- Fácil integración de normas de seguridad en el componente de comunicación por ser único; por ejemplo: Log de auditorias, estadísticas de uso transaccional, etc.

- Reducción de costos de implementación.

- Simplificar los procesos de monitoreo y de elaboración de reportes.

COMUNICACIÓN HOST

Estructura del Proyecto

CONFIDENCIAL
Plataforma: Microsoft .NET
Fecha: 02/02/2005
Versión: 1.0

Contenido

(El contenido del documento original es el siguiente)

Estructura de los Requerimientos

Introducción

En este documento se especifica y define la estructura administrativa del equipo de trabajo para atender los requerimientos asociados a la Propuesta de conexión .NET para comunicación con OS/390 vía TCP/IP y SNA, tales como, roles de cada uno de los miembros del grupo de trabajo, la estructura de los reportes, las reuniones, cronogramas.

Equipo del Proyecto

A continuación, se describe la asignación de roles, para el desarrollo de las funcionalidades de la Propuesta:

Rol	Recurso	% Dedicación	Teléfono	E-mail
Líder de Producto	Marco Jarrín	30 %		
Líder de Programa	Nelson Porras	100 %	2509190 Ext. 2421	nporras@pichincha.com
Líder de Programa	Christian Flores	100 %		CFLORESM@repsolypf.com
Desarrollador	Nelson Porras Christian Flores			
Pruebas	Nelson Porras Christian Flores Santiago Bustillos	SB-10%		obustill@pichincha.com

Estructura de Reportes

El Líder de Programa entregará a todo el equipo de trabajo un reporte del avance de las funcionalidades y tareas puntuales semanalmente. El formato del reporte será el especificado en el Anexo 1.

Estrategia de Comunicación

Frecuencia de Reuniones: 4 Reuniones semanales, a las 19:00, en la residencia de Nelson Porras.

El Líder de Programa Nelson Porras, enviará la agenda vía correo electrónico a todos los participantes con un día de anterioridad a cada reunión hasta las 15:00hrs.

La presencia de los líderes de programa es obligatoria para realizar la reunión.

Para la convocatoria se usará como herramienta de comunicación el calendario de Outlook, en el cual se incluirá la agenda a tratar en la agenda y el tiempo asignado a la reunión. La confirmación de los participantes deberá ser explícita por el mismo medio. La no confirmación significa la aceptación de la convocatoria.

El borrador de acta de reunión será enviado por el líder de Programa utilizando el correo electrónico, a primera hora del siguiente día en que se lleve acabo la reunión.

ANEXO 1. Reporte de Avance de Requerimiento

Reporte de Avance de Requerimiento	
No.	
Empresa	EPN-TSIS
Descripción Requerimiento [yyyy-mm-dd]	COMUNICACIÓN HOST
Fecha de Elaboración [yyyy-mm-dd]	
Desde [yyyy-mm-dd]	
Hasta [yyyy-mm-dd]	

Tarea	Fecha de Inicio	Fecha final	Avance (%)	Recurso	Observaciones

COMUNICACIÓN HOST

VISION Y ALCANCE

PROPUESTA DE COMUNICACIÓN HOST

CONFIDENCIAL
Plataforma: Microsoft .NET
Fecha: 02/02/2005
Versión: 1.0

CONTROL DEL DOCUMENTO.

DATOS GENERALES DEL DOCUMENTO VIGENTE			
Código	**Versión**	**Nombre**	**Autor**
EPNTSIS-01	1.0	Documento de visión y Alcance	Nelson Porras – Cristian Flores

LISTADO DE DISTRIBUCIÓN			
Empresa	**Nombre y Apellidos**	**Cargo**	**Fecha**
OMNISOFT	Marco Jarrín	Supervisor del Proyecto	
Banco del Pichincha	Nelson Porras	Project Manager	
Repsol ypf	Christian Flores	Project Manager	

REGISTROS DE CAMBIOS EN EL DOCUMENTO			
Versión	**Motivo**	**Realizado por**	**Fecha**
1.0	Propuesta inicial	Nelson Porras y Christian Flores	2005-01-31

CONTENIDO.

(El contenido del documento original es el siguiente)

1. INTRODUCCIÓN.

El quehacer diario de una Institución Financiera se orienta a proporcionar servicios de mejor calidad a sus clientes, muchos de estos servicios son netamente tecnológicos orientados a dar soporte a los objetivos institucionales de la Empresa. El Banco del Pichincha C.A, cuenta con 270 oficinas y aproximadamente dos millones de clientes activos, distribuidos en todo el Ecuador. Estos antecedentes, reflejan un alto grado de compromiso tanto Financiero como tecnológico para mantener el liderazgo bancario en el Ecuador.

Los Objetivos de una Institución Financiera se orientan al hecho de "Hacer Banca", no a la Tecnología; sin embargo, dicha tecnología hoy en día se ha convertido en la columna vertebral de toda organización para la operación de sus funciones y consecución de sus fines. Esto ocasiona que toda Organización deba incluir dentro de sus presupuestos, rubros destinados a todo lo que se refiere con Tecnología.

Es necesario entonces, evaluar propuestas tecnológicas cuyos presupuestos no lleguen a cifras exorbitantes y cumplan con las exigencias tecnológicas que requiere una Organización para apoyar a la consecución de los objetivos Institucionales.

En la actualidad, el Banco del Pichincha C.A., a través de la Unidad de Cumplimiento (UC), tiene como objetivo principal la de proporcionar servicios de mayor calidad, dentro de un marco de mejoramiento continuo, incrementando la eficacia en todas las áreas de la Organización, así como su eficiencia operativa, para reducir costos. Es por esto, que nace esta propuesta para apoyar al cumplimiento de este objetivo, reduciendo costos mediante el diseño y estandarización de clases y librarías de comunicación con las que funcionara el core tecnológico del Banco desarrolladas con herramientas de punta y utilizando recursos existentes en la Organización.

Este documento contiene la formulación del problema, donde se identifica y tipifica a los actores que interactúan con la solución; también contiene la visión de la solución, donde se establecen los objetivos; se describe el modelo de la solución, características, la matriz de cobertura de necesidades y las restricciones; finalmente se identifica a las personas requeridas según su rol para la aprobación de este documento.

2. FORMULACIÓN DEL PROBLEMA.

Identificación del Problema	
Descripción	❖ Incapacidad de administración de código fuente para la comunicación entre Front end y Back end, ya que se posee componentes Microsoft de código cerrado. ❖ Dificultad del software actual para soportar las arquitecturas implementadas en el Banco del Pichincha, en lo que se refiere a Equipos 390 y Unix. ❖ Dificultad en consolidar un único mecanismo de comunicación en el cual se pueda establecer filtros de seguridad de acuerdo a los requerimientos de la Organización. ❖ Falta de estándares de programación para desarrollo en Front end ❖ Dificultad en el entendimiento de código para programadores de Front end y Back end.
Impacto	El Banco del Pichincha C.A., al no desarrollar e implementar la presente propuesta, se ve sujeto a: ❖ Establecer otros mecanismos para solventar los requerimientos de conversión de datos para las distintas arquitecturas de hardware que posee el Banco del Pichincha, que no proporciona las herramientas de software actuales. ❖ No administrar o controlar el software de conexión con el OS/390 y Unix. ❖ Difícil administración del software codificado en el Front end, por no poseer estándares definidos. ❖ Difícil integración de programadores Back end con la codificación de los componentes de Back end.
Beneficios de una solución exitosa	❖ Administración total del código generado para la comunicación contra el OS/390 o Unix. ❖ Contar con reglas y estándares de codificación para programadores Front end. ❖ Contar con herramientas útiles para el soporte de desarrollo de aplicaciones Front end. ❖ Participación activa de programadores Host o Back end en el diseño y comprensión de clases Front end. ❖ Contar con un único componente de comunicación hacia el OS/390 y Unix. ❖ Fácil integración de normas de seguridad en el componente de comunicación por ser único; por ejemplo: log de auditorias, estadísticas de uso transaccional, etc. ❖ Reducción de costos de implementación. ❖ Simplificar los procesos de monitoreo y de elaboración de reportes

Matriz de Necesidades			
Código	**Necesidad**	**Actores**	
		Analistas	**Supervisión**

N01	Estándares y normas de programación a ser implementados en desarrollos de aplicaciones .NET	X	X
N02	Componente único de comunicación contra el OS/390 y Unix desde el .NET	X	X
N03	Herramientas útiles para la codificación estándar de los componentes de persistencia a datos.	X	X
N04	Arquitectura a utilizarse para comunicación contra el OS/390 y Unix.	X	X

3. VISIÓN DE LA SOLUCIÓN.

Enunciado

*"Proporcionar estándares de codificación de programas C# de .NET acoplados a la arquitectura de comunicación planteada, utilizando recursos existentes de la Organización para finalmente d*efinir una propuesta alternativa y concreta de comunicación y acceso a la información desde el Web hacia un MAIN FRAME OS/390 y su equivalente en HP UNIX".

Objetivos		
Objetivos de Negocio	**Proporcionar estándares de codificación para el desarrollo de aplicaciones**	
Objetivos Técnicos/Procesos	**1**	Recopilar información de reglas y normas de codificación en diferentes lenguajes de programación.
	2	Analizar reglas y normas recopiladas.
	3	Definir las reglas, normas y estándares a utilizar en el desarrollo de aplicaciones.
	4	Documentar y registrar los avances.

Objetivos de Negocio	**Disponer de una alternativa mas de comunicación contra el OS/390 y Unix.**	
Objetivos Técnicos/Procesos	**1**	Diseñar una propuesta de comunicación que se acopla a las necesidades de la Organización.
	2	Utilizar las herramientas necesarias para la elaboración de la propuesta.
	3	Enfocar la propuesta al desarrollo de aplicaciones WEB con .NET enfocando el uso de los estándares definidos.
	4	Documentar y registrar los avances.

Objetivos de Negocio	**Utilizar plataformas actuales de desarrollo de aplicaciones.**	
Objetivos Técnicos/Procesos	**1**	Utilizar la herramienta de programación C# de .NET.
	2	Explotar las librerías correspondientes a sockets en .NET.
	3	Identificar los utilitarios para el desarrollo de aplicaciones acoplados a la arquitectura planteada.
	4	Documentar y registrar los avances.

Objetivos de Negocio	**Especificar el esquema de red de una agencia acoplada a la arquitectura planteada.**	
Objetivos Técnicos/Procesos	**1**	Definir los componentes que forman parte de las comunicaciones desde una Agencia al S/390 y Unix.
	2	Graficar y explicar el esquema de red planteado.
	3	Documentar y registrar los avances.

4. MODELO DE LA SOLUCIÓN.

Definir claramente los componentes que intervienen en la propuesta de comunicación hacia el OS/390 y Unix. Esta propuesta debe ser consistente con los recursos y requerimientos de la Organización en cuanto a Hardware, software.
Se trata entonces, de proporcionar un entregable en el que se definan las características principales de la propuesta, sus estándares, herramientas de soporte para desarrollo de aplicaciones acopladas a la propuesta y el esquema arquitectónico y de red para una Agencia cualquiera.

Características de la Solución

- Definiciones claras de cuál es el modelo de la propuesta de comunicación.
- Definición de reglas y estándares para el desarrollo e implementación de la propuesta.
- Descripción de ventajas y desventajas para la implementación de la propuesta.
- Definición y diseño de la arquitectura de componentes para la comunicación con el Host.
- Descripción de las herramientas utilizadas para el desarrollo de las librerías de comunicación.
- Definición y diseño de los utilitarios que darán soporte al desarrollo de aplicaciones basados en la propuesta dada.
- Estructurar pruebas iniciales de concepto de las librerías de comunicación.
- Uso de recursos de hardware y software de la organización.

Solución

Para que la solución sea exitosa, es decir, cumpla con las necesidades planteadas, es necesario que:

- La información obtenida y recopilada sea fidedigna, es decir, de fuentes propias de la Institución caso de estudio.
- Los lideres de programa conozcan la metodología de trabajo MSF así como conocimientos de los lenguajes de programación a utilizar

- La tecnología: del hardware y software sea el necesario para el desarrollo integral de la propuesta.

- Informar de los avances cronológicos de la propuesta al equipo de trabajo.

Modelo propuesto

Diagrama físico de componentes:

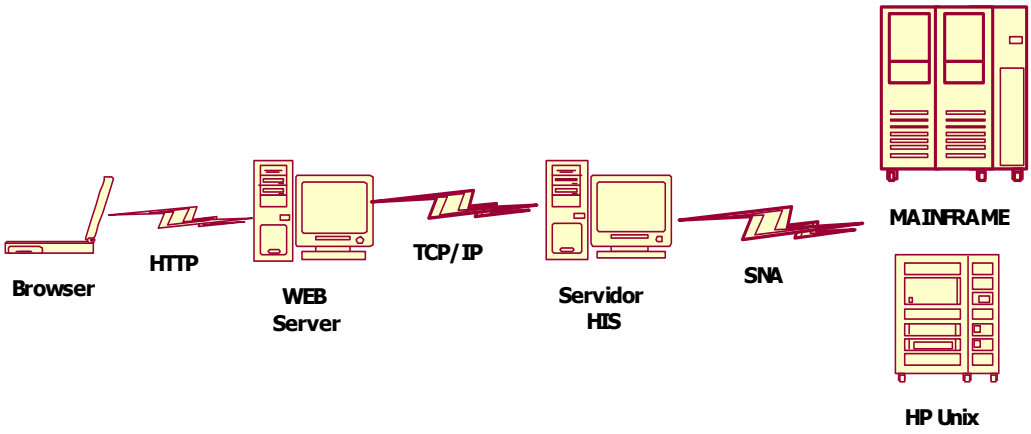
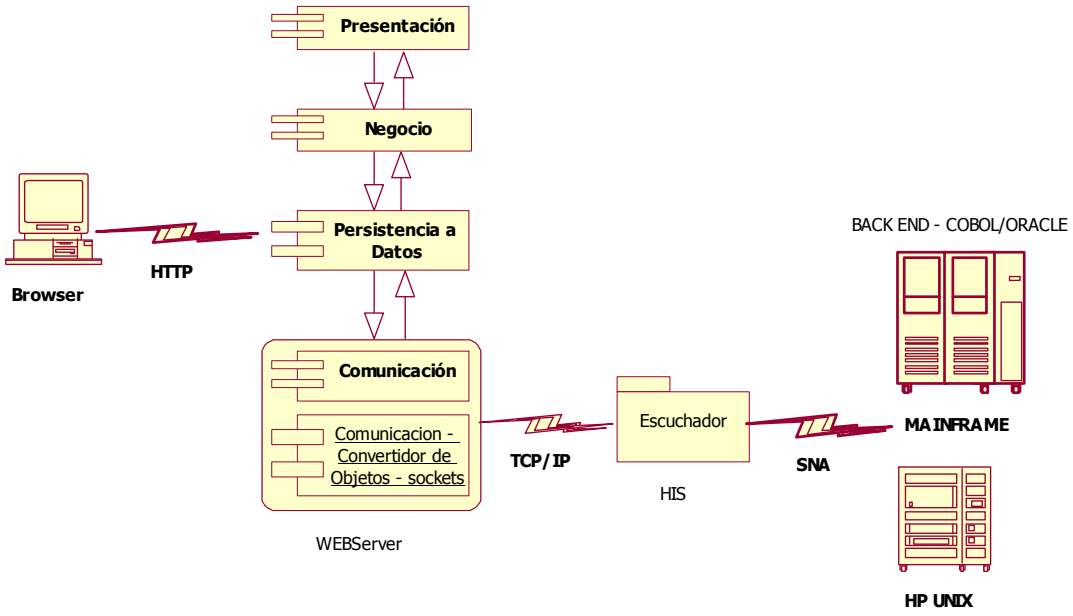


Diagrama lógico de componentes:



Matriz de Cobertura de Necesidades			
Código	**Necesidad**	**Código**	**Características**
N01	Estándares y normas de programación a ser implementados en desarrollos de aplicaciones .NET	**C1**	Definición de reglas y normas de codificación de programas que facilite una adecuada organización y administración de código fuente.

N02	Componente único de comunicación contra el OS/390 y Unix desde el .NET	**C2**	Diseño y definición de las librerías de acceso al OS/390 desde el .NET.
N03	Herramientas útiles para la codificación estándar de los componentes de persistencia a datos.	**C3**	Diseño y definición de los utilitarios que se emplearan en el desarrollo de aplicaciones bajo la plataforma de comunicación planteada.
N04	Arquitectura a utilizarse para comunicación contra el OS/390 y Unix.	**C4**	Diagrama y definición de los componentes que forman parte de la plataforma de comunicación planteada.

5. FACTORES CRÍTICOS DE ÉXITO.

Tabla de Factores
Descripción
Disponer de las herramientas críticas de programación como son: .NET, Cobol, Java.
Contar con un entorno de Desarrollo y Pruebas estables para poder desarrollar la presente propuesta.

6. QUE NO CONTEMPLA LA VISION DE LA SOLUCIÓN.

A continuación se enuncian, una serie de elementos que no serán contemplados por el proyecto, ni están incluidos en la visión del mismo:

- Instalación de software base (por ejemplo: .NET, IIS, Internet Explorer, Java) para el desarrollo de la propuesta.
- Implementación de código generado para la elaboración de la propuesta.
- Entregables de software generado con el único fin de realizar pruebas iniciales de concepto.
- Diseño de Log de aplicaciones y capa de Excepciones.
- Diseño completo de capas adicionales a la de comunicación.
- Diseño de componentes de seguridad.
- Plan de Implantación.
- Análisis de Redes.

7. IDENTIFICACIÓN DE ACTORES.

Actores	
Tipos	**Nombres y Apellidos**
Supervisor	Marco Jarrín
Analistas	Nelson Enrique Porras Vásquez
	Cristian Rubén Flores Mosquera

8. APROBACIÓN DEL DOCUMENTO.

Los abajo firmantes certifican estar de acuerdo con la información presentada en este documento:

Marco Jarrín

Líder de Producto

Nelson Porras

Líder de Programa

Cristian Flores

Líder de Programa

COMUNICACIÓN HOST

REQUERIMIENTOS

PROPUESTA DE COMUNICACIÓN HOST

CONFIDENCIAL
Plataforma: Microsoft .NET
Fecha: 02/02/2005
Versión: 1.0

CONTROL DEL DOCUMENTO.

DATOS GENERALES DEL DOCUMENTO VIGENTE			
Código	**Versión**	**Nombre**	**Autor**
RQ-CM01	1.0	Requerimientos – COMUNICACIÓN HOST.doc	Nelson Porras Christian Flores

LISTADO DE DISTRIBUCIÓN			
Empresa	**Nombre y Apellidos**	**Cargo**	**Fecha**
EPN	Marco Jarrín	Supervisor de Tesis	05/02/2005
Banco del Pichincha	Nelson Porras	Project Manager	05/02/2005
Repsolypf	Christian Flores	Project Manager	05/02/2005

REGISTROS DE CAMBIOS EN EL DOCUMENTO			
Versión	**Motivo**	**Realizado por**	**Fecha**
1.0	Documento inicial	Equipo de Trabajo	05/02/2005

CONTENIDO

(El contenido del documento original es el siguiente)

1. INTRODUCCIÓN.

El avance tecnológico de los últimos años a presionado a las empresas a migrar o desarrollar sus aplicaciones en plataformas actuales, las mismas que para su total implementación requieren de una actualización similar en cuanto a Hardware; esto repercute obviamente en el presupuesto de cualquier empresa cuyas intenciones sea la de actualizar sus procesos y equipos tecnológicos.

Hoy por hoy es mucho mas apetecible el tener una propuesta tecnológica cuyos pre-requisitos de Hardware y Software no sobrepase el presupuesto de una empresa por mas grande que esta sea. Por lo tanto, la presente propuesta dará a conocer de alguna manera los beneficios que tendrá su implementación.

Este es el caso de la solución informática que actualmente se tiene en el Banco del Pichincha, la misma que involucra costos extremadamente altos, lo cual desvirtúa la razón misma de ser de un Banco. Adicionalmente, el Banco del Pichincha se ha embarcado en una serie de proyectos con el fin de reducir costos mediante la optimización de sus procesos; por tal motivo, la presente propuesta pretende apoyar el objetivo enmarcado por la Institución mediante el diseño y estandarización de clases y librarías de comunicación con las que funcionara el core tecnológico del Banco desarrolladas con herramientas de punta, utilizando los recursos tecnológicos que ya posee la Organización.

El presente proyecto busca diseñar una capa de comunicación que conecte el .NET con el MainFrame y su equivalente Unix.

2. CARACTERÍSTICAS GENERALES DE LA SOLUCIÓN.

Matriz de Características	
Código	**Características**
C1	Definición de reglas y normas de codificación de programas que facilite una adecuada organización y administración de código fuente.
C2	Diseño y definición de las librerías de acceso al S/390 desde el .NET.
C3	Diseño y definición de los utilitarios que se emplearan en el desarrollo de aplicaciones bajo la plataforma de comunicación planteada.
C4	Diagrama y definición de los componentes que forman parte de la plataforma de comunicación planteada.

3. CASOS DE USO.

Diagrama de Casos de Uso/ Arquitectura de la Plataforma

CASO DE USO - COMUNICACION HOST

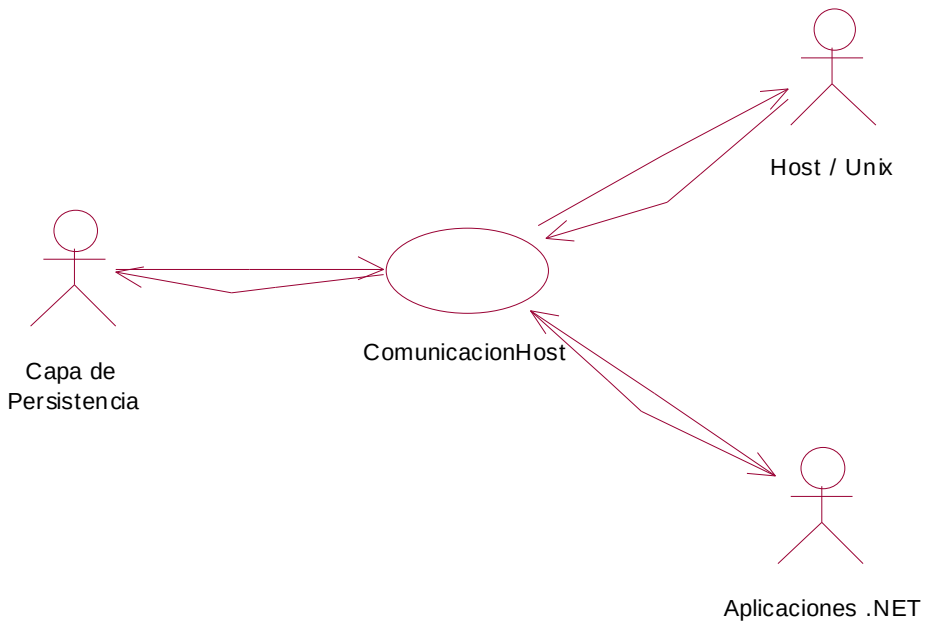


DIAGRAMA DE CLASES - CAPA DE COMUNICACION

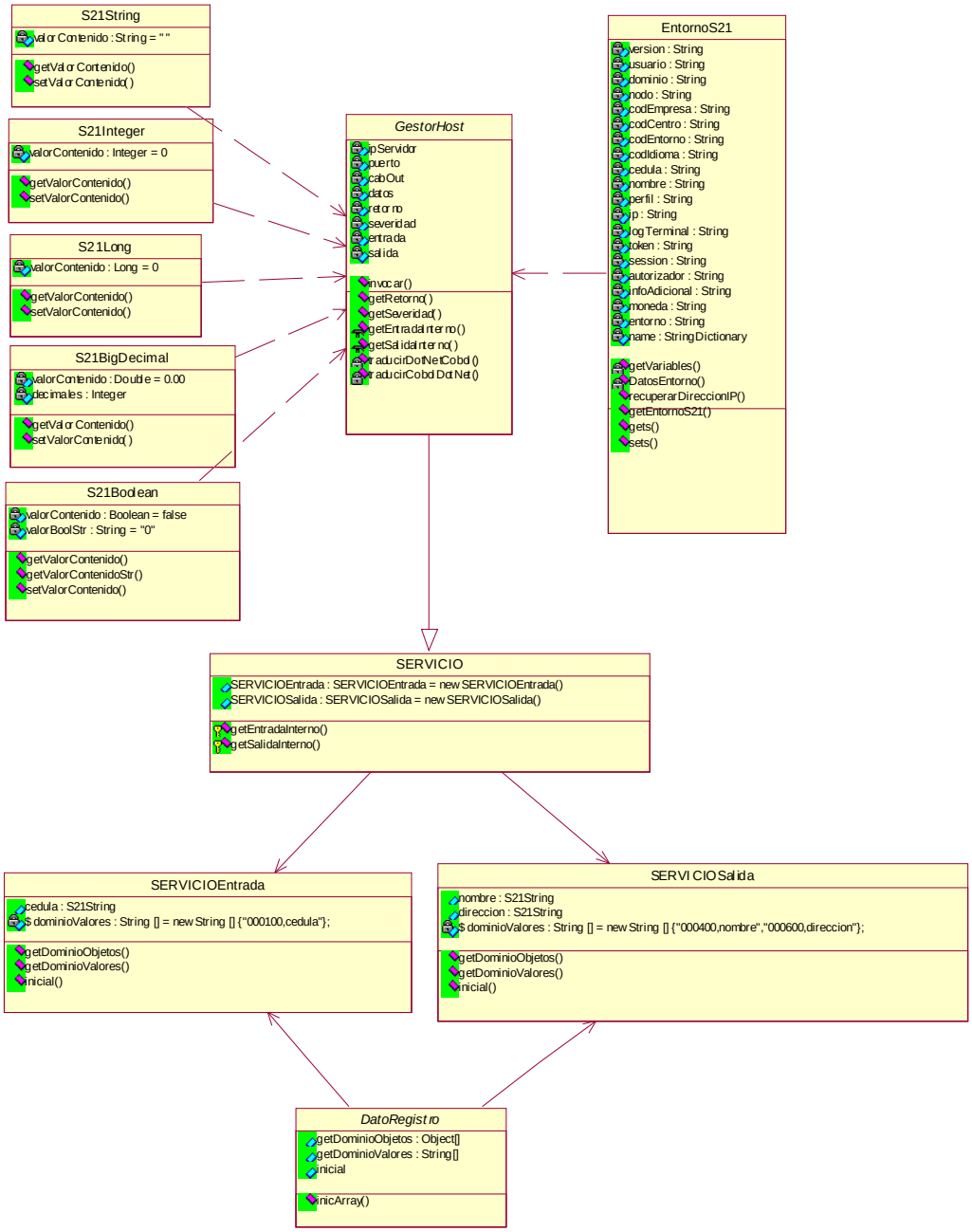
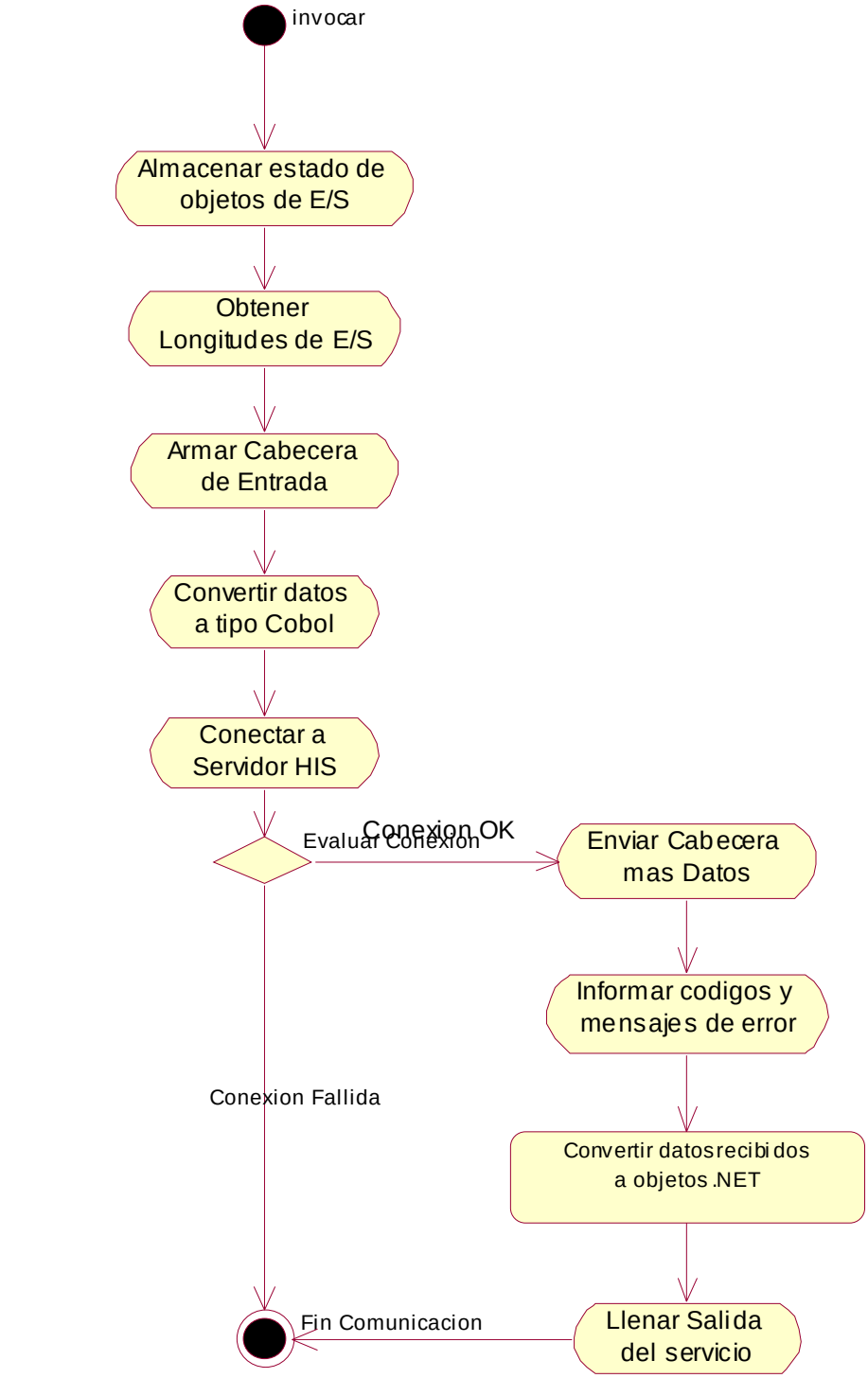


DIAGRAMA DE ACTIVIDAD - COMUNICACION HOST



CASO DE USO - UTILITARIO DE GENERACION DE CODIGO

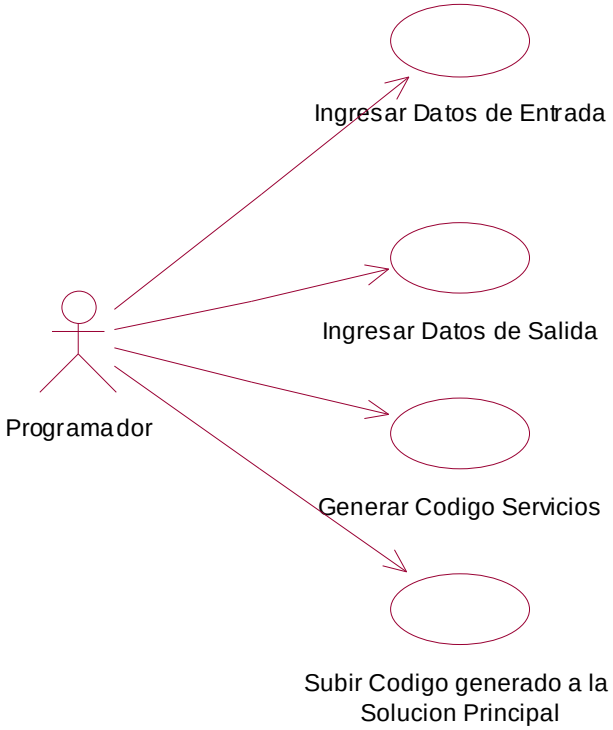
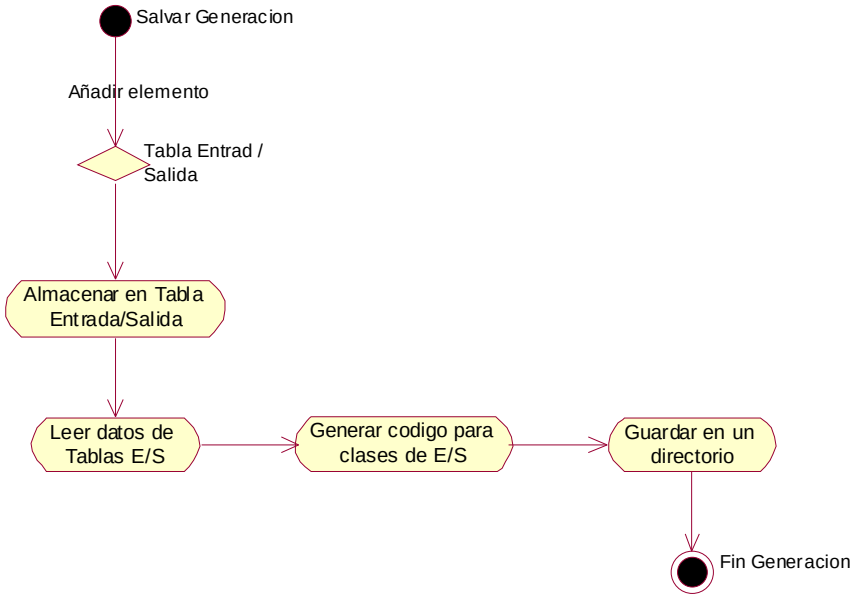


DIAGRAMA DE ACTIVIDAD - UTILITARIOS



Descripción de los Actores Involucrados		
Tipo	**Actor**	**Descripción**
Sistema	Capa de Persistencia	Capa que almacena los servicios que contienen los datos de entrada y salida que serán transmitidos y recibidos desde el Host.
Sistema	Aplicaciones .NET	Todas las aplicaciones del Banco del Pichincha se acoplaran a la arquitectura de comunicación planteada
Sistema	HOST / Unix	Donde se encuentra la capa de Negocio propiamente dicha y el acceso a los datos (Oracle)
Persona	Programador	Opera el sistema para la generación de código

Descripción de los Casos de Uso

CASO DE USO:	COMUNICACIÓN HOST		
Código:	CU1	**Nombre:**	**COMUNICACIÓN HOST**
Descripción:	Caso de uso para describir el objetivo del componente de comunicación		
Código Característica(s) de la solución asociada:	• Convertidor de tipo de datos .NET a Cobol • Convertidor de tipo de datos Cobol a objetos .NET • Traductor de caracteres ASCII y EBCDIC • Formateador de trama de datos • Conexión con HIS		
Precondiciones:	Implementar los métodos abstractos de los componentes de comunicación		
Post condiciones:			

Eventos	**Flujo Básico**		**Flujo Alternativo**	
	1	Armar cabeceras y convertir objetos a tipo de datos Cobol	**A1**	N/A
	2	Enviar mensaje a servidores HIS	**A2**	N/A
	3	Recibir mensaje, convertir datos a Objetos y llenar datos de salida	**A3**	N/A

CASO DE USO:	UTILITARIOS DE GENERACIÓN DE CODIGO		
Código:	CU2	**Nombre:**	**UTILITARIOS**
Descripción:	Caso de uso para describir el objetivo del utilitario para la generación de código para la capa de persistencia a datos		

Código Característica(s) de la solución asociada:	• Recepta y almacena los datos de entrada y salida que conforman el área de comunicación hacia y desde el Host. • Genera el código para las clases de Entrada y Salida de Datos • Almacena en un directorio las clases generadas
Precondiciones:	Implementar los métodos abstractos de los componentes de comunicación
Post condiciones:	Subir el código generado a la solución definitiva.

Eventos	**Flujo Básico**		**Flujo Alternativo**	
	1	Añadir atributos de E/S y almacenar en Tablas	**A1**	N/A
	2	Leer datos almacenados	**A2**	N/A
	3	Generar código para clases de E/S y almacenarlos en un directorio.	**A3**	N/A

4. DIAGRAMA DE RELACIÓN CARACTERÍSTICAS Y CASOS DE USO.

n/a

5. REQUERIMIENTOS FUNCIONALES.

Código		Requerimiento	
Caso De Uso	**Requerimiento**	**Nombre**	**Descripción**
CU1	**RF1.0**	Conversión de Tipo de datos	Procedimientos para convertir tipos de datos u objetos .NET (String, Integer, Long, etc.) en su equivalentes Cobol (PIC X(n), PIC 9(n), PIC 9(n) COMP-3, etc), y viceversa
	RF1.1	Traducción de tipos de datos	Procedimientos que traduzcan los caracteres almacenados en los atributos de los objetos de entrada y salida (ASCII – EBCDIC)
	RF1.2	Formateador de trama de datos	Obtener la cabecera de entrada y acoplarla al mensaje de la trama.
	RF1.3	Conexión con HIS	Enviar y recibir los datos
CU2	**RF2.0**	Almacenar la meta data de áreas de comunicación.	Recepta y almacena los datos de entrada y salida que conforman el área de comunicación hacia y desde el Host.
	RF2.1	Generar el código de clases de E/S de Datos	Construir el código que conformaran las clases de entrada y salida de un servicio especifico
	RF2.2	Almacenar clases generadas	Guardar el código generado en un directorio de trabajo

6. REQUERIMIENTOS SUPLEMENTARIOS.

Requerimientos de Usabilidad			
Código		**Requerimiento**	
Caso de Uso	**Requerimiento Suplementario**	**Nombre**	**Descripción**
CU1	**RS/ U1.0**	De fácil implementación	Código independiente que no necesita que los programadores conozcan su implementación.
CU2	**RS/ U2.0**	De fácil implementación y uso	Interfases claras

Requerimientos de Disponibilidad			
Código		**Requerimiento**	
Caso de Uso	**Requerimiento Suplementario**	**Nombre**	**Descripción**

CU1	**RS/ DI1.0**	Siempre Operativo	Siempre debe estar activo
CU2	**RS/ DI2.0**	Transportable	Solo requiere de un ejecutable

Requerimientos de Rendimiento			
Código		**Requerimiento**	
Caso de Uso	**Requerimiento Suplementario**	**Nombre**	**Descripción**
CU1	**RS/ RE1.0**	Optimo	Igualar o superar el rendimiento actual
CU2	**RS/RE2.0**		

Requerimientos Legales			
Código		**Requerimiento**	
Caso de Uso	**Requerimiento Suplementario**	**Nombre**	**Descripción**
CU1	**RS/LE1.0**		
CU2	**RS/LE2.0**		

Requerimientos asociados al proceso del Cliente			
Código		**Requerimiento**	
Caso de Uso	**Requerimiento Suplementario**	**Nombre**	**Descripción**
CU1	**RS/PC1.0**		
CU2	**RS/PC2.0**		

Requerimientos de Seguridad			
Código		**Requerimiento**	
Caso de Uso	**Requerimiento Suplementario**	**Nombre**	**Descripción**
CU1	**RS/S1.0**		
CU2	**RS/S2.0**		

Requerimientos de Componentes Externos o Proveedores			
Código		**Requerimiento**	
Caso de Uso	**Requerimiento Suplementario**	**Nombre**	**Descripción**
CU1	**RS/CE.P1.0**		

Requerimientos de Mantenimiento	
Código	**Requerimiento**

Caso de Uso	Requerimiento Suplementario	Nombre	Descripción
CU1	RS/M1.0	Comprensible	De fácil entendimiento
CU2	RS/M2.0	Comprensible	De fácil entendimiento

Requerimientos de Licenciamiento			
Código		Requerimiento	
Caso de Uso	Requerimiento Suplementario	Nombre	Descripción
CU1	RS/LI1.0		
CU2	RS/LI2.0		

Requerimientos de Plataforma			
Código		Requerimiento	
Caso de Uso	Requerimiento Suplementario	Nombre	Descripción
CU1	RS/P1.0	.Net con C#	Codificar en C#
CU2	RS/P2.0	.Net con C# y en Windows Forms	Codificar en C# con Windows Forms

Requerimientos de Conectividad			
Código		Requerimiento	
Caso de Uso	Requerimiento Suplementario	Nombre	Descripción
CU1	RS/C1.0		
CU2	RS/C2.0		

Requerimientos de Documentación			
Código		Requerimiento	
Caso de Uso	Requerimiento Suplementario	Nombre	Descripción
CU1	RS/DO1.0		
CU2	RS/DO2.0		

Requerimientos de Tiempo			
Código		Requerimiento	
Caso de Uso	Requerimiento Suplementario	Nombre	Descripción
CU1	RS/T1.0		

CU2	RS/T2.0		

Requerimientos de Estándares			
Código		Requerimiento	
Caso de Uso	Requerimiento Suplementario	Nombre	Descripción
CU1	RS/ES1.0		
CU2	RE/ES2.0		

Requerimientos de Entorno			
Código		Requerimiento	
Caso de Uso	Requerimiento Suplementario	Nombre	Descripción
CU1	RS/EN1.0		
CU2	RS/EN2.0		

Requerimientos asociados a Reportes			
Código		Requerimiento	
Caso de Uso	Requerimiento Suplementario	Nombre	Descripción
CU1	RS/REP1.0		
CU2	RS/REP2.0		

Requerimientos asociados a Estadísticas			
Código		Requerimiento	
Caso de Uso	Requerimiento Suplementario	Nombre	Descripción
CU1	RS/EST1.0		
CU2	RS/EST2.0		

Requerimientos asociados a Formatos			
Código		Requerimiento	
Caso de Uso	Requerimiento Suplementario	Nombre	Descripción
CU1	RS/F1.0		
CU2	RS/F2.0		

Requerimientos de Diseño Grafico			
Código		**Requerimiento**	
Caso de Uso	**Requerimiento Suplementario**	**Nombre**	**Descripción**
CU1	**RS/DG.1.0**		
CU2	**RS/DG.2.0**		

7. REQUERIMIENTOS EXCLUIDOS.

Requerimiento		Justificación
Código	**Nombre**	
RS/LE1.0 RS/LE2.0	Requerimientos Legales	N/A
RS/PC1.0 RS/PC2.0	Requerimientos asociados al proceso del Cliente	N/A
RS/PC1.0 RS/PC2.0	Requerimientos de Seguridad	N/A
RS/CE.P1.0 RS/CE.P2.0	Requerimientos de Componentes Externos o Proveedores	N/A
RS/LI1.0 RS/LI2.0	Requerimientos de Licenciamiento	N/A
RS/C1.0 RS/C2.0	Requerimientos de Conectividad	N/A
RS/DO1.0 RS/DO2.0	Requerimientos de Documentación	N/A
RS/T1.0 RS/T2.0	Requerimientos de Tiempo	N/A
RS/ES1.0 RE/ES2.0	Requerimientos de Estándares	N/A
RS/EN1.0 RS/EN2.0	Requerimientos de Entorno	N/A
RS/EST1.0 RS/EST2.0	Requerimientos asociados a Estadísticas	N/A
RS/F1.0 RS/F2.0	Requerimientos asociados a Formatos	N/A
RS/DG.1.0 RS/DG.2.0	Requerimientos de Diseño Grafico	N/A

8. ATRIBUTOS DE LOS REQUERIMIENTOS.

Còdigo	Requerimiento Nombre	20 P	20 I	20 R	20 C	20 F	Σ	%
RF1.0	Conversión de tipo de datos	5	5	5	5	5	**25**	**100**
RF1.1	Traducción de tipo de datos	5	5	5	5	5	**25**	**100**
RF1.2	Formateador de trama de datos	5	5	5	2	5	**22**	**88**
RF1.3	Conexión con HIS	5	5	5	3	5	**23**	**92**
RF2.0	Almacenar Meta Data	3	3	3	4	5	**18**	**72**
RF2.1	Generar código de clases de E/S	3	3	3	4	5	**18**	**72**
RF2.2	Almacenar clases generadas	3	3	3	4	5	**18**	**72**

Escala:Muy Alta (5)/ Alta (4)/ Media(3)/ Baja (2)/ Muy Baja(1)

P: Prioridad/ I: Impacto/ R: Riesgo/ C: Complejidad/ F: Factibilidad

9. PREMISAS.

- No se publicara información confidencial
- Los equipos de trabajo de la institución estarán a disposición para el proyecto en horas no laborables

10. PERSONAS DE CONTACTO DEL CLIENTE.

Nro.	Nombre y Apellidos	Área en la que labora	Información que suministra
1	Nelson Porras	Tecnología	Arquitectura del sistema actual
2			
3			
4			
5			
6			
7			
8			
9			

11. LINEAMIENTOS PARA EL CONTROL DE CAMBIOS.

- Cualquier requerimiento adicional afectará directamente a tiempos de entrega.
- Los supervisores del proyecto, y el equipo de pruebas son los únicos que podrán solicitar cambios.
- El documento de control de cambio deberá contener mínimo los siguientes datos: Fecha de Solicitud; Responsable, Tema, Descripción.

12. TÉRMINOS CLAVES.

CICS (Customer Information Control System). Sistema transaccional bajo el cual se ejecuta transacciones y programas dentro de un mainframe.

COMTI (COM Transaction Integration). Herramienta desarrollada por Microsoft para integrar la tecnología COM con los productos y sistemas de tercera generación.

MAINFRAME (Equipo). Usado para definir equipos de gran tamaño, pueden llegar a ocupar un cuarto. Es un término comúnmente usado para definir los los grandes equipos de IBM.

.NET (Dot Net). Arquitectura de Microsoft para aplicaciones basadas en Internet.

TCP/IP (Transmisión Control Protocol/Internet Protocol). Protocolo de control de transmisión / Protocolo Internet; es un conjunto de protocolos utilizado en internet generalizándose su uso para la interconexión de redes heterogéneas

SNA (System Network Architecture). Arquitectura de comunicación de datos para equipos IBM, que definen los niveles de protocolos para comunicar terminales y aplicaciones y entre programas.

13. APROBACIÓN DEL DOCUMENTO.

Los abajo firmantes certifican estar de acuerdo con la información presentada en este documento:

Santiago Bustillos López	Nelson Porras Vásquez
Representante del Cliente	**Gerente de Programa**
Representante del Grupo de Usuarios	**Representante del Proveedor**
Representante del personal Técnico	

COMUNICACIÓN HOST

Especificaciones Funcionales

PROPUESTA DE COMUNICACIÓN HOST

CONFIDENCIAL
Plataforma: Microsoft .NET
Fecha: 02/02/2005
Versión: 1.0

CONTENIDO

(El contenido del documento original es el siguiente)

Especificaciones Funcionales

Visión

El avance tecnológico de los últimos años a presionado a las empresas a migrar o desarrollar sus aplicaciones en plataformas actuales, las mismas que para su total implementación requieren de una actualización similar en cuanto a Hardware; esto repercute obviamente en el presupuesto de cualquier empresa cuyas intenciones sea la de actualizar sus procesos y equipos tecnológicos.

Hoy por hoy es mucho mas apetecible el tener una propuesta tecnológica cuyos pre-requisitos de Hardware y Software no sobrepase el presupuesto de una empresa por mas grande que esta sea. Por lo tanto, la presente propuesta dará a conocer de alguna manera los beneficios que tendrá su implementación.

Este es el caso de la solución informática que actualmente se tiene en el Banco del Pichincha, la misma que involucra costos extremadamente altos, lo cual desvirtúa la razón misma de ser de un Banco. Adicionalmente, el Banco del Pichincha se ha embarcado en una serie de proyectos con el fin de reducir costos mediante la optimización de sus procesos; por tal motivo, la presente propuesta pretende apoyar el objetivo enmarcado por la Institución mediante el diseño y estandarización de clases y librarías de comunicación con las que funcionara el core tecnológico del Banco desarrolladas con herramientas de punta, utilizando los recursos tecnológicos que ya posee la Organización.

El presente proyecto busca diseñar una capa de comunicación que conecte el .NET con el MainFrame y su equivalente Unix.

Metas

Metas de Diseño

- Establecer normas y estándares de programación.
- Diseñar una arquitectura que evite el cambio de funcionalidad técnica en el Back End; es decir, que no altere lo que actualmente funciona en el Host.
- Basar el diseño en base a los recursos que actualmente dispone la Organización.
- Diseñar la solución en capas dando mayor énfasis a la capa de comunicación.
- Utilizar los estándares de programación definidos para la solución.
- Definir el modelo de clases que tendrá la solución.
- Construir el modelo de comunicación propuesto con el fin de evaluar factibilidad del proyecto mediante pruebas de concepto.
- Construir los utilitarios para el desarrollo de aplicaciones basados en la arquitectura de comunicación propuesta.
- Definir el esquema inicial arquitectónico de la aplicación así como también el esquema para una agencia piloto.

Metas de Usabilidad

- Proveer de librerías únicas de comunicación en .NET hacia el Host
- Facilitar el uso transparente de las librerías de Comunicación.
- Facilidad de incorporar log transaccionales así como también logs de auditoria.
- Estandarización en la codificación de programas para el desarrollo de nuevas aplicaciones.
- Desarrollo rápido de aplicaciones mediante el uso de los utilitarios definidos
- Administración y control total de las programas fuentes de las librerías de comunicación y utilitarios.
- Fácil integración de conceptos para programadores .NET y Cobol.
- Cero adecuaciones a la estructura del Back End en la implementación de la solución propuesta.

Servicios y Componentes

Comunicación mediante Sockets.
Permitirá conectarse a los servidores HIS para el envió y recepción de tramas de datos.

Conversión de tipos de datos.
Permitirá traducciones de objetos (.NET) a Cobol y viceversa.

Estandarización de la capa de persistencia a datos.
Incorpora utilitarios la generación de código evitando la manipulación del mismo.

Uso de normas y estándares definidos.
Presenta reglas y estándares de codificación recopilados de algunos lenguajes de programación.

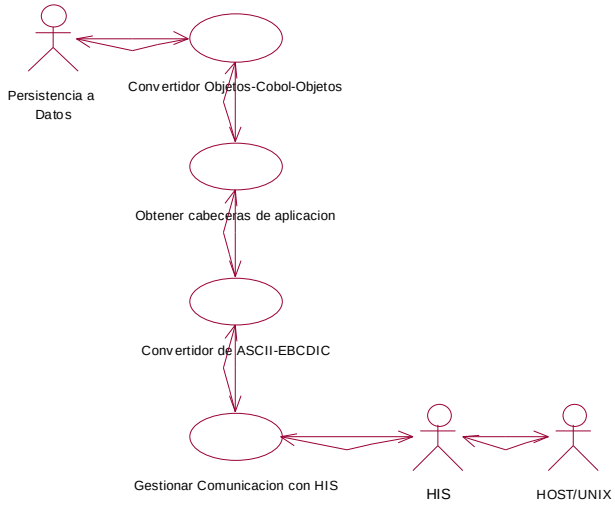
Software Back End.

Evita la reestructuración de programas Cobol que actualmente funcionan sin problema.

Estructura del Diseño

Diseño Conceptual

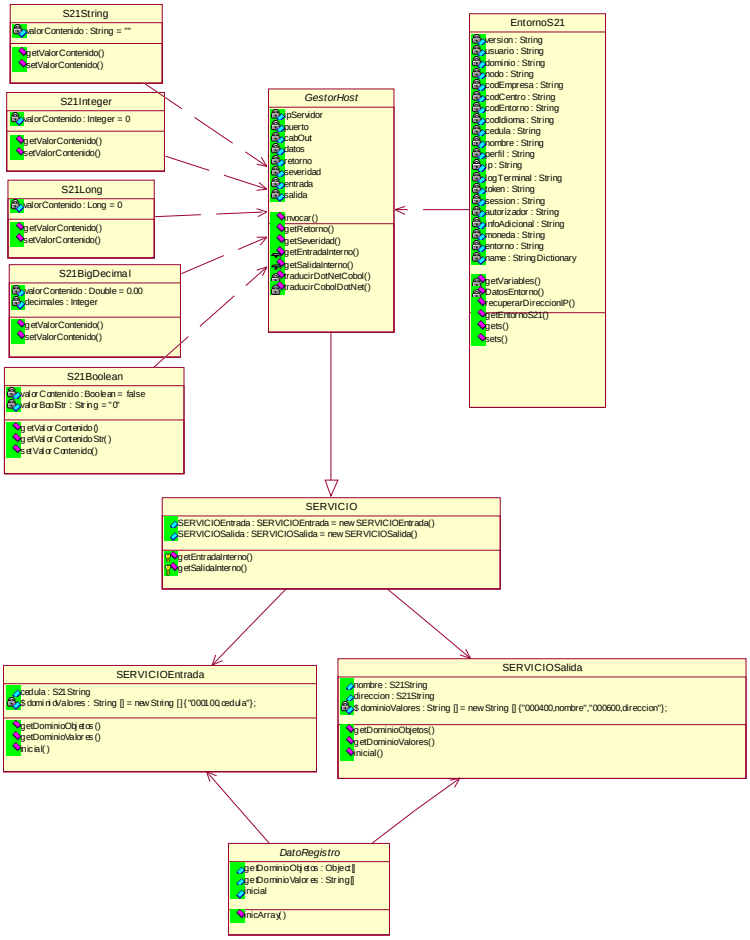
Sistema Actual	**Sistema Propuesto**	**Roles**
No Aplica (Construcción paralela de una propuesta de Comunicación Host utilizando COMTI's de Microsoft).	Desarrollo del Front End(.NET) para los diferentes módulos del core Bancario (Activos, Pasivos, Contabilidad, etc) Integrando el Host con la Web mediante librerías de comunicación únicas; facilitando su desarrollo mediante utilitarios y evitando el rediseño del Back End.	**Administración de Software:** Administrar el código estándar desarrollado Programadores .NET Acoplarse al estándar definido y utilizar las librerías de comunicación transparentemente.

Descripción del Proceso	Diagrama de Procesos
El componente de comunicación se encargara de las siguientes gestiones: ❖ Recibe desde la capa de persistencia un objeto con estado denominado servicio que contiene los atributos o campos de entrada y salida que serán enviados y recibidos hacia y desde el Host respectivamente. Esta clase será clase hijo al componente de comunicación. ❖ El contenido del objeto será traducido a su equivalente en tipos de datos Cobol. ❖ Luego, por medio de los métodos heredados se obtendrán las cabeceras de entrada utilizadas para ejecutar el respectivo programa en Host y se definirá el mensaje completo conformado por la cabecera y el cuerpo. ❖ Finalmente, el mensaje será traducido a ASCII o EBCDIC según el caso, y convertido a bytes para ser enviado mediante sockets a los servidores HIS, quienes reenviaran dicho mensaje al HOST. ❖ El proceso de recepción de datos desde el Host, implementa la misma lógica anterior pero en sentido contrario.	

Diseño Lógico

A continuación se presenta el diseño lógico de la solución planteada, basada en el diseño conceptual:

DIAGRAMA DE CLASES - CAPA DE COMUNICACION



El diseño se basa en cuatro clases que representan los tipos de datos que se manejará en la solución; estas clases serán independientes. La clase EntornoS21 es una clase con estado la cual facilita toda la información de cabecera de entrada para enviar al Host; la particularidad de esta clase es que se la debe llenar al inicio de la sesión del usuario, es decir, en el logon.
La clase DatoRegistro es una clase Abstracta que define tres métodos para ser implementados por las clases que la hereden; su finalidad es la de poder acceder desde la clase GestorHost a la meta data de los atributos de un servicio. La clase GestorHost, es la clase principal del componente de comunicación; su definición es abstracta, definiendo dos métodos cuyo propósito es la de obtener el estado de los datos de entrada y salida de un servicio; además, implementa métodos privados para realizar la conversión de datos respectiva, y finalmente implementa su método principal llamado invocar, en el que se arma la cabecera y el cuerpo del mensaje para luego abrir comunicación con los servidores HIS y enviar el contenido del mensaje. Finalmente, para acoplarse a la arquitectura planteada, se tiene los denominados servicios que básicamente heredan de la clase GestorHost, implementando sus métodos abstractos e instanciando sus dos atributos principales, los mismos que son una instancia de las clases de entrada y salida en las cuales se definen los atributos que serán enviados y llenados hacia y desde el Host respectivamente; estas clases heredan de la clase DatoRegistro implementando sus métodos abstractos.

A continuación se muestra el diseño del utilitario planteado para apoyar al desarrollo de aplicaciones basadas en la propuesta planteada:

CASO DE USO - UTILITARIO DE GENERACION DE CODIGO

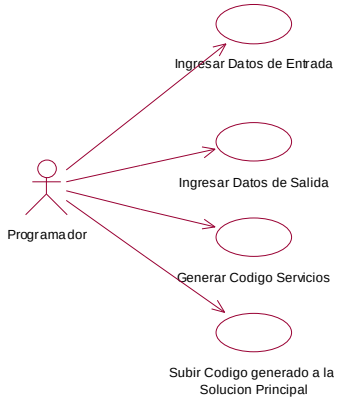
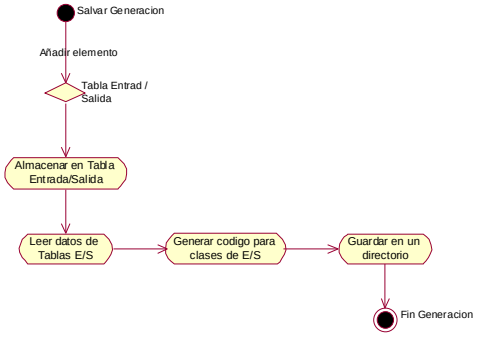
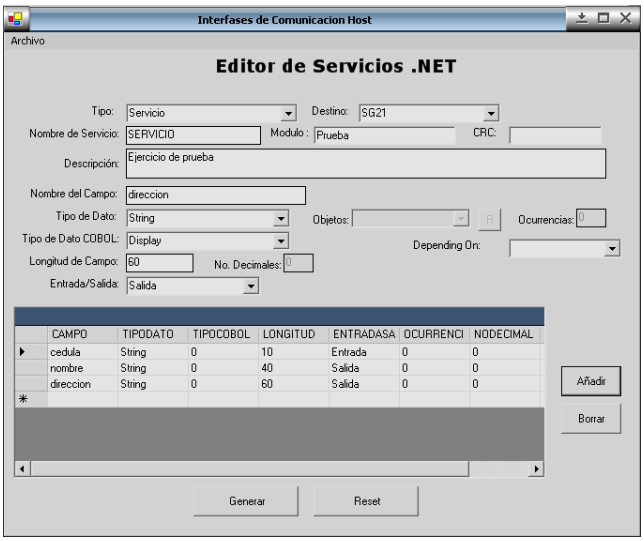


DIAGRAMA DE ACTIVIDAD - UTILITARIOS



INTERFASE DE USUARIO



La implementación de esta ayuda, se basa en Windows Forms de .Net, su

uso es la de generar código para la capa de persistencia a datos usando la arquitectura de comunicación propuesta. Su uso es muy simple; se basa en ingresar los campos de entrada y salida que tendrá un servicio; básicamente es la meta data del área de comunicación que será transportada hacia y desde el Host, en el que se define los tipos de datos Cobol con su equivalencia en .NET, también se definen las longitudes de los campos, así como también se indica si el servicio manejara ocurrencias.

Diseño Físico

[Esta sección describe donde reside actualmente el Sistema en cuanto a plataforma se refiere, también se describen cada una de las especificaciones necesarias que debe cumplir la nueva plataforma y las configuraciones mínimas recomendadas en cuanto a software y hardware.]

La implementación de la presente propuesta residirá en primera instancia en el servidor de desarrollo ubicado en el Centro de Computo del Banco del Pichincha; para luego ser distribuido al servidor de Test y finalmente a los servidores de producción.

Requerimientos de Hardware / Software.

SERVIDOR WEB

- *Sistema Operativo:*
  - *Nombre: Windows.*
  - *Versión: 2000 Server.*
  - *Service Pack : SP3.*
  - *Lenguaje: English (United States).*
- *Servidor WEB:*
  - *Nombre: Internet Information Server*
  - *Versión: 5.0*
- *Componentes:*
  - *Nombre: Framework .NET*
  - *Versión: 1.0 o mayor.*
- *Browser:*
  - *Nombre: Internet Explorer*
  - *Versión: 6.0.2800 o mayor*
  - *Service Pack: SP1 o mayor*

COMUNICACIÓN HOST

Documento Maestro

PROPUESTA DE COMUNICACIÓN HOST

CONFIDENCIAL
Plataforma: Microsoft .NET
Fecha: 02/02/2005
Versión: 1.0

Contenido

(El contenido del documento original es el siguiente)

Plan de Pruebas

¿Qué persigue?

Ejecutar el plan que contiene las especificaciones de prueba permite garantizar que el sistema cumpla con los requerimientos establecidos y acordados entre el Cliente y el equipo de desarrollo, atendiendo ciertos parámetros de calidad, funcionamiento y operatividad esperados en una solución desarrollada en Microsoft .NET con C#.

Básicamente se busca garantizar que se ejecute correctamente:

- Una transacción en el Host.
- Que los datos retornados sean consistentes.
- Igualar o mejorar el rendimiento de la aplicación actual.

Recursos Necesarios

Para el establecimiento de casos de prueba que puedan servir en el monitoreo de la consistencia de datos en la base, se deben tener los siguientes recursos:

1. A continuación se detalla los datos de los usuarios necesarios para realizar las pruebas respectivas:

USUARIO	EMPRESA	CENTRO	PERFIL	DESCRIPCION
SBUSTIL	0010	0066	ASEBATOT	Usr. de Negocio
NPORRAS	0010	0012	ASEBATOT	Usuario Técnico

Donde: 0010 = Banco del Pichincha
0066 = Agencia Conocoto
ASEBATOT = Perfil con acceso total.

2. El lugar desde donde se va a ejecutar la aplicación es el Edificio de Sucursal Norte, Segundo Piso, Área de Tecnología.

3. Usuario de Oracle para comprobar consistencia de datos:
Usuario = M2D
Password = confirmado en el momento de las pruebas

4. Los usuarios y contraseñas adicionales requeridos conformen se necesiten en las pruebas, serán definidos y entregados en su momento por Nelson Porras.

Casos de Prueba

Los casos de prueba que se detallan a continuación son los necesarios a ejecutar para la solución desarrollada:

- Pruebas Funcionales
- Pruebas de Infraestructura
- Pruebas de Estrés

A continuación el detalle de cada una:

Pruebas Funcionales

Son todas aquellas pruebas que indican si el sistema está funcionando según las necesidades y requerimientos postulados en el documento de visión; son *Pruebas Funcionales*

Este tipo de pruebas se divide en dos:

1. Pruebas de Usabilidad
2. Pruebas de Operatividad

1.- Pruebas de Usabilidad

Las pruebas de usabilidad se realizan tomando en cuenta los distintos tipos de usuarios y sus roles en el sistema con el fin de asegurar que el comportamiento del sistema va acorde con las necesidades y el nivel de cada usuario. Las pruebas de usabilidad están concentradas en verificar las siguientes características: amigabilidad, consistencia en el funcionamiento, estándares e interacción. Por otra parte, como se menciono en el documento de Visión, la solución planteada no contempla la capa de presentación; sin embargo se ha desarrollará una pagina .aspx que facilite las pruebas de usabilidad en cuanto a verificar si la data retornada es consistente y cumple con lo que se esperaba. El resto de este tipo de pruebas como: Disponibilidad de botones de acción y edición de campos, validación de campos, estándares de interfaz, etc., son irrelevantes para el proyecto, tomando en cuenta que solo es una prueba de concepto.

2.-Pruebas de Operatividad

Las pruebas de Operatividad se realizan tomando en cuenta los siguientes factores: de que manera fue desarrollado el sistema o aplicativo, cuales son los objetivos del sistema o aplicativo, y cuales deben ser los resultados que el sistema o aplicativo le ofrezca al usuario.

Verificación de resultados: Es revisar los resultados de cada uno de los procesos que realice el aplicativo.
La verificación de resultados se lo hará comparando los resultados del aplicativo contra los datos de la base; los resultados serán descritos en una hoja Excel para luego corregir las fallas si las hubiese.

Los resultados para el conjunto de estas pruebas se detallan en el documento de plan de pruebas.

Pruebas de Infraestructura

El objetivo de este tipo de pruebas es verificar que la infraestructura que soporta el Sistema de **Comunicación Host** tenga un desempeño satisfactorio en el momento que se ejecute la aplicación de manera que pueda sostener tanto la aplicación en prueba como el resto de las actividades que el ambiente necesite realizar en ese momento.

Se enfocará entonces las pruebas con lo que tiene que ver con

Pruebas de Comunicación: Esta prueba consiste en medir los tiempos de conexión, el desempeño de la aplicación, la sincronización de la data y su consistencia.

Los resultados para el conjunto de estas pruebas se detallan en el documento de plan de pruebas., sin embargo se muestra un detalle de lo realizado:

Properties

Test type:	Dynamic
Simultaneous browser connections:	10
Warm up time (secs):	0
Test duration:	00:00:01:00
Test iterations:	3
Detailed test results generated:	Yes

Summary

Total number of requests:	3
Total number of connections:	3
Average requests per second:	0,05
Average time to first byte (msecs):	36.637,00
Average time to last byte (msecs):	36.641,00
Average time to last byte per iteration (msecs):	36.641,00
Number of unique requests made in test:	1
Number of unique response codes:	1

Errors Counts

HTTP:	0
DNS:	0
Socket:	0

Additional Network Statistics

Average bandwidth (bytes/sec):	5.527,18
Number of bytes sent (bytes):	1.053
Number of bytes received (bytes):	330.578
Average rate of sent bytes (bytes/sec):	17,55
Average rate of received bytes (bytes/sec):	5.509,63
Number of connection errors:	0

Number of send errors:	0
Number of receive errors:	0
Number of timeout errors:	0

Response Codes

Response Code: 200 - The request completed successfully.

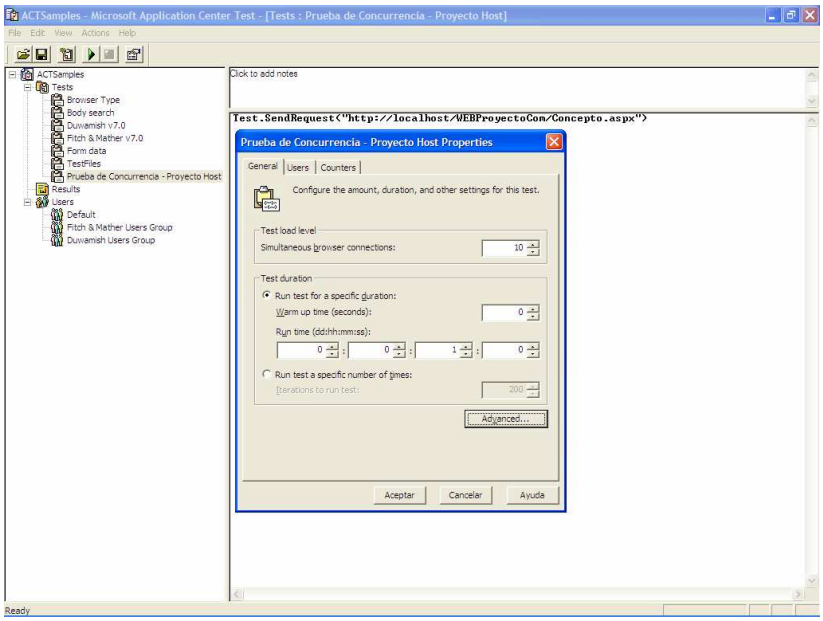
Count:	3
Percent (%):	100,00

Pruebas de Estrés

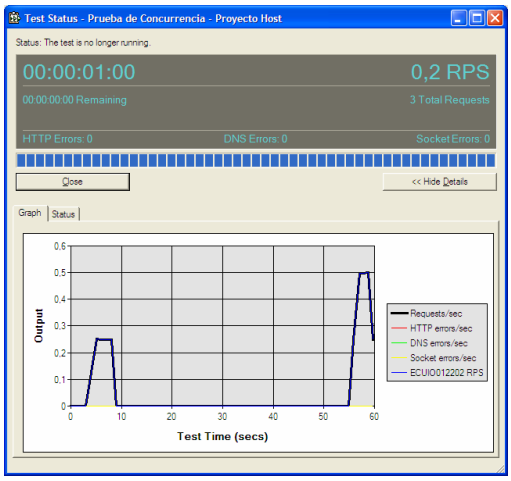
Consisten en pruebas internas que validen el acceso de la aplicación de manera concurrente de varios usuarios, para comprobar la efectividad y tiempo de respuestas de los mismos. En este tipo de pruebas se evalúa el desempeño de la aplicación bajo el uso de múltiples usuarios simultáneos o usando shells que simulen dicho ambiente. El tiempo de respuesta del sistema puede observarse con el acceso simultáneo de diversos usuarios.

Los resultados para el conjunto de estas pruebas se detallan en el documento de matriz de pruebas., sin embargo se muestra un detalle de lo realizado:

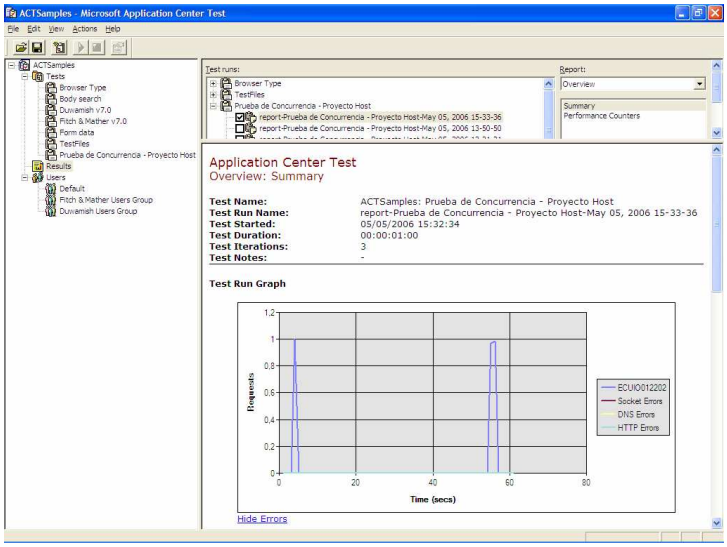
1. Parametrización de las pruebas de Estrés:



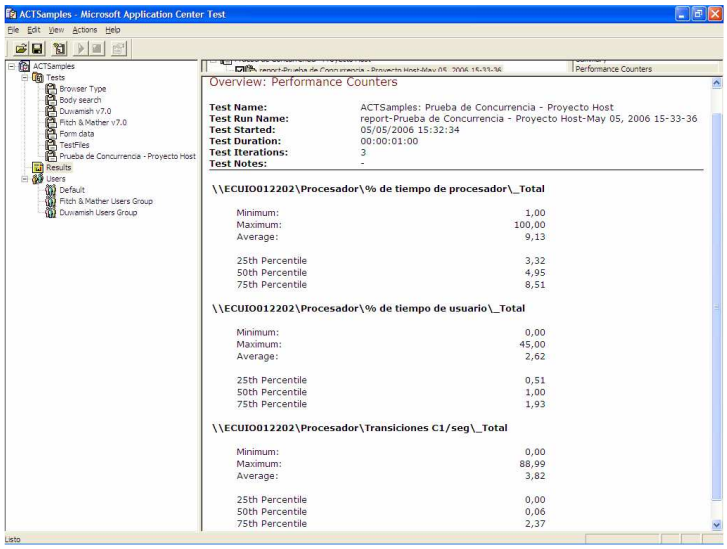
2. Resultados obtenidos:



3. Rendimiento:



4. Estadísticas:



Técnicas de Prueba	*[Esta sección presenta varias técnicas de prueba para ser aplicadas al sistema nuevo o aplicación]*
Pruebas de Caja Blanca	Las Pruebas de caja blanca son todas aquellas que se realizan en perfecto conocimiento de los procesos a seguir para verificar el resultado deseado, es decir, que se introduce la información correcta y se realizan los procedimientos adecuados para comprobar que la información resultante sea la esperada. Estas se pueden realizar mediante la navegación del sistema y el seguimiento de los pasos especificados en los procedimientos documentados, tales como diagramas de flujo, diagramas de proceso y el documento de diseños funcionales. Estas pruebas se las realizó paulatinamente de acuerdo al avance de desarrollo, y no es el objetivo de este proyecto.
Pruebas de Caja Negra	Este tipo de pruebas se basan en la utilización del sistema sin realizar un seguimiento de los procedimientos de uso, por lo que se tocan caminos no previstos o poco comunes dentro de las opciones del sistema. Por esto, este tipo de pruebas no tiene una metodología específica. Estas pruebas tiene como finalidad encontrar errores que puedan surgir por mal manejo de la herramienta o por desconocimiento de la misma, y pudiendo realizar así las validaciones pertinentes para evitar en lo posible los errores desconocidos. Estas pruebas se las realizó paulatinamente de acuerdo al avance de desarrollo, y no es el objetivo de este proyecto.

Plan de Implantación

El presente proyecto no consta de un plan de implantación, pues se basa únicamente en una propuesta.

APENDICE H – METODOLOGIA MSF – DOCUMENTO DE PLAN DE PRUEBAS

				Fecha:	08/03/2005
PLAN DE PRUEBAS					
Producto / Solución:	**COMUNICACIÓN HOST**				
Proceso en Prueba:	**CONSISTENCIA DE DATOS**				
Horario:	**15:00**	**Usuario:**	**Santiago Bustillos**	**Perfil:**	Usuario Final

Reporte del Usuario						Reporte del Proveedor		
ITEM	**FUNCIONALIDAD**	**PRE-REQUISITO**	**RESULTADOS ESPERADOS**	**RESULTADOS OBTENIDOS**	**CALIFICACION DE LA PRUEBA**	**FECHA COMPROM. DE SOLUCIÓN**	**FECHA EFECTIVA DE SOLUCIÓN**	**JUSTIFICACION AL FALLO PRESENTADO**
1	Verificar que los datos retornados sean los correctos	Crear el usuario para prueba asi como tambien tener acceso al PL/SQL para verificar los datos	Que los codigos y descripciones sean exactamente los que arroje la base de datos al consultar con la siguiente sentencia: SELECT CODIGO_REGISTRO, DEL_REGISTRO FROM T06TC002 WHERE COD_TABLA_ID = 'nombre_tabla_consultada'	Los resultados son los esperados, se comprobo la consistencia entre lo retornado por el servicio y la base de datos, y todo es correcto.	Exitosa			
2	Verificar que sucede cuando no existe un dato no encontrado	Crear el usuario para prueba asi como tambien tener acceso al PL/SQL para verificar los datos	Debe aparecer datos en blanco	Los resultados son los esperados	Exitosa			
3	Verificar rendimiento		Debe igualar o superar un tiempo de 4 segundos	Se hicieron varias pruebas: Tiempo: min:seg:miliseg : Tiempo 1 = 0:4.246 Tiempo 2 = 0:0.260 Tiempo 3 = 0:6.409 Tiempo 4 = 0:1.261 Tiempo 5 = 0:3.565 Tiempo 6 = 0:3.515 Tiempo 7 = 0:0.290 Tiempo 8 = 0:6.489 Tiempo 9 = 0:1.241 Tiempo 10 = 0:2.273 Promedio = 2:6	Exitosa			

Fecha:	05/05/2006

PLAN DE PRUEBAS

Producto / Solución:	**COMUNICACIÓN HOST**
Proceso en Prueba:	**CONCURRENCIA**

Horario:	**15:36**	**Usuario:**	**Equipo de Trabajo**	**Perfil:**	Usuario Final

Reporte del Usuario						Reporte del Proveedor		
ITEM	**FUNCIONALIDAD**	**PRE-REQUISITO**	**RESULTADOS ESPERADOS**	**RESULTADOS OBTENIDOS**	**CALIFICACION DE LA PRUEBA**	**FECHA COMPROM. DE SOLUCIÓN**	**FECHA EFECTIVA DE SOLUCIÓN**	**JUSTIFICACION AL FALLO PRESENTADO**
1	Verificar rendimiento con 10 hilos de ejecución concurrentemente durante 1 minuto	Disponer de la Herramienta Microsoft Applicaction Center Test	Debe igualar o superar un tiempo de 4 segundos	15:32:34.647 11 15:32:36.662 12 15:32:36.662 13 15:32:36.678 14 15:32:37.530 15 15:32:37.553 16 15:32:38.530 17 15:32:38.350 18 15:32:38.553 19 15:32:39.162 20 15:32:39.553 21	Exitosa			

Fecha:	08/03/2005

PLAN DE PRUEBAS

Producto / Solución:	Utilitarios del proyecto HOST
Proceso en Prueba:	Usabilidad

Horario:	**9:30**	**Usuario:**	**Bustillos Santiago**	**Perfil:**	Usuario Final

Reporte del Usuario						Reporte del Proveedor		
ITEM	**FUNCIONALIDAD**	**PRE-REQUISITOS**	**RESULTADOS ESPERADOS**	**RESULTADOS OBTENIDOS**	**CALIFICACION DE LA PRUEBA**	**FECHA COMPROM. DE SOLUCIÓN**	**FECHA EFECTIVA DE SOLUCIÓN**	**JUSTIFICACION AL FALLO PRESENTADO**
1	Verificar la usabilidad	Framework .NET	La aplicación correra independientemente de donde recida	Los resultados son los esperados	Exitosa			
2	Ingresar datos del area de comunicación	Conocer las estructuras de datos de Cobol	Aparecera los datos ingresados en una tabla ubicado debajo de la pantalla del editor de servicios .NET	Los resultados son los esperados	Exitosa			
3	Generar el codigo correspondiente a las areas de entrada y salida de comunicación		Verificar la existencia de los archivos generados en el directorio de trabajo	El código es almacenado en el directorio C:\EditorNET	Exitosa			

APENDICE I - Clase DatoRegistro.cs

```
using System;
using System.Reflection;

namespace com.pichincha.Comunicacion
{
	/// <remarks>
	/// Representa el tipo de dato DatoRegistro para Siglo21.
	/// Las clases de entrada y salida de cualquier servicio se
	/// deben derivar de esta clase
	/// </remarks>
	public abstract class DatoRegistro
	{
		public DatoRegistro()
		{
			inicial();
		}

		public abstract Object [] getDominioObjetos();

		public abstract String [] getDominioValores();

		public abstract void inicial();

		public void inicArray(DatoRegistro[] arrDatos, Type t)
		{
			try
			{
				for (int i=0; i<arrDatos.Length; i++)
				{
					try
					{
						arrDatos[i] =
(DatoRegistro)System.Activator.CreateInstance(t);
					}
					catch (Exception e)
					{
						throw new Exception("Excepción al intentar
instanciar una clase abstracta o un interfaz." + e.Message)
					}
				}
			}
			catch (Exception ex){
				throw new Exception("Excepción inesperada." + ex.Message)
			}
		}
	}
}
```

APENDICE J - Clase GestorHost.cs

```
using System;
using System.Collections;
using System.Collections.Specialized;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.IO;
using System.Data;
using com.pichincha.Util;

namespace com.pichincha.Comunicacion
{
	/// <remarks>
	/// Clase abstracta que define y administra la conexión al servidor Interserv.
	/// </summary>
	public abstract class GestorHost
	{

		#region Variables privadas
		private static string ipServidor = null;
		private static int puerto = 0;
		const int SOCKET_ERROR = -1;
		byte[] cab_out = new byte[54]; //cabecera de entrada => devuelta por el
servicio	DatoRegistro entrada;
		private const int DISPLAY    = 0;
		private const int COMP       = 1;
		private const int COMP3      = 2;
		private byte [] message = new byte[10000];
		private ArrayList longitudCampo    = null;
		private ArrayList longitudCampoIn  = new ArrayList();
		private ArrayList longitudCampoOut = new ArrayList();
		private int longitudTotal;
		private int contRegistro = 0;
		private int longitud_in = 0;
		private int longitud_out = 0;
		private int retorno = -1;
		private short severidad = -1;
		private const int tamRec = 32;
		private byte[] datos;
		private DatoRegistro entrada;
		private DatoRegistro salida;
		#endregion

		#region Constructores
		public GestorHost()
		{
		}
		#endregion

		#region Métodos públicos
		/// <summary>
		/// Método que invoca el servicio en el servidor InterServ
		/// </summary>
		/// <param name="CRCServicio">Número de identificación CRC para cada
servicio</param>
		/// <returns>int Define si el servicio fue o no invocado con
éxito</returns>
		/// Los puertos a partir del 12500 hasta 16499 seran habilitados para
Siglo21.
		/// Los puertos a partir del 16500 serán para los canales
		public string invocar(string CRCServicio, StringDictionary entorno)
		{
			/*string numOperacion = "0";
			string numLinea = "0";*/
			string resto = " ".PadRight(20,' ');

			string cabe = (string)entorno["version"] +
(string)entorno["usuario"] + (string)entorno["dominio"] + (string)entorno["nodo"];

			// En este punto se puede dar lógica básica de balanceo

			if (ipServidor == null)
```

```
                        {
                                DataTable TBConexion =
Recursos.recuperaRecursos(Recurso.CONEXION);
                                DataRow rowConexion  = TBConexion.Rows[0];
                                ipServidor = rowConexion["SERVIDOR"].ToString();
                                puerto     = int.Parse(rowConexion["PUERTO"].ToString());
                        }

                        IPAddress myIP = IPAddress.Parse(ipServidor);
                        IPEndPoint hostRemoto = new IPEndPoint(myIP, puerto);

                        Socket s = new Socket(hostRemoto.Address.AddressFamily,
                                SocketType.Stream,
                                ProtocolType.Tcp);
                        try
                        {
                                s.Connect(hostRemoto);
                        }
                        catch (Exception e)
                        {
                                Console.WriteLine("ERROR - Conectando al host remoto : " +
e.ToString());
                        }

                        entrada = getEntradaInterno();
                        salida  = getSalidaInterno();
                        getLongitudCobol();     //Obtengo longitudes en entrada y salida
                        string nombreServicio = GetType().Name.ToString().PadRight(8,' ');
                        byte[] datos = traducirDotNetCobol();
                        string str_long_in  = longitud_in.ToString().PadLeft(6,'0');
                        string str_long_out = longitud_out.ToString().PadLeft(6,'0');


                        cabe  += nombreServicio + str_long_in + str_long_out +
(string)entorno["codEmpresa"] +
                                        (string)entorno["codCentro"] +
(string)entorno["codEntorno"] + (string)entorno["codIdioma"] +
                                                CRCServicio + "N    ND"; //+
numOperacion.PadLeft(9,'0') + numLinea.PadLeft(2,'0') + (string)entorno["token"] +
"SG21" + resto;

                        String datosDevueltos = "";

                        byte[] cab_servicio = Encoding.ASCII.GetBytes(cabe);

                        // Envío los cabecera del servicio
                        int i = s.Send(cab_servicio);

                        // Envío los datos del servicio
                        s.Send(datos);
                        // Recibo la cabecera de vuelta
                        s.Receive(cab_out);
                        datosDevueltos = Encoding.ASCII.GetString(cab_out);
                        severidad = short.Parse(datosDevueltos.Substring(1,1));
                        retorno = int.Parse(datosDevueltos.Substring(2,6));

                        if (retorno == 0)
                        {
                                // Recibo los datos de vuelta
                                byte[] bytes = new byte[longitud_out];
                                if (longitud_out > 0)
                                {
                                        int nBytesLeidos=0;
                                        int indice=0;
                                        int leer = 2000;
                                        do
                                        {
                                                byte[] bytesEnv = new byte[leer];
                                                nBytesLeidos = s.Receive(bytesEnv);
                                                if (nBytesLeidos > 0)
                                                {
                                                        concatenaBytes(indice,
nBytesLeidos,ref bytes, bytesEnv);
                                                        indice += nBytesLeidos;
                                                }
                                        } while ( nBytesLeidos > 0 && indice <
longitud_out);
```

```
                                traducirCobolDotNet(bytes);
                        }
                }
                s.Shutdown(SocketShutdown.Both);
                s.Close();
                return datosDevueltos;
        }

        public int getRetorno()
        {
                return retorno;
        }
        public short getSeveridad()
        {
                return severidad;
        }
        #endregion

        #region Métodos protegidos abstractos
        protected abstract DatoRegistro getEntradaInterno();

        protected abstract DatoRegistro getSalidaInterno();
        #endregion

        #region Métodos privados

        private string getCabMsgApache(int longData, string recurso) {
                string cabApache = "POST " + recurso + " HTTP/1.0\nContent-
type:application/x-www-form-urlencoded\nContent-length: " +longData + "\n";
                return cabApache;
        }

        private void concatenaBytes(int longIni, int tamanio, ref byte[] final,
byte[] aCopiar)
        {
                int numByteCopy = 0;
                for(int i = longIni; i < longIni+tamanio; i++)
                {
                        final[i] = aCopiar[numByteCopy++];
                }
        }

        private void getLongitudCobol()
        {
                int aux = 0;
                if (entrada == null)
                        longitud_in = 0;
                else
                {
                        longitudTotal = 0;
                        longitudCampo = new ArrayList();
                        aux = LongitudCobol(entrada, 0);
                        longitudCampoIn = longitudCampo;
                        longitud_in = longitudTotal;
                }

                if (salida == null)
                        longitud_out = 0;
                else
                {
                        longitudTotal = 0;
                        longitudCampo = new ArrayList();
                        aux = LongitudCobol(salida, 0);
                        longitudCampoOut = longitudCampo;
                        longitud_out = longitudTotal;
                }
        }


        private byte [] traducirDotNetCobol()
        {
                if (entrada == null)
                        return new byte[0];

                int numeroDatos = entrada.getDominioValores().Length;
                datos = new byte[longitud_in];
                int indice = 0;
```

```
                        for (int i = 0; i < numeroDatos; i++)
                        {
                                Object obj = entrada.getDominioObjetos()[i];
                                int tipoDato =
int.Parse(entrada.getDominioValores()[i].Substring(5,1));
                                if (obj is DatoRegistro || obj.GetType().IsArray)
                                {
                                        if (obj.GetType().IsArray)
                                        {
                                                Object [] objArray = (Object [])obj;
                                                //DEPENDING ON
                                                int lonArray = 0;
                                                int depon =
salida.getDominioValores()[i].IndexOf("->");
                                                if (depon != -1)
                                                {
                                                        int ValDepOn =
int.Parse(salida.getDominioValores()[i].Substring(depon+2));
                                                        lonArray =
((S21Integer)salida.getDominioObjetos()[ValDepOn]).getValorContenido();
                                                }
                                                else
                                                {
                                                        lonArray =
int.Parse(entrada.getDominioValores()[i].Substring(7,5));
                                                }
                                                for (int itot = 0; itot < lonArray; itot++ )
                                                {

                                                        indice =
ObjetoDatoRegistro((DatoRegistro) objArray[itot], i, indice);
                                                        indice++;
                                                }
                                                indice--;
                                        }
                                        else
                                                indice = ObjetoDatoRegistro((DatoRegistro)
obj, i, indice);  //+i
                                }
                                else
                                {
                                        byte[] datosEntradaRec = new
byte[(int)longitudCampoIn[indice]];
                                        datosEntradaRec = DotNetCobolObjeto(obj, tipoDato,

indice);
                                        for (int j = 0; j < (int)longitudCampoIn[indice];

j++)
                                                datos[contRegistro++] = datosEntradaRec[j];
                                }
                                indice++;
                        }
                        return datos;
                }

                private byte[] traducirDotNetCobolCOMP(int valor, int lon_bin)
                {

                        byte[] salidaJC= new byte[lon_bin];

                        for (int i=lon_bin-1;i>=0;i--)
                        {
                                salidaJC[i]=(byte)(valor & 0x000000ff);
                                valor>>=8;
                        }
                        return salidaJC;
                }


                private byte[] traducirDotNetCobolCOMP3(String cadena, int lon_bin)
                {

                        byte[] salida= new byte[lon_bin];
                        double valorPrimitivo = double.Parse(cadena);

                        Boolean zona=true;
                        int j=salida.Length-1;
                        if (valorPrimitivo < 0)
```

```
                {
                        salida[j]=13;
                }
                else
                {
                        salida[j]=12;
                }
                for (int i=cadena.Length-1;i>=0;i--)
                {
                        char [] car = cadena.ToCharArray(i,1);
                        byte carByte=(byte)char.GetNumericValue(car[0]);
                        if (carByte>=0 && carByte<=9)
                        {
                                if (zona)
                                {
                                        carByte<<=4;
                                        salida[j]=(byte)(salida[j] | carByte);
                                        zona=false;
                                        j--;
                                }
                                else
                                {
                                        zona=true;
                                        salida[j]=carByte;
                                }
                        }
                }
                return salida;
        }


        private byte[] traducirDotNetCobolDISPLAY(String cadena, int lon, Object
obj)
        {

                int offset = lon - cadena.Length;
                long valorPrimitivo = long.Parse(cadena);
                if (offset > 0)
                {       //      Completo con 0 por la izda.
                        char[] espacios = new char[offset];
                        for (int i = 0; i < offset; i++)
                        {
                                espacios[i] = '0';
                        }
                        cadena = new String(espacios)+cadena;
                }

                if (offset < 0)
                {
                        if( valorPrimitivo < 0 )
                                cadena = cadena.Substring(1,lon+1);
                        else
                                cadena = cadena.Substring(0,lon);
                }

                byte[] salida =       TraductorEBCDICASCII.toEBCDIC(cadena);

                byte carByte;
                if (valorPrimitivo < 0 )
                {
                        carByte = Convert.ToByte(-33);
                }
                else
                {
                        carByte = Convert.ToByte(49);
                }
                // nelson - comento para tramas HB61
                //salida[salida.Length-1]=(byte)(salida[salida.Length-1] &
carByte);

                return salida;
        }


        private byte[] traducirTipoDatoFinal(Object obj, int longitud, int
tipoCobol)
        {
```

```
                        byte[] datosEntrada = new byte[longitud];
                        if (obj is S21Integer) {
                                int datvalor = ((S21Integer) obj).getValorContenido();
                           datosEntrada = traducirDotNetCobolDISPLAY(datvalor.ToString(),
longitud, obj);
                    }
                        return datos;
                }


                private void traducirCobolDotNet(byte[] nuevoValor)
                {
                        if (salida == null)
                                return;

                        int numeroDatos = salida.getDominioValores().Length;

                        int lonArray = 0;

                        datos = new byte[longitud_out];
                        int indice = 0;
                        for (int i = 0; i < numeroDatos; i++)
                        {
                                Object obj = salida.getDominioObjetos()[i];
                                int tipoDato =
int.Parse(salida.getDominioValores()[i].Substring(5,1));
                                if (obj is DatoRegistro || obj.GetType().IsArray)
                                {
                                        if (obj.GetType().IsArray)
                                        {
                                                Object [] objArray = (Object [])obj;

                                                //DEPENDING ON
                                                int depon =
salida.getDominioValores()[i].IndexOf("->");
                                                if (depon != -1)
                                                {
                                                        int ValDepOn =
int.Parse(salida.getDominioValores()[i].Substring(depon+2));
                                                        lonArray =
((S21Integer)salida.getDominioObjetos()[ValDepOn]).getValorContenido();
                                                }
                                                else
                                                {
                                                        lonArray =
int.Parse(salida.getDominioValores()[i].Substring(7,5));
                                                }

                                                for (int itot = 0; itot < lonArray; itot++ )
                                                {

                                                        indice =
ObjetoDatoRegistroCobol(nuevoValor, (DatoRegistro) objArray[itot], i, indice); //+i

                                                        indice++;
                                                }
                                                indice--;
                                        }
                                        else
                                                indice = ObjetoDatoRegistroCobol(nuevoValor,
(DatoRegistro) obj, i, indice);
                                }
                                else
                                {
                                        CobolDotNetObjeto(nuevoValor, obj, tipoDato,
indice);
                                }
                                indice++;
                        }
                }


                private void traducirCobolDotNetCOMP(byte[] nuevoValor, int posObj,
Object obj)
                {
                        int valor=0;
                        for (int i=0;i<nuevoValor.Length;i++)
```

```
                {
                        valor<<=8;
                        valor=valor | ((int)nuevoValor[i] & 0x000000ff);
                }
                if (nuevoValor.Length==2)
                {
                        valor<<=16;
                        valor>>=16;
                }
                ((S21Integer)obj).setValorContenido(valor);
        }


        private void traducirCobolDotNetCOMP3(byte[] nuevoValor, int posObj,
Object obj)
        {
                int carInt;
                int valorContenido=0;
                Boolean negativo=false;
                int factor=1;
                for (int i=nuevoValor.Length-1;i>=0;i--)
                {
                        carInt=((int)nuevoValor[i] & 0x0000000f);
                        if (carInt==13)
                        {
                                negativo=true;
                        }
                        else
                        {
                                if (carInt != 12)
                                {
                                        valorContenido=valorContenido+carInt*factor;
                                        factor=factor*10;
                                }
                        }
                        carInt=((int)nuevoValor[i] & 0x000000f0);
                        carInt>>=4;
                        valorContenido=valorContenido+carInt*factor;
                        factor=factor*10;
                }
                if (negativo)
                        valorContenido=(-valorContenido);

                if (obj is S21Integer)
                        ((S21Integer)obj).setValorContenido(valorContenido);
                else
                        if (obj is S21Long)
                                ((S21Long)obj).setValorContenido(valorContenido);


        }


        private void traducirCobolDotNetDISPLAY(byte[] nuevoValor, int posObj,
Object obj)
        {
                StringBuilder buf=new StringBuilder();
                int signo = 1;
                int carInt;
                int valorContenido;
                carInt=((int)nuevoValor[nuevoValor.Length-1] & 0x000000f0);
                if (carInt==208)
                        signo = -1;
                carInt=((int)nuevoValor[nuevoValor.Length-1] & 0x0000000f);

                String strDato = TraductorEBCDICASCII.toASCII(nuevoValor);
                buf.Append(strDato.Substring(0,strDato.Length-1));
                buf.Append(forDigit(carInt,10));
                string aux = buf.ToString();
                if (aux.Length == 0)
                        aux = "0";
                try
                {
                        valorContenido=int.Parse(aux)*signo;
                }
                catch (Exception e)
                {
```

```
                        valorContenido=0*signo;
                        Console.WriteLine("ERROR - Traducir CobolDotNetDisplay : "
+ e.ToString());
                }

                if (obj is S21Integer)
                        ((S21Integer)obj).setValorContenido(valorContenido);
                else
                        ((S21Long)obj).setValorContenido(valorContenido);
        }


        private void traducirCobolDotNetDISPLAYBig(byte[] nuevoValor, int posObj,
Object obj)
        {
                StringBuilder buf=new StringBuilder();
                int carInt;
                carInt=((int)nuevoValor[nuevoValor.Length-1] & 0x000000f0);
                if (carInt==208)
                        buf.Append('-');
                carInt=((int)nuevoValor[nuevoValor.Length-1] & 0x0000000f);
                //String salidaStr=TraductorEBCDICASCII.toASCII(nuevoValor);
                String salidaStr=Encoding.ASCII.GetString(nuevoValor);
                if(salidaStr.IndexOf("-") != -1)
                        salidaStr=salidaStr.Substring(0,salidaStr.IndexOf("-
"))+salidaStr.Substring(salidaStr.IndexOf("-")+1);
                buf.Append(salidaStr.Substring(0,salidaStr.Length-1));
                buf.Append(forDigit(carInt,10));
                String valorAux = buf.ToString();
                if (valorAux.Trim().CompareTo("0") == 0)
                        valorAux = "0.0";
                valorAux=valorAux.Trim();
                double valorDouble=double.Parse(valorAux);
                if (valorDouble >= 0)
                        ((S21BigDecimal)obj).setValorContenido((double)valorDouble
/ ((S21BigDecimal)obj).getDivisor());
                else
                        if (valorDouble < 0)
                                ((S21BigDecimal)obj).setValorContenido(((double)
valorDouble / ((S21BigDecimal)obj).getDivisor()) * -1);
                return;
        }


        private void traducirCobolDotNetCOMPBig(byte[] nuevoValor, int posObj,
Object obj)
        {
                int  valor=0;
                for (int i=0;i<nuevoValor.Length;i++)
                {
                        valor<<=8;
                        valor=valor | ((int)nuevoValor[i] & 0x000000ff);
                }
                if (nuevoValor.Length==2)
                {
                        valor<<=16;
                        valor>>=16;
                }
                if(valor == 0)
                        ((S21BigDecimal)obj).setValorContenido((double) valor);
                else
                {
                        String valorAux = valor.ToString().Trim();
                        double valorDouble=double.Parse(valorAux);
                        double valorD = 0.0;

                        if (valorDouble > 0)

        valorD=(double)(valorDouble/((S21BigDecimal)obj).getDivisor());
                        else
                                if (valorDouble < 0)

        valorD=(double)((valorDouble/((S21BigDecimal)obj).getDivisor())*-1);
                        ((S21BigDecimal)obj).setValorContenido((double) valorD);
                }
                return;
        }
```

```
		private void traducirCobolDotNetCOMP3Big(byte[] nuevoValor, int posObj,
Object obj)
		{
			int carInt;
			long valorContenido=0;
			bool negativo=false;
			long factor=1;
			for (int i=nuevoValor.Length-1;i>=0;i--)
			{
				carInt=((int)nuevoValor[i] & 0x0000000f);
				if (carInt==13)
				{
					negativo=true;
				}
				else
				{
					if (carInt != 12)
					{
						valorContenido=valorContenido+carInt*factor;
						factor=factor*10;
					}
				}
				carInt=((int)nuevoValor[i] & 0x000000f0);
				carInt>>=4;
				valorContenido=valorContenido+carInt*factor;
				factor=factor*10;
			}
			if (negativo)
				valorContenido=(-valorContenido);

	((S21BigDecimal)obj).setValorContenido((double)valorContenido/((S21BigDecimal)obj
).getDivisor());
			return;
		}


		private void traducirCobolDotNetBoolean(byte[] nuevoValor, int posObj,
Object obj)
		{
			bool valor;
			if ((int)nuevoValor[0] == 240)
				valor=true;
			else
				valor=false;
			((S21Boolean)obj).setValorContenido((bool)valor);
		}


		private char forDigit(int digit, int radix)
		{
			if ((digit >= radix) || (digit < 0))
			{
				return '\0';
			}
			if ((radix < 2) || (radix > 36))
			{
				return '\0';
			}
			if (digit < 10)
			{
				return (char)('0' + digit);
			}
			return (char)('a' - 10 + digit);
		}


		private int LongitudCobol(DatoRegistro obj, int registro)
		{
			int numeroDatos = obj.getDominioValores().Length;
			int contador = registro;
			int contArray = 0;
			int lonArray = 0;
			for (int i = 0;		i < numeroDatos; i++)
			{
```

```
                        int longitud =
int.Parse(obj.getDominioValores()[i].Substring(0,5));
                        int tipoDato =
int.Parse(obj.getDominioValores()[i].Substring(5,1));

                        if (longitud == 0)
                        {
                                Object obj1 = obj.getDominioObjetos()[i];
                                if (obj1 is DatoRegistro || obj1.GetType().IsArray)
                                {
                                        if (obj1.GetType().IsArray)
                                        {
                                                contArray = contador;
                                                lonArray =
int.Parse(obj.getDominioValores()[i].Substring(7,5));
                                                Object [] objArray = (Object
[])obj.getDominioObjetos()[i];
                                                contador =
LongitudCobol((DatoRegistro) objArray[0], contador);
                                                int limite = contador;
                                                for (int itot = 0; itot < lonArray -
1; itot++ )
                                                {
                                                        for (int iar = contArray; iar
<= limite; iar++ )
                                                        {

        longitudCampo.Add((int)longitudCampo[iar]);
                                                                longitudTotal +=
(int)longitudCampo[contador++];
                                                        }
                                                }
                                        }
                                        else
                                                contador =
LongitudCobol((DatoRegistro) obj1, contador);
                                }
                        }
                        else
                        {
                                switch (tipoDato)
                                {
                                        case DISPLAY   :
        longitudCampo.Add(longitud);                    break;
                                        case COMP              :
        longitudCampo.Add(longitud / 2 + 1); break;
                                        case COMP3             :
        longitudCampo.Add(longitud / 2 + 1);            break;
                                }
                                longitudTotal += (int)longitudCampo[contador];
                        }
                        contador++;
                }
                return contador-1;
        }


        private int ObjetoDatoRegistro(DatoRegistro obj, int registro, int
numreg)
        {
                int numeroDatos = obj.getDominioValores().Length;
                int contador = numreg;
                int lonArray =0;
                for (int i = 0; i < numeroDatos; i++)
                {
                        Object obj1 = obj.getDominioObjetos()[i];
                        int tipoDato =
int.Parse(obj.getDominioValores()[i].Substring(5,1));
                        if (obj1 is DatoRegistro || obj1.GetType().IsArray)
                        {
                                if (obj1.GetType().IsArray)
                                {
                                        Object [] objArray = (Object[])obj1;
                                        DatoRegistro obj2 =
(DatoRegistro)objArray[0];
                                        lonArray =
        int.Parse(obj.getDominioValores()[i].Substring(7,5));
```

```
                                        for (int itot = 0; itot < lonArray; itot++ )
                                        {
                                                contador =
ObjetoDatoRegistro((DatoRegistro) objArray[itot], i, contador);
                                                contador++;
                                        }
                                        contador--;
                                }
                                else
                                        contador = ObjetoDatoRegistro((DatoRegistro)
obj1, i, contador);
                        }
                        else
                        {
                                byte[] datosEntradaRec = new
byte[(int)longitudCampoIn[contador]];
                                datosEntradaRec = DotNetCobolObjeto(obj1, tipoDato,
contador);
                                for (int j = 0; j < (int)longitudCampoIn[contador];
j++)
                                        datos[contRegistro++] = datosEntradaRec[j];
                        }

                        contador++;
                }
                return contador - 1;
        }


        private byte[] DotNetCobolObjeto(Object obj, int tipoDato, int registro)
        {
                byte[] datosEntrada = new byte[(int)longitudCampoIn[registro]];
                switch (tipoDato)
                {
                        case DISPLAY   :
                                if (obj is int)
                                {
                                        int datvalor = ((S21Integer)
obj).getValorContenido();
                                        datosEntrada =
traducirDotNetCobolDISPLAY(datvalor.ToString(), (int)longitudCampoIn[registro], obj);
                                }
                                if (obj is S21Long)
                                {
                                        if (obj is long)
                                        {
                                                long datvalor = ((S21Long)
obj).getValorContenido();
                                                datosEntrada =
traducirDotNetCobolDISPLAY(datvalor.ToString(), (int)longitudCampoIn[registro], obj);
                                        }
                                }

                                if (obj is S21String)
                                {
                                        S21String datvalor = new
S21String(((S21String) obj).getValorContenido());
                                        datosEntrada =
       TraductorEBCDICASCII.toEBCDIC(datvalor.toStringRellenaDato((int)longitudCampoIn[r
egistro]));
                                }

                                if (obj is S21BigDecimal)
                                {
                                        string datvalor = ((S21BigDecimal)
obj).getValorContenidoString();
                                        datosEntrada =
traducirDotNetCobolDISPLAY(datvalor, (int)longitudCampoIn[registro], obj);
                                }

                                break;
                        case COMP      :
                                if (obj is S21Integer)
                                {
                                        int datvalor = ((S21Integer)
obj).getValorContenido();
```

```
                                        datosEntrada =
traducirDotNetCobolCOMP(datvalor, (int)longitudCampoIn[registro]);
                                    }
                                    if (obj is S21Long)
                                    {
                                        int datvalor = (int)((S21Long)
obj).getValorContenido();
                                        datosEntrada =
traducirDotNetCobolCOMP(datvalor, (int)longitudCampoIn[registro]);
                                    }
                                    break;
                                case COMP3     :
                                    if (obj is S21Integer)
                                    {
                                        int datvalor = ((S21Integer)
obj).getValorContenido();
                                        datosEntrada =
traducirDotNetCobolCOMP3(datvalor.ToString(), (int)longitudCampoIn[registro]);
                                    }
                                    if (obj is S21Long)
                                    {
                                        long datvalor = ((S21Long)
obj).getValorContenido();
                                        datosEntrada =
traducirDotNetCobolCOMP3(datvalor.ToString(), (int)longitudCampoIn[registro]);
                                    }

                                    if (obj is S21BigDecimal)
                                    {
                                        string datvalor = ((S21BigDecimal)
obj).getValorContenidoString();
                                        datosEntrada =
traducirDotNetCobolCOMP3(datvalor, (int)longitudCampoIn[registro]);
                                    }

                                    break;
                        }
                        return datosEntrada;
                }


                private int ObjetoDatoRegistroCobol(byte [] nuevoValor, DatoRegistro obj,
int registro, int numreg)
                {
                        int numeroDatos = obj.getDominioValores().Length;
                        int contador = numreg;
                        int lonArray =0;
                        for (int i = 0; i < numeroDatos; i++)
                        {
                                Object obj1 = obj.getDominioObjetos()[i];
                                int tipoDato =
int.Parse(obj.getDominioValores()[i].Substring(5,1));

                                //****
                                if (obj1 is DatoRegistro || obj1.GetType().IsArray)
                                {
                                    if (obj1.GetType().IsArray)
                                    {
                                        Object [] objArray = (Object[])obj1;
                                        DatoRegistro obj2 =
(DatoRegistro)objArray[0];
                                        lonArray =
        int.Parse(obj.getDominioValores()[i].Substring(7,5));

                                        ArrayList vector = new ArrayList();
                                        for (int itot = 0; itot < lonArray; itot++ )
                                        {
                                            contador =
ObjetoDatoRegistroCobol(nuevoValor, (DatoRegistro) objArray[itot], i, contador);
                                            vector.Add((DatoRegistro)
objArray[itot]);

                                            contador++;
                                        }
                                        contador--;
                                    }
                                    else
```

```
                                        contador =
ObjetoDatoRegistroCobol(nuevoValor, (DatoRegistro) obj1, i, contador);
                        }
                        else
                        {
                                CobolDotNetObjeto(nuevoValor, obj1, tipoDato,
contador);
                        }
                        contador++;
                }
                return contador - 1;
        }


        private void CobolDotNetObjeto(byte[] nuevoValor, Object obj, int
tipoDato, int registro)
        {
                byte[] dato = new byte[(int)longitudCampoOut[registro]];
                int posicion=0;
                if (registro > 0)
                        for(int pr = registro-1; pr >= 0; pr--)
                                posicion += (int)longitudCampoOut[pr];

                try
                {
                        for (int i=0; i<dato.Length; i++)
                        {
                                dato[i]=nuevoValor[posicion + i];
                        }
                }
                catch(Exception e)
                {
                        Console.WriteLine("ERROR - Indice desbordado" +
e.Message);
                        return;
                }

                switch (tipoDato)
                {
                        case DISPLAY   :
                                if (obj is S21Integer || obj is S21Long)
                                {
                                        traducirCobolDotNetDISPLAY(dato, registro,
obj);
                                }
                                if (obj is S21BigDecimal)
                                {
                                        traducirCobolDotNetDISPLAYBig(dato,
registro, obj);
                                }
                                if (obj is S21String)
                                {

        ((S21String)obj).setValorContenido(TraductorEBCDICASCII.toASCII(dato));
                                }
                                if (obj is S21Boolean)
                                {
                                        traducirCobolDotNetBoolean(dato, registro,
obj);
                                }
                                break;
                        case COMP      :
                                if (obj is S21Integer || obj is S21Long)
                                {
                                        traducirCobolDotNetCOMP(dato, registro,
obj);
                                }
                                if (obj is S21BigDecimal)
                                {
                                        traducirCobolDotNetCOMPBig(dato, registro,
obj);
                                }

                                break;
                        case COMP3     :
                                if (obj is S21Integer || obj is S21Long)
                                {
```

```
                                        traducirCobolDotNetCOMP3(dato, registro,
obj);
                                    }
                                    if (obj is S21BigDecimal)
                                    {
                                        traducirCobolDotNetCOMP3Big(dato, registro,
obj);
                                    }

                                    break;
                        }
                }

                #endregion

        }
}
```

APENDICE K - Clase S21String.cs

```
using System;

namespace com.pichincha.Comunicacion
{
	/// <remarks>
	/// Representa el tipo de dato String para Siglo21.
	/// </remarks>
	public class S21String
	{
		private String valorContenido="";

		public S21String()
		{
		}

		public S21String(String valor)
		{
			this.valorContenido = valor;
		}

		public String getValorContenido()
		{
			return valorContenido;
		}


		public void setValorContenido(String newValue)
		{
			valorContenido = newValue;
		}

		public String toString()
		{
			return base.ToString();
		}

		public String toStringRellenaDato(int longTotal) {
			String texto = " ";
			int len = this.valorContenido.Length;
			for(int i = 1; i < longTotal-len; i++)
				texto += " ";
			if (longTotal-len == 0)
				return valorContenido;
			return valorContenido+texto;
		}

	}
}
```

APENDICE L - Clase S21Integer.cs

```
using System;

namespace com.pichincha.Comunicacion
{
	/// <remarks>
	/// Representa el tipo de dato Integer para Siglo21.
	/// </remarks>
	public class S21Integer
	{
		private int valorContenido=0;

		public S21Integer()
		{
		}

		public S21Integer(int valor)
		{
			this.valorContenido = valor;
		}

		public int getValorContenido()
		{
			return valorContenido;
		}

		public void setValorContenido(int newValue)
		{
			valorContenido = newValue;
		}

		public String toString()
		{
			return base.ToString();
		}

	}
}
```

APENDICE M - Clase S21Long.cs

```
using System;

namespace com.pichincha.Comunicacion
{
	/// <remarks>
	/// Representa el tipo de dato Long para Siglo21.
	/// </remarks>
	public class S21Long
	{
		private long valorContenido=0;

		public S21Long()
		{
		}

		public S21Long(long valor)
		{
			this.valorContenido = valor;
		}

		public long getValorContenido()
		{
			return valorContenido;
		}

		public void setValorContenido(int newValue)
		{
			valorContenido = newValue;
		}

		public String toString()
		{
			return base.ToString();
		}

	}
}
```

APENDICE N - Clase S21BigDecimal.cs

```
using System;

namespace com.pichincha.Comunicacion
{
	/// <remarks>
	/// Representa el tipo de dato BigDecimal para Siglo21.
	/// </remarks>
	public class S21BigDecimal
	{
		private double valorContenido=0.00;
		private int decimales;
		private long valor;
		private long divisor;
		private const int DECIMALES_REDONDEO=4;
		private const int FACTOR_REDONDEO=5001;

		public S21BigDecimal()
		{
			this.decimales = 2;
			divisor = (long)Math.Pow((double)10,(double)decimales);
		}

		public S21BigDecimal(double valorDouble)
		{
			this.valorContenido = valorDouble;
			//this.decimales = 2;
			this.valor = calculoValor(valorDouble, decimales);
			divisor = (long)Math.Pow((double)10,(double)decimales);
		}

		public S21BigDecimal(double valorDouble, int dec)
		{
			this.valorContenido = valorDouble;
			this.decimales = dec;
			this.valor = calculoValor(valorDouble, dec);
			divisor = (long)Math.Pow((double)10,(double)decimales);
		}


		public S21BigDecimal(int dec)
		{
			this.decimales = dec;
			divisor = (long)Math.Pow((double)10,(double)decimales);
		}

		private static long calculoValor(double valorDouble, int dec)
		{
			//Calculo parte entera
			long parteEntera = (long) (valorDouble);
			//Calculo parte decimal
			double valDecimal = valorDouble % 1;
			for (int i = 0; i < dec+DECIMALES_REDONDEO; i++)
				valDecimal = valDecimal * 10;
			//	long parteDecimal = (long) (Math.rint(valDecimal));
			long parteDecimal = (long) valDecimal;

			return calculoValor(parteEntera, parteDecimal, dec);
		}

		private static long calculoValor(long parteEntera, long valDecimal, int
dec)
		{
			bool negativo = false;
			//Calculo negativo
			if (parteEntera < 0 || valDecimal < 0)
				negativo = true;
			parteEntera = Math.Abs(parteEntera);
			valDecimal = Math.Abs(valDecimal);

			//Calculo parte entera
			if (parteEntera != 0)
			{
```

```
                        for (int i = 0; i < dec; i++)
                                parteEntera = parteEntera * 10;
                }
                //Calculo parte decimal
                long parteDecimal = 0;
                if (valDecimal != 0)
                {
                        double vDecimal = valDecimal + FACTOR_REDONDEO;
                        for (int i = 0; i < DECIMALES_REDONDEO; i++)
                                vDecimal = vDecimal / 10;
                        parteDecimal = (long)vDecimal;
                }
                if (negativo)
                        return (parteEntera + parteDecimal) * (-1);
                else
                {
                        return parteEntera + parteDecimal;
                }
        }

        public int getDecimales()
        {
                return decimales;
        }

        public long getDivisor()
        {
                return divisor;
        }

        public double getValorContenido()
        {
                return valorContenido;
        }

        public string getValorContenidoString()
        {
                //Calculo parte entera
                long parteEntera = (long) (valorContenido);
                //Calculo parte decimal
                double valDecimal = valorContenido % 1;
                for (int i = 0; i < getDecimales()+DECIMALES_REDONDEO; i++)
                        valDecimal = valDecimal * 10;
                long parteDecimal = (long) valDecimal;
                string decstr = parteDecimal.ToString();
                for (int i = 0; i < this.decimales; i++)
                {
                        decstr += "0";
                }

                return parteEntera.ToString() + decstr;
        }

        public long getValorContenidoLong()
        {
                return valor;
        }


        public void setValorContenido(double newValue)
        {
                valorContenido = newValue;
        }


        public String toString()
        {
                return base.ToString();
        }

        }
}
```

APENDICE O - Clase S21Boolean.cs

```
using System;

namespace com.pichincha.Comunicacion
{

	/// <remarks>
	/// Representa el tipo de dato Boolean para Siglo21.
	/// </remarks>
	public class S21Boolean
	{
		private bool valorContenido=false;
		private string valorBoolStr="0";

		public S21Boolean()
		{
		}

		public S21Boolean(bool valor)
		{
			this.valorContenido = valor;
			if (valor)
				valorBoolStr = "1";
			else
				valorBoolStr = "0";
		}

		public bool getValorContenido()
		{
			return valorContenido;
		}

		public string getValorContenidoStr()
		{
			return valorBoolStr;
		}

		public void setValorContenido(bool newValue)
		{
			valorContenido = newValue;
		}

		public String toString()
		{
			return base.ToString();
		}

	}
}
```

BIBIOGRAFIA

DIRECCIONES ELECTRÓNICAS

- Juval Lowy
  http://www.idesign.net/idesign/download/IDesign%20CSharp%20Coding%20Standard.zip

- El lenguaje de programación C#.
  José Antonio Gonzáles Seco
  http://www.josanguapo.com

- Introduction to SNA
  www.yale.edu/pclt/COMM/SNA.HTM

- Microsoft Host Integration Server: SNA Server
  http://www.microsoft.com/hiserver/evaluation/overview/TopTen.asp

- SNA Server.
  http://www.windowstimag.com/atrasados/1997/07_mar97/SNAServer.htm

- Protocolo TCP/IP
  http://www.monografias.com/trabajos15/tcp-protocolo/tcp-protocolo.shtml

- Protocolos de Transporte
  http://eia.udg.es/~atm/tcp-ip/tema_4_6_2.htm

- Fundamentos de Cobol – Cobol básico
  http://programarenc.webcindario.com/PaginasdeCOBOL/cobol.htm

- Uso de los Enlaces Distribuidos
  http://support.microsoft.com/default.aspx?scid=http://www.microsoft.com/intlkb/spain/E10/2/84.asp#appliesto

- TXSeries
  http://publib.boulder.ibm.com/infocenter/txen/index.jsp?topic=/com.ibm.txseries510.doc/erzhae0085.htm

- Using COMTI to Connect NT and IBM Mainframes
  http://www.winnetmag.com/Windows/Article/ArticleID/4903/4903.html

LIBROS

- TANEMBAUM Andrew. “Redes de Computadoras”. Editorial Prentice may Hispanoamericana S.A. 1998.