



T  
005.86  
ZAM  
R-2



# **ESCUELA SUPERIOR POLITECNICA DEL LITORAL**

**FACULTAD DE INGENIERÍA EN ELECTRICIDAD Y COMPUTACIÓN**

## **PROYECTO DE GRADUACIÓN**

### **"Tecnología Cliente Servidor" Simulación del Proceso de Registros en una Universidad**

**Previo a la obtención del Título de:**

**INGENIERO EN COMPUTACION**

**Presentado por:**

**Nubia Zambrano Consuegra  
Carlos Carvajal Lema  
Ricardo Alvarez Tagle**

**GUAYAQUIL - ECUADOR  
1999**

## **AGRADECIMIENTO**

Al ING. CARLOS VALERO  
Director de Topico, por su  
ayuda y colaboración para la  
realización de este trabajo.

## **DEDICATORIA**

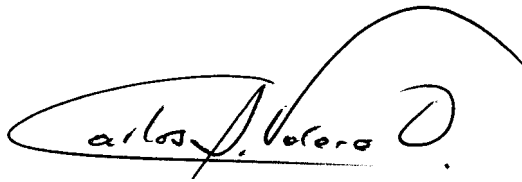
A NUESTROS PADRES  
Y HERMANOS

## TRIBUNAL DE GRADO



---

**ING. CARLOS MONSALVE**  
Presidente del Tribunal



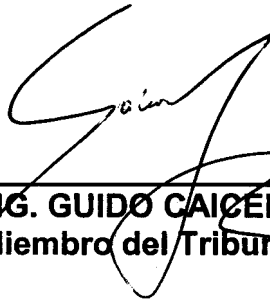
---

**ING. CARLOS VALERO**  
Director del Proyecto



---

**ING. REBECA ESTRADA**  
Miembro del Tribunal



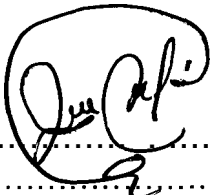
---

**ING. GUIDO CAICEDO**  
Miembro del Tribunal

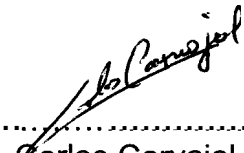
## DECLARACION EXPRESA

"La responsabilidad por los hechos, ideas y doctrinas expuestos en esta tesis, nos corresponden exclusivamente; y, el patrimonio intelectual de la misma, a la ESCUELA SUPERIOR POLITECNICA DEL LITORAL"

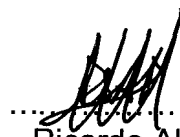
(Reglamento de Exámenes y Títulos profesionales de la ESPOL)



.....  
.....  
Nubia Zambrano



.....  
.....  
Carlos Carvajal



.....  
.....  
Ricardo Alvarez

## **RESUMEN**

La elección de la arquitectura de software es un elemento clave para el diseño de determinada solución computacional. Este documento muestra el desarrollo de una aplicación cliente/servidor usando CORBA, una tecnología para el desarrollo de sistemas distribuidos que cada vez madura más. Se hace mención de las diferentes alternativas para el desarrollo de aplicaciones cliente/servidor, centrandó la atención en CORBA y los ORB's . Basados en esto se crea un prototipo de una aplicación de registros estudiantiles en el que intervienen un servidor de bases de datos, un web server, y como middleware CORBA y sus llamadas a métodos estáticos haciendo uso del servicio de nombramiento, teniendo como lenguaje de programación Java y los controles Swing. Con este desarrollo se anotan las conclusiones de las ventajas y desventajas del uso de esta tecnología.

## INDICE GENERAL

<b>RESUMEN .....</b>	<b>VI</b>
<b>INDICE GENERAL.....</b>	<b>VII</b>
<b>INTRODUCCION.....</b>	<b>1</b>
<b>I.. TECNOLOGÍA CLIENTE-SERVIDOR .....</b>	<b>2</b>
1.1 HISTORIA DE LOS SISTEMAS DISTRIBUIDOS.....	2
1.1.1 <i>Sistemas Monoliticos y Mainframes</i> .....	2
1.1.2 <i>Arquitectura Cliente/Servidor</i> .....	3
1.1.2.1 <i>Pero... Que es realmente Cliente/Servidor?</i> .....	5
1.1.3 <i>Multitier Cliente/Servidor</i> .....	6
1.1.4 <i>Sistemas Distribuidos</i> .....	8
1.2 OBJETOS DISTRIBUIDOS .....	10
1.3 ALTERNATIVAS PARA LA CONSTRUCCIÓN DE SISTEMAS DISTRIBUIDOS .....	11
1.3.1 <i>CORBA</i> .....	12
1.3.2 <i>DCOM</i> .....	12
1.3.3 <i>DCE</i> .....	13
1.4 CARACTERÍSTICAS DEL SISTEMA OPERATIVO .....	15
1.5 MIDDLEWARE.....	16
1.5.2 <i>Java Applets</i> .....	17
1.5.3 <i>Thread</i> .....	18
<b>II.- CORBA (COMMON OBJECT REQUEST BROKER ARCHITECTURE).19</b>	
2.1 CORBA, CONCEPTOS FUNDAMENTALES.....	19
2.2 ORB .....	20
2.4 IDL .....	21
2.5 MODELO DE COMUNICACIONES DE CORBA .....	22
2.5.1 <i>Caracteristicas</i> .....	22
2.5.2 <i>Inicialización de CORBA ¿Cómo un componente encuentra su ORB?</i> .....	23
<b>III. CORBA/JAVA .....</b>	<b>24</b>
3.1 ARQUITECTURA DEL ORB .....	24
3.1.1 <i>Object implementation</i> .....	25
3.1.2 <i>Cliente</i> .....	25
3.1.3 <i>Object Request Broker (ORB)</i> .....	25
3.1.4 <i>Interfaz de ORB</i> .....	26
3.1.5 <i>CORBA IDL stubs and skeletons</i> .....	26
3.1.6 <i>Dynamic Invocation Interface (DII)</i> .....	26
3.1.7 <i>Object Adapter</i> .....	27
3.2 INVOCACIÓN DE LOS METODOS .....	28

EL CLIENTE PUEDE INVOCAR A LOS METODOS DEL SERVER POR MEDIO DEL ORB DE CORBA, MEDIANTE DOS FORMAS: .....	28
3.2.1 <i>Invocaciones estáticas</i> .....	28
3.2.2 <i>Invocaciones dinámicas</i> .....	28
3.3 INTERFACES DE INVOCACIONES DINÁMICAS .....	29
3.4 SERVICIO DE NOMBRAMIENTO .....	30
<b>IV. APLICACIÓN DE LA TECNOLOGÍA CLIENTE/SERVIDOR .....</b>	<b>32</b>
4.1 DESCRIPCIÓN DEL PROYECTO .....	32
4.1.1 <i>Dominio del Sistema</i> .....	32
4.1.2 <i>Funciones y Objetivos</i> .....	32
4.2 MODELO DE CASOS DE USO .....	34
4.2.1 <i>Casos de Uso</i> .....	35
4.2.2 <i>Actores</i> .....	40
4.3 REQUERIMIENTOS NO FUNCIONALES .....	40
4.4 MODELOS DE ANÁLISIS .....	41
4.4.1 <i>Modelo de Objetos</i> .....	41
4.4.2 <i>Análisis de Escenarios</i> .....	42
4.4.3 <i>Diagramas de Interacción de Objetos</i> .....	47
4.5 MODELO DE INTERFACES DE USUARIO .....	57
4.5.1 <i>Flujo de Ventanas</i> .....	57
4.5.2 <i>Layouts</i> .....	58
4.6 MODELOS DE DISEÑO .....	73
4.6.1 <i>Arquitectura del sistema</i> .....	73
4.7 IMPLEMENTACIÓN .....	75
4.7.1 <i>Ambiente de Desarrollo</i> .....	75
4.7.1.1 <i>VisualAge</i> .....	75
4.7.1.2 <i>Visibroker for Java</i> .....	76
4.7.1.3 <i>Java Development Kit (JDK)</i> .....	76
4.7.1.4 <i>SQL</i> .....	77
4.7.2 <i>Código fuente</i> .....	77
<b>CONCLUSIONES Y RECOMENDACIONES .....</b>	<b>82</b>
<b>BIBLIOGRAFÍA .....</b>	<b>83</b>



## **INTRODUCCION**

El esquema cliente-servidor es un modelo de computación en el que el procesamiento requerido para ejecutar una aplicación o conjunto de aplicaciones relacionadas se divide entre dos o más procesos que cooperan entre sí. Usualmente la mayoría del trabajo pesado se hace en el proceso llamado servidor, y el (los) proceso(s) cliente(s) solo se ocupa de la interacción con el usuario (aunque esto puede variar).

Los principales componentes del esquema cliente-servidor son entonces los **Clientes**, los **Servidores** y la **infraestructura de comunicaciones**.

Este documento plantea la construcción de un sistema cliente/servidor utilizando CORBA, y en función de esta tecnología de desarrollo de sistemas distribuidos se seleccionaron los componentes y herramientas a utilizar.

CORBA implica manejo de objetos, por tanto fue necesario un diseño orientado a objetos y la utilización de una herramienta de desarrollo que lo soporte, como Java. La selección del sistema operativo, requerimientos mínimos de hardware, herramientas de desarrollo y manejo de datos como lo son: JDK v1.1.7, Visibroker v3.4, Visual Age for Java v2.0, NetScape Communicator v4.5.

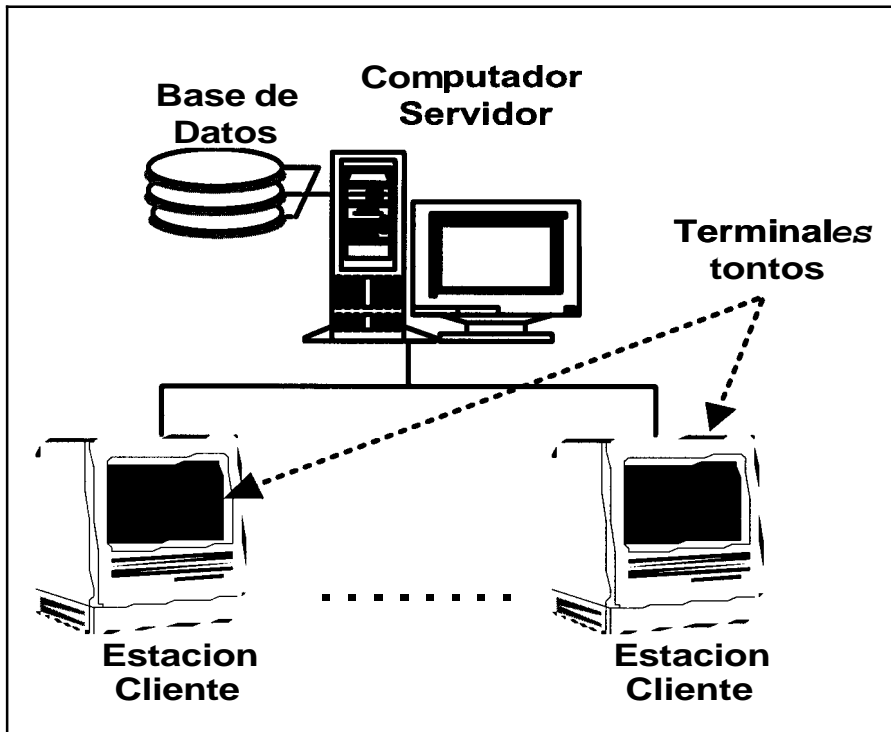
# **I. - TECNOLOGÍA CLIENTE-SERVIDOR**

## **1.1 Historia de los Sistemas Distribuidos**

### **■1.1 Sistemas Monolíticos y Mainframes**

En un principio se tenía a los mainframe's con bases de datos jerárquicas y terminales tontos. Estos generalmente eran costosos y difíciles de mantener, pero sin embargo fueron capaces de soportar a gran número de usuarios y tener la ventaja (o desventaja dependiendo del punto de vista) de ser manejados centralmente.

Las aplicaciones escritas para los mainframes fueron monolíticas, es decir la interfaz de usuario, la lógica del negocio y la lógica de acceso a datos toda estaba contenida en una gran aplicación, entonces como los terminales de acceso a las aplicaciones eran 'tontos' toda la aplicación corría en el servidor. Una aplicación monolítica típica está descrita en la figura No. 1.



**Figura No.1**      **Arquitectura de sistemas monolíticos**

### 1.1.2 Arquitectura Cliente/Servidor

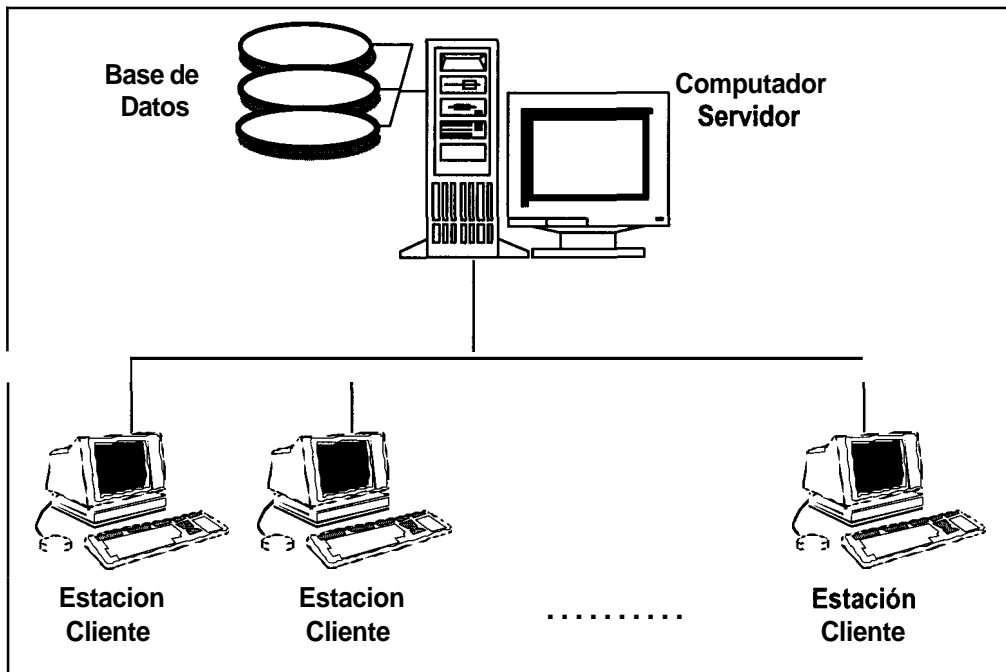
Con el advenimiento de los PC's se rompió el paradigma de la arquitectura monolítica de las aplicaciones basadas en los mainframes. Mientras estas aplicaciones requerían que el servidor en sí ejecutara todo el procesamiento, las aplicaciones basadas en la arquitectura cliente/servidor permitían que algún procesamiento sea llevado a cabo por los PC's de los usuarios.

Con la revolución cliente/servidor vino la proliferación de los servidores UNIX; muchas aplicaciones sencillamente no requerían el poder masivo de los mainframes, y debido a que la arquitectura cliente/servidor fue capaz de mover mucha de la carga de procesamiento al lado del PC, por tanto estos

pequeños servidores UNIX fueron mucho más efectivos y menos costosos que los mainframes. También estas máquinas fueron mucho más adaptables a pequeños negocios que los mainframes, que simplemente estaban fuera del alcance de estos. Otro de los beneficios fue el reforzamiento de los departamentos dentro de la organización para poder manejar su propio servidor, lo que les permitía desarrollar sus propias aplicaciones sin tener que pasar por el departamento que controlaba los mainframes para que desarrolle sus aplicaciones y, finalmente mientras los terminales tontos solo se limitan a ejecutar aplicaciones en el mainframe, los PC's también pueden ejecutar otras tareas independientemente del mainframe incrementando así la utilidad de las máquinas de escritorio.

Las aplicaciones cliente/servidor típicamente distribuyen los componentes de la aplicación de modo que la base de datos debería residir en el servidor (con un UNIX o mainframe), la interfaz de usuario debería residir en el cliente y, la lógica del negocio debería residir en alguno de los dos o ambos componentes. Por tanto si se hacen cambios al componente cliente, tiene que ser distribuido a cada usuario.

Esta arquitectura es conocida como *Two-tier* y es ilustrada en la figura No. 2.



**Figura No.2 Arquitectura cliente/servidor two-tier**

### 1.1.2.1 Pero... Que es realmente Cliente/Servidor?

Cliente/Servidor es una arquitectura de diseño de software de aplicación que es el resultado de la subdivisión de un sistema de información, en un conjunto de procesos servidores, generalmente especializados, que pueden ejecutarse en variadas plataformas (hardware + software), es decir que tenga interoperabilidad; y que provee un gran número de procesos clientes, sobre diferentes plataformas físicamente interconectadas por una red local o de área extendida, utilizando uno o varios protocolos de comunicación.

Cientes y Servidores, son dos entidades lógicamente separadas, que operan juntas sobre una red de trabajo para complementar una tarea. Toda

herramienta o proceso a nivel cliente, se lo denomina “Front End”, mientras que el proceso servidor con el que se conecta es llamado “Back End”.

Ciente/Servidor le da la libertad y flexibilidad de mezclar e igualar componentes en cualquier nivel, reuniendo variedad de combinaciones de clientes y servidores, siendo clave la manera de unir las piezas y hacerlas trabajar en conjunto; escogiendo las plataformas adecuadas tanto para los procesos clientes o procesos servidores, los protocolos de red, la infraestructura de computación distribuida, considerando estas como las decisiones mas sencillas ya que las mas complejas se daran a nivel de desarrollo de aplicaciones y herramientas de cliente/servidor.

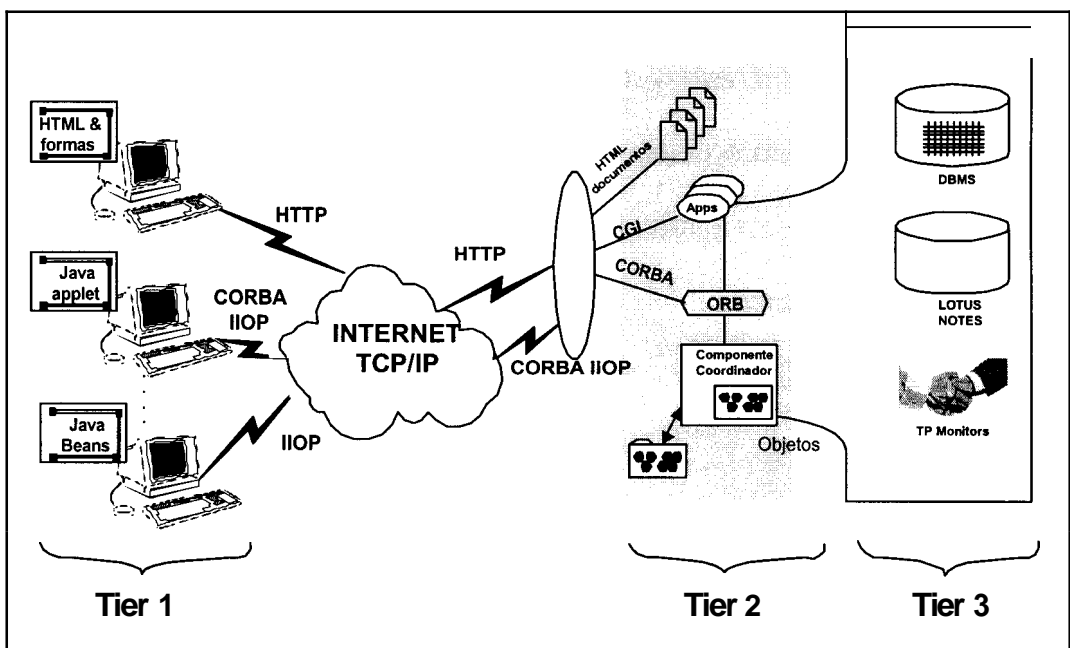
A lo largo de este tiempo se han buscado diversas modalidades de soluciones de software para red de area local, soluciones que se venden como paquetes comerciales que se distinguen entre si por la naturaleza del servicio que ofrecen a sus clientes ejemplos de estos paquetes son: Servidores de bases de datos, Servidores de transacciones, Servidores de groupware, Servidores de objetos y Servidores de web.

### **1.1.3 Multitier Cliente/Servidor**

Aunque la arquitectura cliente/servidor solucionaba problemas de las aplicaciones basadas en los mainframes, estas no estaban libres de fallas en si. Debido a que la funcionalidad de acceso a las bases de datos y la lógica

del negocio estaban frecuentemente contenidas en el componente cliente, cualquier cambio que se haga en uno de estos componentes o en la base en si significaba el desarrollo de un componente nuevo para ser instalado a todos los usuarios de la aplicacion.

Este esquema basico cliente/servidor comunmente llamado "two-tier" fue entonces reemplazado por el esquema cliente/servidor multitier. El termino "Tier " significa o representa una capa dentro del esquema, es decir una aplicacion puede ser two-tier o three-tier o n-tier; siendo el mas famoso y usado de estos esquemas el three-tier, que divide el sistema en tres tiers o capas lógicas: la interface de usuario, la capa de las reglas del negocio, y la capa de acceso a los datos, como se muestra en la figura No. 3.



**Figura No.3** Arquitectura cliente/servidor three-tier

La arquitectura cliente servidor multicapa o multitier mejora la arquitectura basada en dos capas porque hace la aplicación menos frágil ya que se aísla al cliente del resto de la aplicación, lo que le permite mayor flexibilidad en la implantación de la misma. De este modo la interfaz de usuario se comunica solamente con la capa de las reglas del negocio, jamás directamente con la capa de acceso a datos. La capa de las reglas del negocio se comunica con la interfaz de usuario por un lado y con la capa de acceso a los datos en el otro, en consecuencia los cambios hechos en la capa de acceso a datos no afectarán la interfaz del usuario debido a que están aisladas la una de la otra.

Los componentes servidores de una aplicación multicapa pueden estar corriendo en una misma máquina o en varias máquinas servidoras, así múltiples componentes de la lógica del negocio o múltiples componentes de acceso a datos (si se usan varias bases de datos) se pueden crear para una aplicación, distribuyendo la carga de procesamiento, obteniendo así una aplicación más robusta y escalable.

#### **1.1.4 Sistemas Distribuidos**

La siguiente evolución de las arquitecturas de aplicación son los sistemas distribuidos. Esta arquitectura toma el concepto de cliente/servidor multicapa como su consecuencia natural. Más que diferenciar de lógica del negocio y acceso a datos, el modelo de sistemas distribuidos simplemente



expone toda la funcionalidad de la aplicación como objetos, cada uno de los cuales puede usar los servicios proporcionados por los otros objetos en el sistema, o aun otros objetos de otros sistemas. La arquitectura también puede hacer borrosa la distinción de cliente y servidor debido a que el componente cliente también puede crear objetos que cumplan el rol de servidor. Esta arquitectura proporciona el máximo en flexibilidad.

Esta arquitectura debe su flexibilidad al reforzamiento en la definición de las interfaces de los componentes. La interface de un componente especifica a otros componentes los servicios que son ofrecidos por ese componente y como estos pueden ser usados. Debido a que la interface de un componente se mantiene constante, la implementación del mismo puede ser cambiada dramáticamente, sin afectar a otros componentes. Por ejemplo, un componente que proporcione información de clientes para una compañía puede guardar esa información en una base de datos relacional, luego los diseñadores pueden decidir que una base de datos orientada a objetos sería lo más apropiado. De esta forma los diseñadores de aplicación pueden hacer cambios en la implementación de los componentes, incluso como cambiar el tipo de base de datos, manteniendo la interface del componente intacta. Es decir, la implementación de los componentes es libre de cambiar.

Una interface define el protocolo de comunicación entre dos componentes

separados de un sistema. Estos componentes pueden ser procesos separados, objetos separados, una aplicación de usuario, o cualquier entidad separada que necesite comunicarse con otra. La interface describe que servicios proporciona un componente y el protocolo para usar estos servicios; en el caso de un objeto, la interface se puede pensar como el conjunto de métodos definidos por ese objeto, incluyendo los parámetros de entrada y salida.

Los sistemas distribuidos son realmente sistemas cliente/servidor multicapa en que el número de clientes y servidores son potencialmente grandes. Estos sistemas generalmente proveen de servicios adicionales, tales como directorio de servicios, que permite a varios componentes de la aplicación ser localizados por otros.

El directorio de servicios se refiere al conjunto de servicios que ofrecen los objetos, que pueden ser objetos, o aun personas; para ser localizados por otros objetos. No solo los objetos pueden ser de diferente tipo sino que la información en si puede variar, por ejemplo un libro de teléfonos puede ser usado para buscar números telefónicos y direcciones postales, esta información estará agrupada por grupos relacionados; es decir es como las páginas amarillas.

## **1.2 Objetos Distribuidos**

Los objetos distribuidos son piezas inteligentes de software que pueden

intercambiar mensajes transparentemente en cualquier lugar en el mundo. El lenguaje y compilador que se usa para crear servidores de objetos distribuidos son totalmente transparentes a sus clientes, estos no necesitan saber donde reside el objeto distribuido ni el sistema operativo en el que se ejecuta; y pueden estar en la misma maquina o bien en cualquier lugar de la red mundial.

Los objetos distribuidos por ser objetos gozan de todas las bondades que estos proveen, por lo que cualquier sistema de objetos proporciona las tres propiedades básicas: encapsulación, herencia y polimorfismo.

El interes fundamental de los objetos distribuidos esta en como esta tecnologia puede ser extendida para enfrentarse con problemas complejos que son heredados al crear sistemas cliente/servidor robustos y de imagen simple, es decir como los objetos pueden trabajar juntos a traves de una maquina y los limites de una red para crear soluciones cliente servidor.

Nos encaminamos a los sistemas cliente servidor basados en objetos en donde se apreciara lo mejor de la grandeza y potencial de estos.

### **1.3 Alternativas para la construcción de sistemas distribuidos**

Cuando diseñamos e implementamos aplicaciones distribuidas, ciertamente CORBA no es la unica elección del desarrollador, existen otros mecanismos con los que se puede construir este tipo de aplicaciones. Dependiendo de

la naturaleza de la aplicación (complejidad de la plataforma, lenguaje de programación, etc.) hay varias alternativas a considerar por el desarrollador.

### 1.3.1 CORBA

CORBA (Common Object Broker Architecture), es una especificación normativa de un conjunto de industrias informáticas asociadas con el nombre de OMG. Esta norma cubre todos los ámbitos de los sistemas de objetos distribuidos.

El estándar CORBA define un entorno para el desarrollo de aplicaciones distribuidas. Esta arquitectura hace posible que la programación en red sea mucho más fácil, permitiendo crear aplicaciones distribuidas que interactúen con otras en cualquier lenguaje de programación corriendo sobre diferentes máquinas.

CORBA es una plataforma que ofrece servicios como transacciones, seguridad, eventos, servicio de nombramiento, persistencia y ciclo de vida.

### 1.3.2 DCOM

DCOM (Distributed Component Object Model) es un conjunto de conceptos de Microsoft y interfaz de programa en que el objeto cliente puede requerir servicios desde el objeto servidor o de otras computadoras en una red. El Component Object Model (COM) proveen un conjunto de interfaces, permitiendo a clientes y servidores con el mismo computador. (corriendo en

sistemas Windows 95 o Windows NT).

DCOM puede trabajar sobre una red dentro de una empresa o sobre una red pública, como lo es el Internet. Usa TCP/IP y HTTP. DCOM viene como parte de NT 4.0 y esta agregado en el sistema operativo Windows 95.

DCOM es generalmente equivalente a CORBA en termino de proveer un conjunto de servicios distribuidos.

### **1.3.3 DCE**

DCE (Distributed Computing Environment) o “Ambiente de Computación Distribuida”, es un conjunto integrado de sistemas y servicios, esto consiste de varias herramientas para crear, usar, y mantener aplicaciones distribuidas. Es portátil y flexible, tal que la referencia (codigo fuente) para su implementación sea independiente de las redes y de los sistemas operativos. DCE ofrece un conjunto de servicios organizados en dos categorias: servicios fundamentales y servicios para compartir datos. Los primeros incluyen herramientas para el desarrollo de software tales como RPC, servicios de nombres, seguridad, tiempo y threads. Los segundos proveen al usuario final manejo de archivos distribuidos y soporte sin disco. Estos servicios son portables a muchos computadores, porque estan escritos en codigo C y son soportados por los servidores DCE en una red.

Finalmente, DCE apoya la portabilidad e interoperabilidad proveyendo al diseñador con capacidades que esconden diferencias entre hardware,

software y elementos de gestión de redes. Por ejemplo el RPC automáticamente convierte datos usados por una computadora al formato usado por otra.

DCE, es un software diseñado y mantenido por la OSF (Open Software Foundation), y actualmente esta no utilizado.

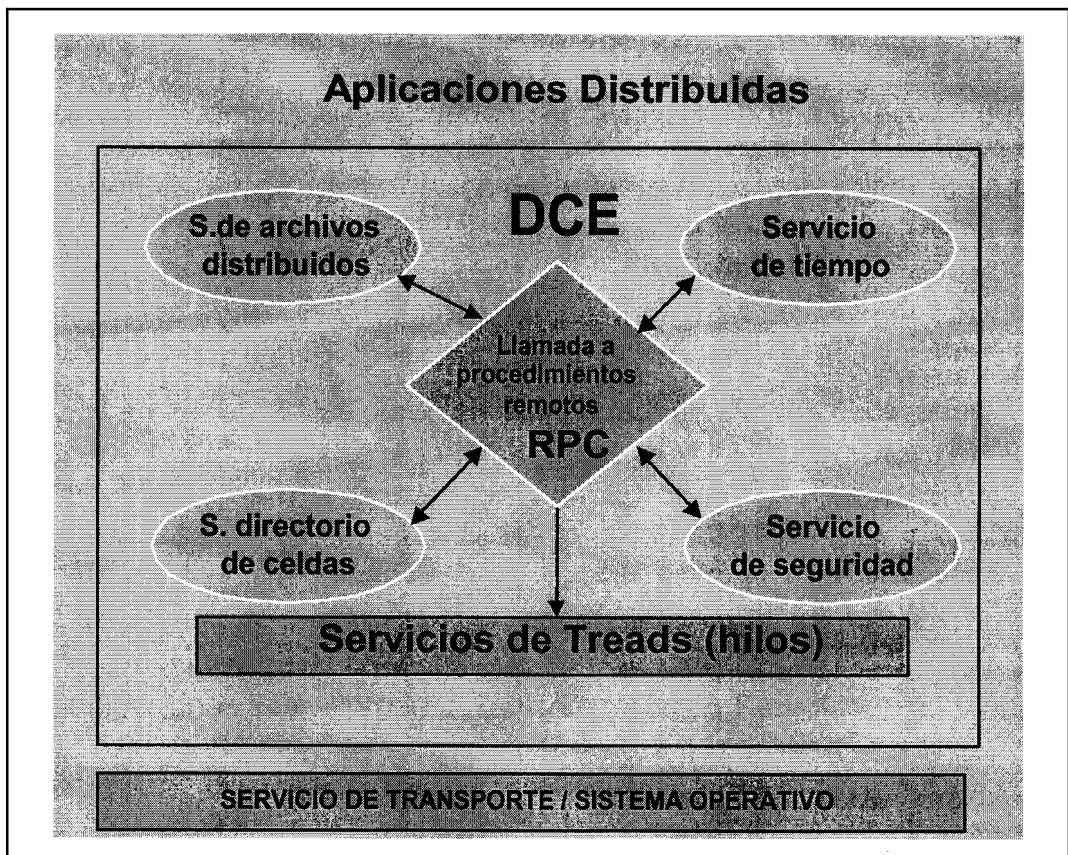


Figura No. 4 DCE

## 1.4 Características del Sistema Operativo

Los servidores requieren un alto nivel de concurrencia, debido a que, idealmente, una tarea separada debe ser asignada a cada uno de los clientes.

La Multitarea es la manera natural de simplificar la codificación de aplicaciones complejas que pueden ser divididas en una colección de tareas concurrentes y lógicamente distintas, a las que se denominan co-rutinas o threads.

Las siguientes serían las características o requerimientos necesarios:

Tareas colaborativas.

Prioridad entre tareas.

Semaforos.

Comunicaciones inter – proceso, tanto Local como Remotas.

Threads.

Protección Intertareas.

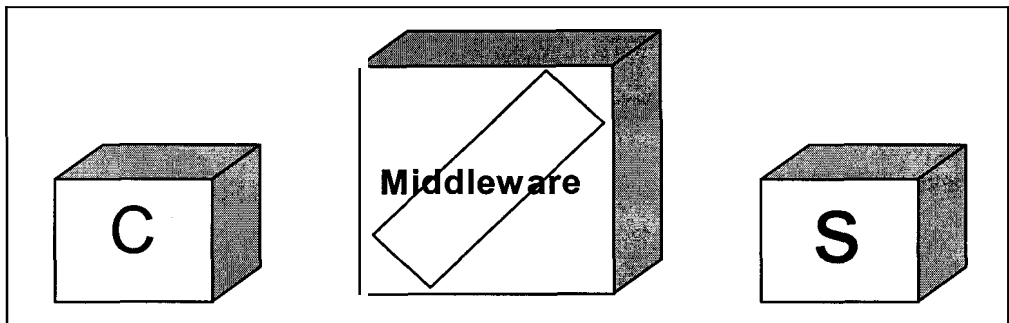
Sistema de archivo de alto desempeño y multiusuario.

Administración de memoria eficiente.

Instrucciones de ejecución con enlaces dinámicos.

## 1.5 Middleware

Middleware es un termino que cubre todo el software distribuido necesitado para soportar interacciones entre clientes y servidores. Es el software que esta en medio del sistema cliente/servidor.



**Figura No. 5 Middleware**

### 1.5.1 ¿ Donde comienza y termina el middleware?

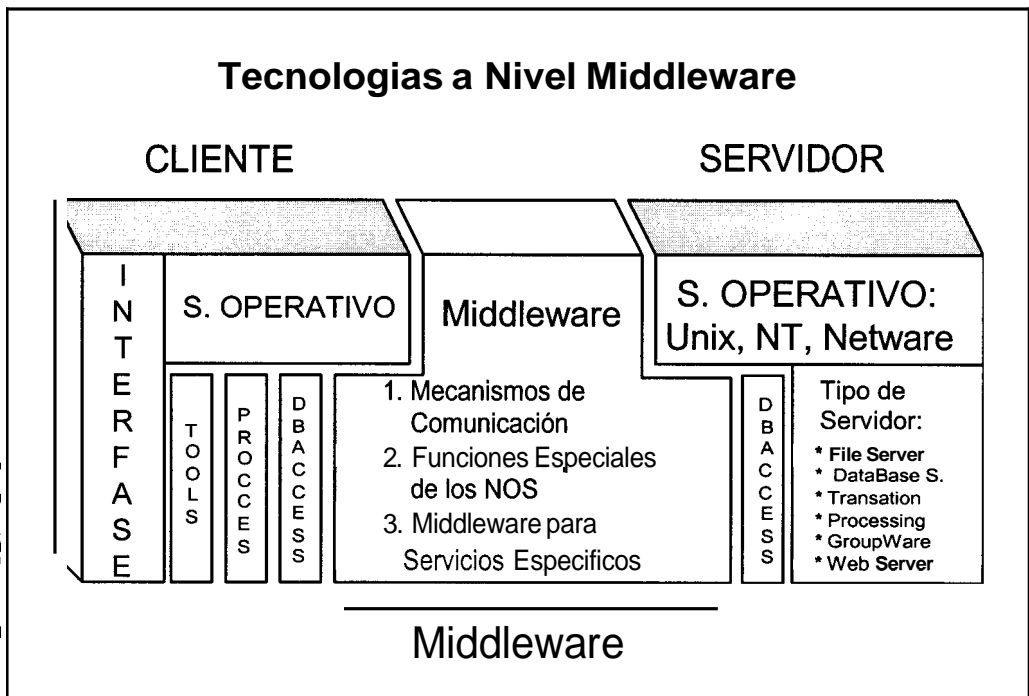
Comienza con el conjunto de API del lado del cliente que es usado para invocar un servicio, y cubre la transmision de los requerimientos sobre la red y los resultados. Middleware no incluye el software que provee los servicios comunes, es decir lo propio del servidor. Tambien no incluye la interfaz de usuario o la lógica de la aplicacion, es decir lo que pertenece al cliente.

Se divide al middleware en dos clases:

- **Middleware general.-** Es el substrato para muchas interacciones cliente/server. Incluye gran cantidad de comunicacion, directorios distribuidos, servicios de autentificacion, tiempo de trabajo de la red, llamadas a procedimientos remotos, y servicios encolados.



- Middleware de servicios específicos.- Necesitado para llevar a cabo un particular tipo de servicio cliente/server
  - Database - especifica middleware tales como ODBC, DRDA EDA/SQL, SAG/CLI, y Oracle



**Figura No. 6 Descripción del Middleware**

### 1.5.2 Java Applets

Un applet no es una aplicación, sino más bien un componente que corre en un ambiente de browser. Estos permiten crear aplicaciones tamaño componente que los servidores puedan cargar sobre clientes usando páginas HTMLs.

### 1.5.3 Thread

Servicio de threads: Permite multiples secuencias o flujos de control, lo cual en particular **permite** ejecutar varios servicios simultaneamente. Cada thread es esencialmente un camino independiente entre un cliente y un servidor, permitiendo a un cliente interactuar con muchos servidores y viceversa (en el contexto de sistemas distribuidos). El servicio de threads incluye operaciones para crear y controlar multiples threads en un sólo proceso y para sincronizar el acceso a datos globales.

Cada tarea puede tener uno o mas threads, que son la unidad minima de ejecucion de un programa. Los threads comparten los recursos asignados a la tarea a que pertenecen.

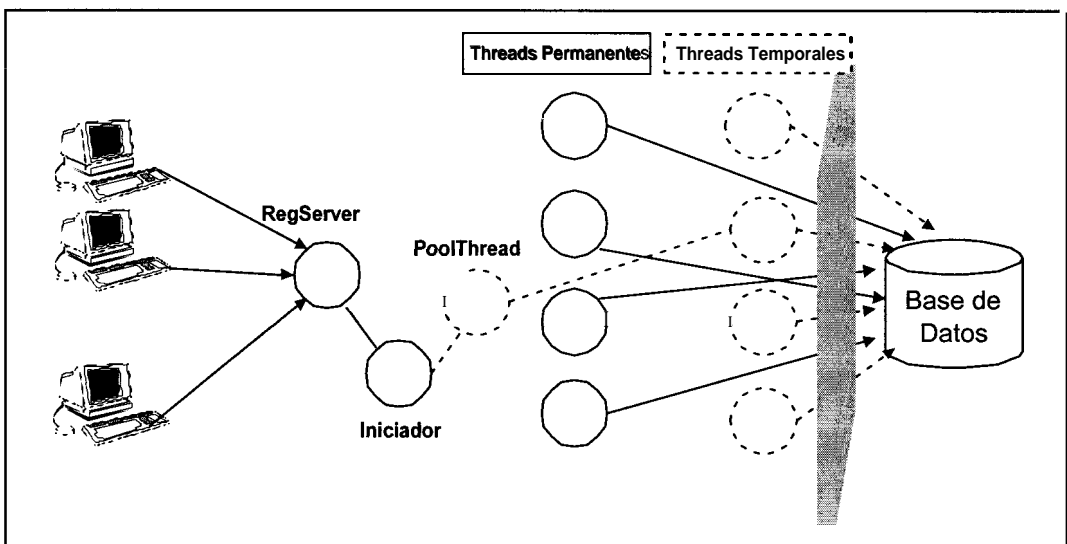


Figura No. 7 Threads

## **II.- CORBA (Common Object Request Broker Architecture)**

### **2.1 CORBA, conceptos fundamentales.**

CORBA, es un middleware que permite conectar clientes y servidores heterogeneos (tanto en el lenguaje de programacion como en la plataforma de ejecucion). Esto se consigue de forma transparente para los clientes y servidores. Los clientes simplemente invocan un método de un objeto servidor y no tienen que preocuparse de la localización de dicho objeto.

El encargado de la traduccion y transporte de la llamada es el ORB (Object Request Broker), una pieza fundamental de la arquitectura de CORBA.

Y todo esto se consigue gracias al IDL (Interface Definition Language), que es una especie de lenguaje de programacion sin llegar a serlo del todo. Mediante IDL se especifica que servicios (es decir, sus metodos) ofrece cada objeto servidor.

CORBA es una arquitectura orientada a objetos. Los objetos CORBA muestran características de otros sistemas orientados a objetos, incluyendo herencia de interfaces y polimorfismo, y además provee la posibilidad de ser usado con otros lenguajes orientados a objetos tales como C o Cobol.

## 2.2 ORB

Un ORB provee los mecanismos requeridos por los objetos para comunicarse con otros objetos a través de lenguajes heterogéneos, herramientas, plataformas y redes. También provee el entorno para manejar estos objetos, advertir su presencia y describir sus metadatos. Un ORB CORBA es un bus de objetos que se describe así mismo.

Usando un ORB, un objeto cliente puede invocar transparentemente un método en un objeto servidor, el cual puede estar en la misma máquina o a través de la red, el ORB intercepta la llamada y es responsable de encontrar el objeto que implemente la petición, pasarle los parámetros, invocar el método, y regresar los resultados. Los papeles de cliente/servidor son solo usados para controlar las interacciones entre los objetos, ya que los objetos en el ORB pueden actuar a la vez como cliente y como servidor.

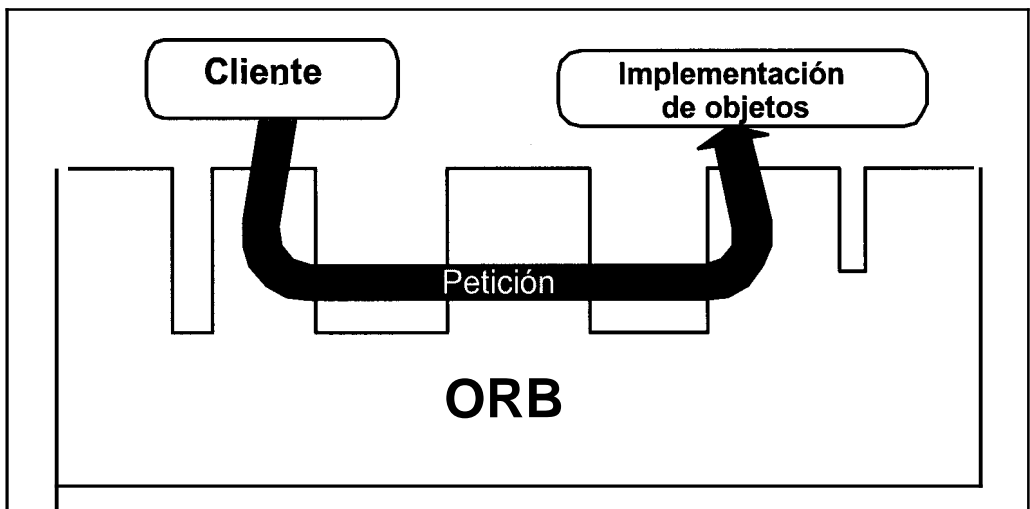


Figura No. 8 ORB

## 2.4 IDL

IDL (Interface Definition Language), que es una especie de lenguaje de programación sin llegar a serlo del todo. El IDL no es un lenguaje de programación en sí, es decir, no se pueden escribir aplicaciones en IDL; el único propósito del IDL es definir interfaces. Mediante IDL se especifica que servicios (es decir, sus métodos) ofrece cada objeto servidor. Usando un compilador de IDL, se genera el código necesario en un lenguaje de programación determinado para el lado cliente (los stubs) y el lado servidor (skeletons) que implementa dicha funcionalidad.

Es decir en el lado cliente el código generado se usa en la implementación del cliente como cualquier librería convencional (por ejemplo, una librería para el manejo de sockets) que hace ver al cliente el objeto(s) servidor y sus servicios. En la parte servidora se genera código que recibe las peticiones locales o vía red de los clientes y se encarga de llamar a las funciones correctas del servidor cuyas cabeceras/esqueletos también se han generado, debiendo ser implementado el código de los métodos lógicamente por el programador.

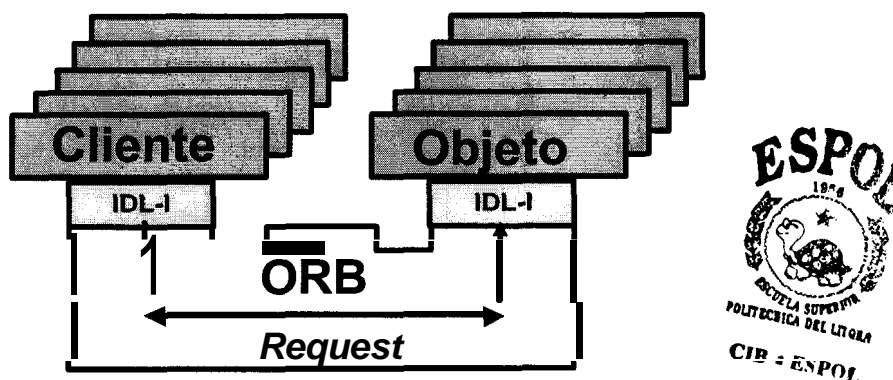


Figura No. 9 IDL

## 2.5 Modelo de Comunicaciones de CORBA

CORBA utiliza la noción de *referencias de objeto*, que también son llamados IOR (Interoperable Object Reference) o IORs para facilitar la comunicación entre objetos. Cuando un componente de una aplicación quiere acceder a un objeto CORBA, primero obtiene un IOR para ese objeto; usando el IOR del objeto, el componente (llamado cliente del objeto) puede entonces invocar métodos en el otro objeto (llamado servidor en esta instancia).

Los ORBs de CORBA usualmente se comunican usando el protocolo IIOP, existen otros protocolos pero el IIOP (Internet InterORB Protocol) es más popular.

### 2.5.1 Características

Toda implementación de CORBA tiene las siguientes características

Sigue el modelo de objeto de CORBA

Sigue la arquitectura de CORBA

Utiliza la semántica y la sintaxis del IDL de OMG

Implementa los siguientes componentes:

- DII: Dynamic Invocation Interface
- DSI: Dynamic Skeleton Interface
- Interface Repository
- ORB Interface
- Basic Object Adapter

### **2.5.2 Inicialización de CORBA ¿Cómo un componente encuentra su ORB ?**

Las llamadas que un objeto debe seguir para inicializarse así mismo son:

1. Obtener la referencia al objeto de su ORB.
2. Obtener el puntero a su adaptador de objetos.
3. Descubrir que servicios iniciales están disponibles.
4. Obtener las referencias a los objetos de los servicios que se desean.

Para cada uno de los pasos anteriores se debe llamar el método apropiado.

Un objeto puede inicializarse así mismo en más de un ORB.

### III. CORBA/JAVA

En la actualidad CORBA Y JAVA son dos tecnologías que se complementan haciendo uso de las ventajas que ofrecen cada una de ellas.

Con un Java ORB, un applet Java ordinario puede invocar directamente metodos en objetos CORBA usando el protocolo IIOp sobre el Internet.

#### 3.1 Arquitectura del ORB

La siguiente figura ilustra los componentes principales en la arquitectura de CORBA ORB.

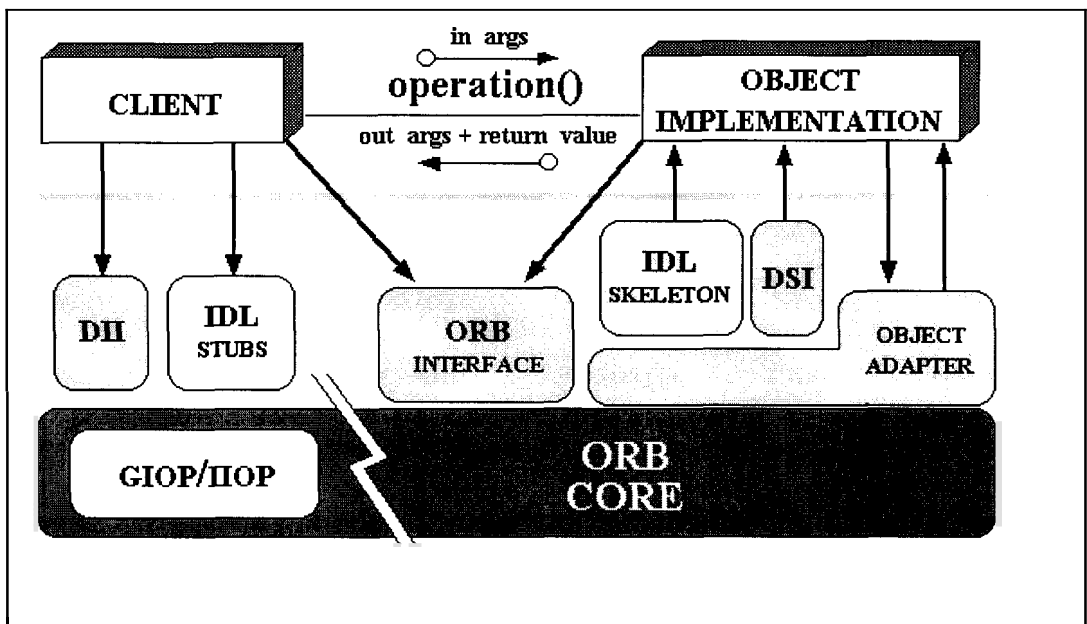


Figura No. 10 Arquitectura del ORB



### **3.1.1 Object implementation**

Define las operaciones que ponen en marcha la interfaz del CORBA IDL. La implementación de los objetos puede ser escrita en una variedad de lenguajes incluyendo C, C++ y java.

### **3.1.2 Cliente**

Es la entidad de un programa que invoca una operación en una implementación de un objeto. Tener acceso a los servicios de un objeto remoto debe ser transparente al llamador. Debe ser tan simple como llamando un método en un objeto, es decir `obj --> op(args)`. Los restantes componentes que aparecen en la figura ayudan a soportar este nivel de transparencia.

### **3.1.3 Object Request Broker (ORB)**

El ORB proporciona un mecanismo para una comunicación transparente entre peticiones de clientes y las implementaciones de los objetos. El ORB simplifica la programación distribuida despreocupando al cliente de los detalles de las llamadas del método. Esto hace que las peticiones del cliente aparezcan como llamadas locales del procedimiento. Cuando un cliente invoca una operación, el ORB es responsable de encontrar la implementación del objeto, activándola transparentemente en caso de ser necesario, entregando la petición al objeto y devolviendo cualquier respuesta al llamador.

### **3.1.4 Interfaz de ORB**

Un ORB es una entidad lógica que se puede poner en ejecución de varias maneras (tales como uno o más procesos o un conjunto de librerías). Para despreocupar las aplicaciones de los detalles de implementación, CORBA define una interfaz abstracta para un ORB. Esta interfaz proporciona varios ayudantes de funciones que convierten las referencias de objetos a strings y viceversa y crean listas de argumentos para las peticiones hechas a través de interfaces de invocación dinámica descritas más adelante.

### **3.1.5 CORBA IDL stubs and skeletons**

Los stubs y los skeletons del CORBA IDL sirven de "goma" entre el cliente y las aplicaciones del servidor, respectivamente y el ORB. La transformación entre las definiciones de CORBA IDL y el lenguaje de programación es automatizada por un compilador de CORBA IDL. El uso de un compilador reduce el potencial de inconsistencias entre los stubs del cliente y los skeletons del servidor y aumenta las oportunidades para las optimizaciones automatizadas del compilador.

### **3.1.6 Dynamic Invocation Interface (DII)**

Esta interfaz permite que un cliente tenga acceso directamente a los mecanismos subyacentes de la petición proporcionados por un ORB. Las aplicaciones utilizan el DII para publicar dinámicamente peticiones a los objetos sin requerir enlace con stubs IDL específicos. Diferente de los stubs IDL (que permiten solo peticiones al estilo RPC), el DII también permite que

los clientes hagan llamadas sincronicas diferidas no-bloqueantes (operaciones de send y receive por separado) y unidireccionales (unicamente send).

### 3.1.7 Object Adapter

- Asiste al ORB con las peticiones que entregan a un objeto y con la activación del mismo. Mas importante aun, un Object Adapter asocia implementaciones de un objeto con el ORB. Los Object Adapters pueden ser especializados para proveer soporte de ciertos estilos de implementacion de objetos. Por ejemplo, Object Adapters para bases de datos orientadas a Objetos (OODB) y librerias de Object Adapters para objetos no remotos.
- El cliente para hacer un requerimiento utiliza una referencia de objetos para solicitar un método.
- Si el objeto servidor es remoto, la referencia apunta a una funcion stub (lado del cliente) y usa la estructura ORB para invocar al objeto servidor.
- Luego el stub por medio del ORB identifica la maquina donde corre el objeto servidor y se conecta a ella
- Finalmente el stub envía la referencia de y los parametros al codigo skeleton (lado del servidor)
- El skeleton transforma la llamada y parametros en una implementacion requerida.
- Object Adapter, es el medio que comunica el ORB con el objeto de implementacion

- El resultado es enviado por la misma vía

### **3.2 Invocación de los métodos**

El cliente puede invocar a los métodos del server por medio del ORB de Corba, mediante dos formas:

- Invocaciones estáticas
- Invocaciones dinámicas

#### **3.2.1 Invocaciones estáticas**

Se lo define como una interface, la cual contiene descripción:

- 1) Atributos (variables)
- 2) Métodos, que van a ser invocados por el cliente
- 3) Argumento de los métodos.

#### **3.2.2 Invocaciones dinámicas**

Antes de invocar dinámicamente un método en un objeto, primero debemos encontrar el objeto y obtener su referencia. Una vez que tengamos la referencia, debemos usar esta para recuperar las interfaces de los objetos y dinámicamente construir la petición. Se debe especificar la petición del método que usted quiera ejecutar y sus parámetros.

### 3.3 Interfaces de invocaciones dinamicas

Los servicios que se necesitan para invocar dinamicamente un objeto (figura 5), son parte del CORBA core. Se necesitan **4** interfaces para usar invocación dinamica:

**CORBA:: Objeto:** Es una interfaz pseudo-objeto, que define operaciones que todo objeto Corba puede soportar. Esta es la interfaz raiz para todos los objetos Corba.

**CORBA:: Requerimiento:** Es una interfaz pseudo-objeto, que define las operaciones en un objeto remoto.

**CORBA:: NVList:** Es una interfaz pseudo-objeto, que ayuda construir una lista de parametros. La interfaz NVList, define operaciones que ayudan a manipular una lista.

**CORBA:: ORB:** Es una interfaz pseudo-objeto, que define propositos generales de los metodos. Se pueden invocar estos metodos en un pseudo-objeto ORB de una implementación Cliente o Servidor.

### 3.4 Servicio de Nombramiento

El servicio de nombramiento de objetos es el principal mecanismo para que los objetos en un ORB localicen otros objetos. El servicio de nombramiento convierte los nombres humanos en referencias a objetos. Se puede opcionalmente asociar uno o mas nombres con la referencia a un objeto.

Se puede referenciar un objeto CORBA usando una secuencia de nombres que forme un arbol de nombre jerarquico. Resolver un nombre significa encontrar el objeto asociado con el nombre en un contexto dado. Enlazar un nombre es crear una asociacion nombre/objeto para un contexto particular.

Cada componente nombrado es una estructura con dos atributos: 1) un identificador que es el nombre del objeto; 2) el tipo que es una cadena en donde se puede poner una descripción del nombre del atributo.

Las interfaces *NamingContext* y *BindingIterator* implementan el servicio de nombrado. Se invoca el metodo **bind** en la interfaz *NamingContext* para asociar el nombre de un objeto con un contexto enlazado. De esta forma es como se implementa una jerarquia de nombrado. El metodo *list* permite iterar a traves de un conjunto de nombres (regresa un objeto *BindingIterator*). Se puede iterar invocando *next-one* y *next\_n* en el objeto regresado *BindingIterator*.

En resumen, el servicio de nombrado de objetos define las interfaces que permiten gestionar el espacio de nombres para los objetos, consultar y navegar a traves de ellos. Los objetos que proveen estas interfaces viven en

el ORB.

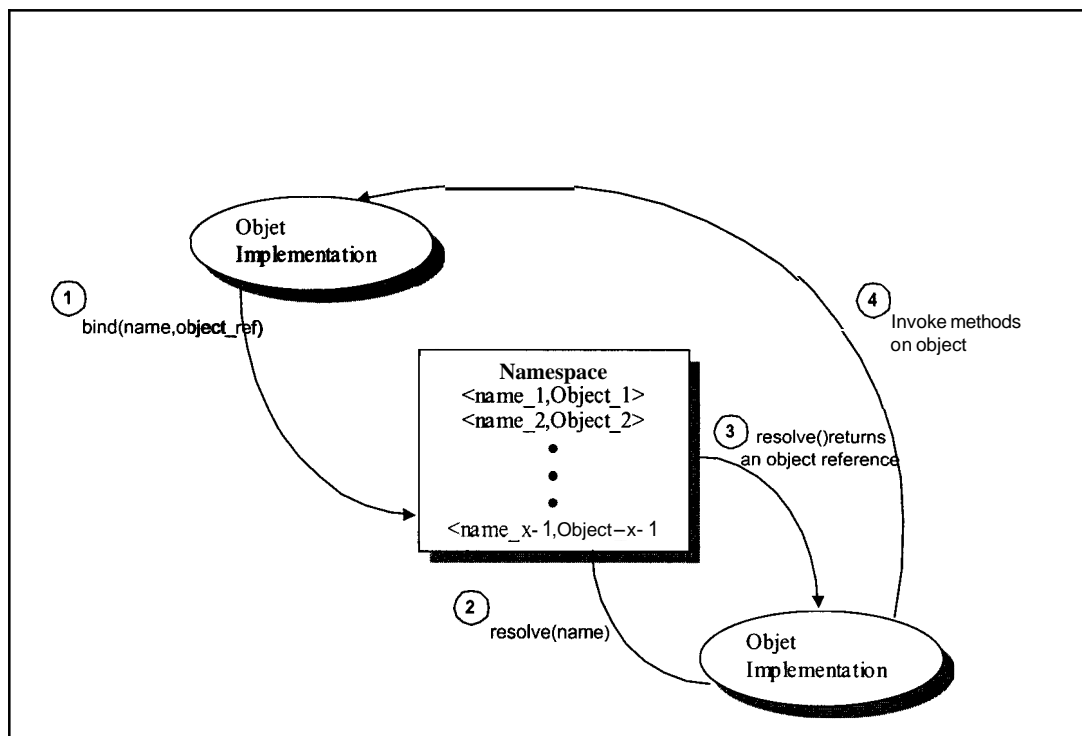


Figura No. 11 Servicio de Nombramiento



## **IV. APLICACIÓN DE LA TECNOLOGÍA CLIENTE/SERVIDOR**

### **4.1 Descripción del Proyecto**

#### **4.1.1 Dominio del Sistema**

El proyecto a realizar es el “Registro de estudiantes”, el sistema debera permitir el desarrollo del proceso de inscripción de estudiantes, ingreso de profesores, materias, paralelos y horarios. Se asumira que la informacion correspondiente a pagos y deudas para los registros ya esta ingresada en el sistema.

#### **4.1.2 Funciones y Objetivos**

Las características funcionales de la aplicación son las siguientes:

- El sistema debera tener una interface grafica con iconos
- El sistema debera estar disponible tanto para ser accesado desde una red local o desde el Internet.
- Soportara el ingreso de informacion de estudiantes, datos como nombre apellido, dirección telefono, cedula; y automaticamente se asignara un codigo de matricula al estudiante.



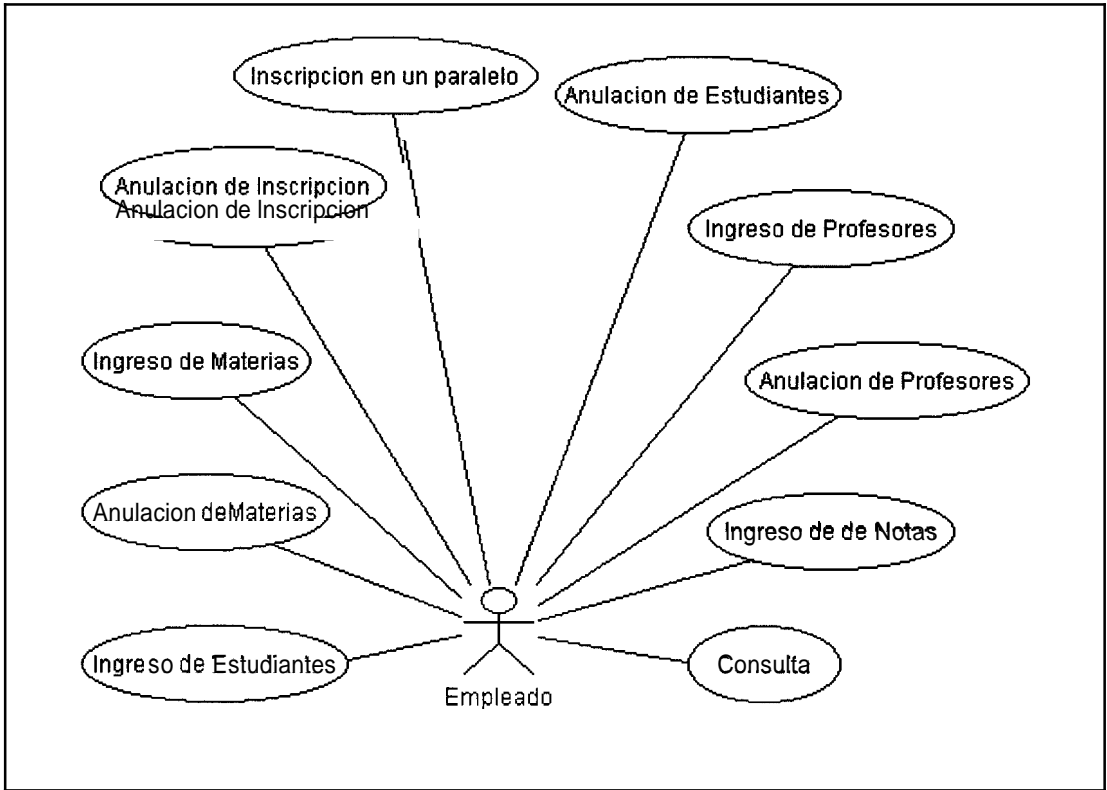
- Soportara el ingreso de informacion de profesores, datos como nombre apellido, dirección telefono, cedula; y automaticamente se asignara un codigo de matricula al estudiante.
- Ingreso de materias, la informacion requerida es el nombre de la materia y su descripción, automaticamente se asignara un codigo para cada materia.
- Ingreso de horarios, los datos del horario que sirvan para el ingreso de un paralelos, tambien se debera generar automaticamente un codigo correspondiente.
- Ingreso de paralelos, que incluye el codigo de materia, codigo del profesor, codigo del horario y cupos disponibles. En esta opcion el sistema generara un codigo secuencial para cada paralelo correspondiente a una materia determinada.
- El sistema debera permitir la inscripcion de un estudiante en un paralelo determinado, previo el chequeo de sus deudas. Si se debe registrar en mas de una materia, para cada materia nueva se debe verificar la existencia de su codigo y el cupo. Solo podrán registrarse las personas que no tengan deudas.
- Anulacion de una inscripcion, convirtiendo en disponibles los cupos respectivos.
- Anulacion de paralelos. En este caso, automaticamente tambien se cancelaran todas las inscripciones hechas para ese curso.
- Anulacion de materias, con esta opcion, automaticamente tambien se

cancelaran todas las inscripciones hechas para ese curso, y todos los paralelos definidos para esta materia.

## **4.2 Modelo de Casos de Uso**

Estos casos de uso fueron obtenidos en base al requerimiento funcional inicial:

1. Inscripcion en un Paralelo
2. Anulacion de Inscripción
3. Ingreso de Materias
4. Anulacion de Materias
5. Ingreso de Profesores
6. Anulacion de Profesores
7. Ingreso de Estudiantes
8. Anulacion de Estudiantes
9. Consulta de Registro
10. Ingreso de Notas



**Figura 12. Diagrama de Casos de Uso**

#### 4.2.1 Casos de Uso

##### ■ - Inscripcion en un paralelo

Descripción: El empleado recibe la lista de materias en que el estudiante se desea registrar, los ingresa al sistema; para que el registro se realice debe haber cupo en todas las materias requeridas, en el caso contrario se informara de las materias no disponibles

Notas: El cupo maximo de cada paralelo es de 50 estudiantes, si se

desea registrar en mas de una materia se debe verificar la existencia de su codigo y cupo. Una vez ingresadas todas las materias la transacción solo hara un solo envio final(debe haber una opción de confirmación). El número maximo de materias a registrar a un estudiante es 6. **Se** debe validar el cruce de horarios y deudas, no se valida la fecha del registro.

Actores: Empleado, Base de datos.

## **2. Anulacion de inscripcion**

Descripcion: El empleado recibe la solicitud de anulacion de inscripcion el la(las materias que el estudiante requiera, verifica la existencia del registro, la elimina y vuelve a dejar el(los) cupos disponibles.

Notas: Si se anula el unico estudiante registrado en una materia saldra el mensaje de que deberia anularse la materia ya que no hay nadie registrado.

Actores: Empleado, Base de datos.

## **3. Ingreso de Materias**

Descripcion: La unidad académica envía la solicitud de creación de materias, y el sistema, al ingresar los datos de la nueva materia generara un codigo para la materia ingresada.

Actores: Empleado, Base de datos.

#### **4. Anulacion de Materias**

Descripcion: La unidad academica envía la solicitud de anulacion de materias, el empleado ingresa el codigo de la materia y al anular la materia se anulan todas las inscripciones a los paralelos que tenga, y se dejan disponibles los horarios de los mismos.

Notas: Se asume que solo se tiene un termino académico en curso.

Actores: Empleado, Base de datos.

#### **5. Ingreso de Profesores**

Descripcion: El empleado recibe la solicitud de ingreso de profesores, ingresa los datos del profesor nuevo y el sistema valida la existencia del profesor por medio del numero de cedula, si no existe, entonces genera un nuevo codigo de profesor.

Notas: Un profesor puede pertenecer a una unidad academica particular pero puede dictar materias de otras unidades academicas.

Actores: Empleado, Base de datos.

#### **6. Anulacion de Profesores**

Descripcion: El empleado recibe la solicitud de anulacion de profesores, ingresa su codigo al sistema y el profesor queda inhabilitado para dictar clases(estado 'A'). Por tanto todas las materias en que conste como profesor quedaran anulados tambien hasta que se asigne un nuevo profesor.

Notas: Si en los paralelos que estaban asignados para el profesor

existían estudiantes registrados entonces la cantidad de materias en las que apareciera el estudiante inscrito sera el total de registros menos las que dictaba el profesor anulado.

Actores: Empleado, Base de datos.

## **7. Ingreso de Estudiantes**

Descripcion: El empleado recibe los datos del estudiante nuevo y el sistema da como resultado un numero de matricula.

Notas: Un estudiante de una determinada unidad academica podra tomar materias de otra unidad academica.

Actores: Empleado, Base de datos.

## **8. Anulacion de Estudiantes**

Descripcion: El empleado recibe la solicitud de eliminación de estudiantes y el sistema cambia su estado ('A').

Notas: Si el estudiante estaba tomando cursos, todos esos cupos se ponen vigentes. El estado del estudiante es 'A' anulado.

Actores: Empleado, Base de datos.

## **9. Consulta de Registro**

Descripcion: El empleado puede consultar las inscripciones de estudiantes, y las calificaciones en determinado paralelo.

Notas: Las consultas se podrán hacer por paralelo, profesor, horario y estudiante.

Actores: Empleado, Base de datos.

## **10. Ingreso de Notas**

Descripción: Una vez registrados los estudiantes el empleado recibirá las hojas de calificaciones de los estudiantes de un paralelo determinado, al ingresar estas calificaciones el sistema automáticamente calculará el promedio y determinará la aprobación o reprobación del curso.

Notas: El nivel académico del estudiante estará determinado por el número de materias aprobadas que tenga.

Actores: Empleado, Base de datos.

## 4.2.2 Actores

### **Actor1: Empleado**

Descripción: Es la persona encargada de interactuar directamente con el sistema y se encarga de manipular las transacciones que tiene disponibles.

### **Actor2: Base de Datos**

Descripción: Es el repositorio de datos correspondiente a cursos, profesores, estudiantes, horarios, etc.

Notas: No existe un calendario en la base de datos.

## 4.3 Requerimientos no funcionales

- El sistema deberá ser capaz de procesar una inscripción en no más de tres minutos.
- El sistema deberá estar disponible el 100% del tiempo.
- Acceso a Internet para procesar las inscripciones.



## 4.4 Modelos de análisis

### 4.4.1 Modelo de Objetos

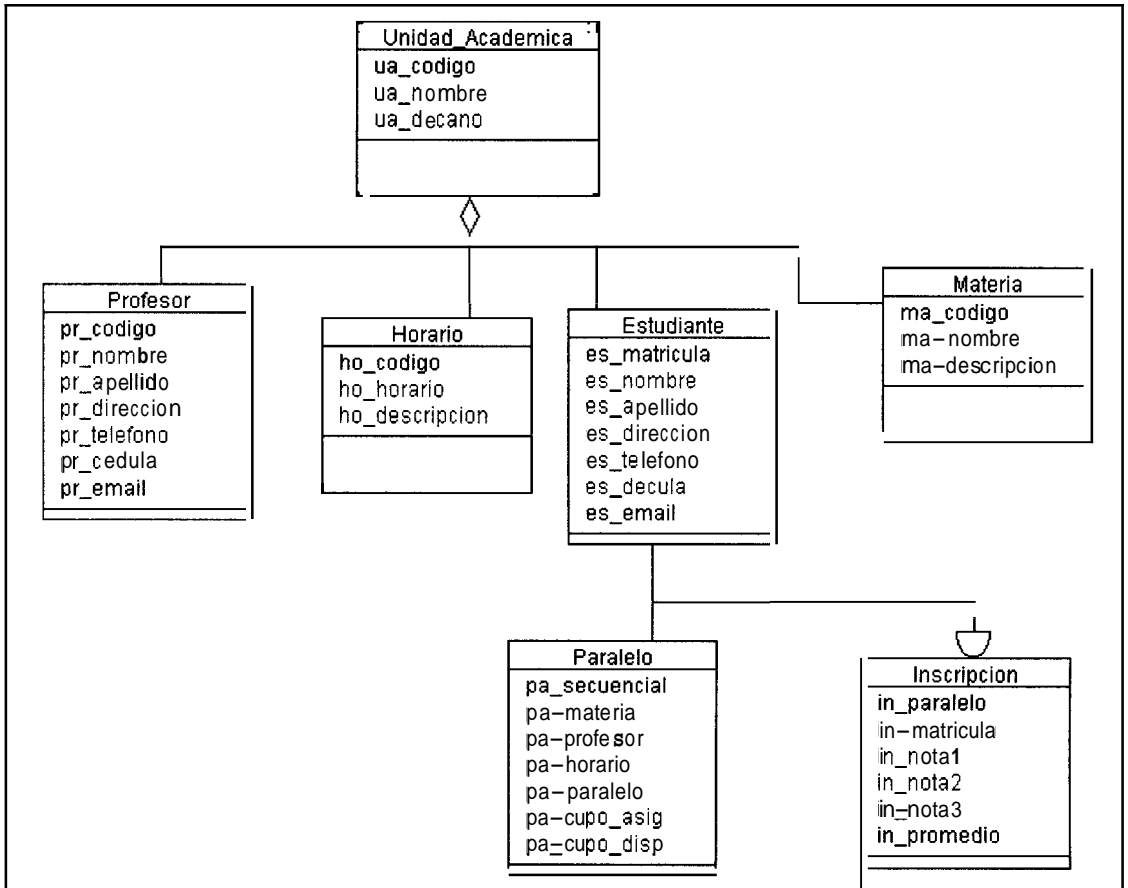


Figura No. 13 Modelo de Objetos

#### 4.4.2 Análisis de Escenarios

##### **Caso de Uso 1: Inscripción en un paralelo**

##### **Escenario 1.1: Inscripción exitosa en un paralelo**

###### Asunciones:

- El empleado llena correctamente los datos acerca de las materias en que se registrara un estudiante en forma correcta.
- Existe cupo disponible en cada materia.
- El estudiante no tiene problemas de deudas.

###### Salidas:

- Estudiante inscrito en las materias que deseo.
- Cupo de cada paralelo en que se registro el estudiante se disminuye en uno.

##### **Escenario 1.2: Inscripción errada en un paralelo**

###### Asunciones:

- El empleado llena correctamente los datos acerca de las materias en que se registrara un estudiante en forma correcta.
- No existe cupo disponible en cada materia.
- El estudiante no tiene problemas de deudas.

###### Salidas:

- Estudiante no inscrito en las materias que deseo.

- Cupo de cada paralelo en que se deseaba registrar el estudiante se mantiene.

## **Caso de Uso 2: Anulación de Inscripción**

### **Escenario 2.1: Anulación exitosa de Inscripción**

#### Asunciones:

- El empleado llena correctamente los datos acerca de las inscripciones que se anularan.
- La inscripción existe

#### Salidas:

- Se elimina la inscripción
- Se deja disponible el cupo en el paralelo respectivo.

## **Caso de Uso 3: Ingreso de Materias**

### **Escenario 3.1: Ingreso exitoso de Materias**

#### Asunciones:

- El empleado llena correctamente los datos acerca de la materia a crear.

#### Salidas:

- Genera un código para la materia creada.

**Caso de Uso 4: Anulación de Materias****Escenario 4.1: Anulación exitosa de Materias**Asunciones:

- La materia existe.
- El empleado llena correctamente los datos acerca de la materia a eliminar.

Salidas:

- Se eliminan todas las inscripciones a los paralelos de la materia eliminada
- Se anula la materia.

**Caso de Uso 5: Ingreso de Profesores****Escenario 5.1: Ingreso exitoso de Profesores**Asunciones:

- El empleado llena correctamente los datos acerca del nuevo profesor.

Salidas:

- Genera un código para la materia creada.

**Caso de Uso 6: Anulación de Profesores****Escenario 6.1: Anulación exitosa de Profesores**Asunciones:

- El profesor existe.

- El empleado llena correctamente los datos acerca del profesor a eliminar.

Salidas:

- Se eliminan todas las inscripciones a los paralelos que dictaba el profesor a eliminar.
- Se anula el profesor.

**Caso de Uso 7: Ingreso de Estudiantes**

**Escenario 7.1: Ingreso exitoso de Estudiantes**

Asunciones:

- El empleado llena correctamente los datos acerca del estudiante a ingresar.

Salidas:

- Genera un código de matricula para el estudiante.

**Caso de Uso 8: Anulación de Estudiantes**

**Escenario 8.1: Anulación exitosa de Estudiantes**

Asunciones:

- El empleado llena correctamente los datos acerca del estudiante a eliminar.

Salidas:

- Los cupos de los cursos que estaba tomando el estudiante quedan disponibles

- Se elimina el estudiante.

### **Caso de Uso 9: Consulta de Registro**

#### **Escenario 9.1: Consulta exitosa de Registro**

##### Asunciones:

- El empleado llena correctamente los datos acerca del estudiante a consultar.

##### Salidas:

- Los cursos que esta tomando el estudiante y sus calificaciones

### **Caso de Uso 10: Ingreso de Notas**

#### **Escenario 10.1: Ingreso exitoso de Notas**

##### Asunciones:

- El estudiante existe.
- El estudiante esta registrado en por lo menos una materia.

##### Salidas:

- Se actualizan las calificaciones.
- Se calcula el promedio de calificaciones.

### 4.4.3 Diagramas de Interacción de Objetos

#### Escenario 1.1 Inscripción de un estudiante

##### Asunciones:

- El estudiante llena la forma de ingreso correctamente.
- Existe cupo disponible en el paralelo.
- No hay problemas de deudas.

##### Resultados:

- Estudiante inscrito .
- Cupo del paralelo se disminuye en uno.

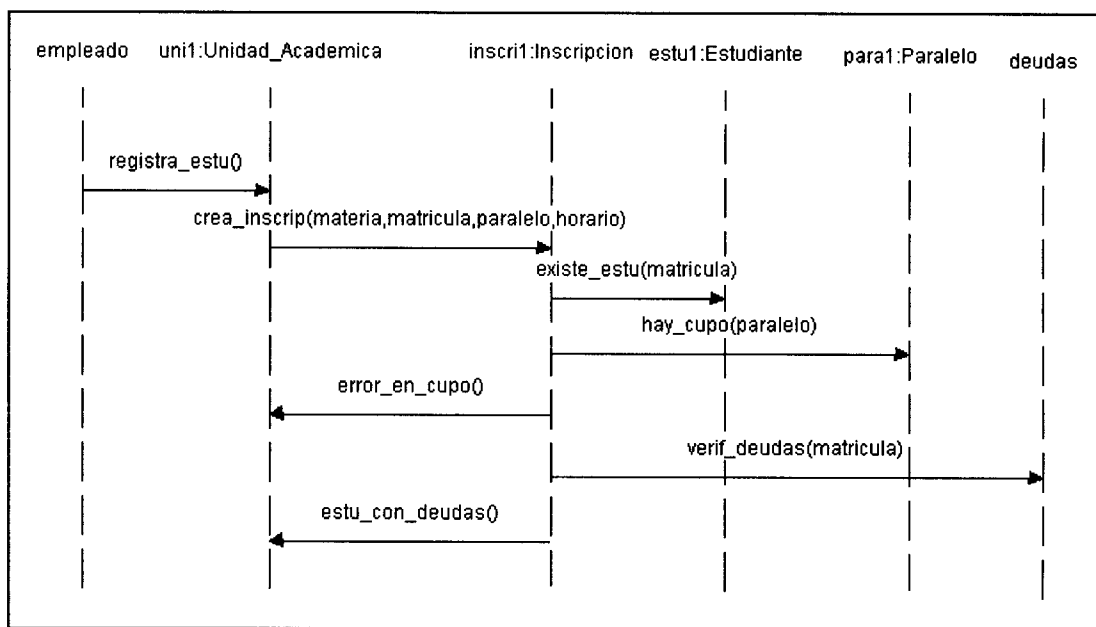


Figura No. 14 Diagrama de Interacción de Objetos 1.1

## Escenario 2.1 Anulación de inscripción

### Asunciones:

- El estudiante llena la forma de anulación correctamente.
- La inscripción existe



### Resultados:

- Se anula la inscripción.
- Cupo del paralelo aumenta en uno.

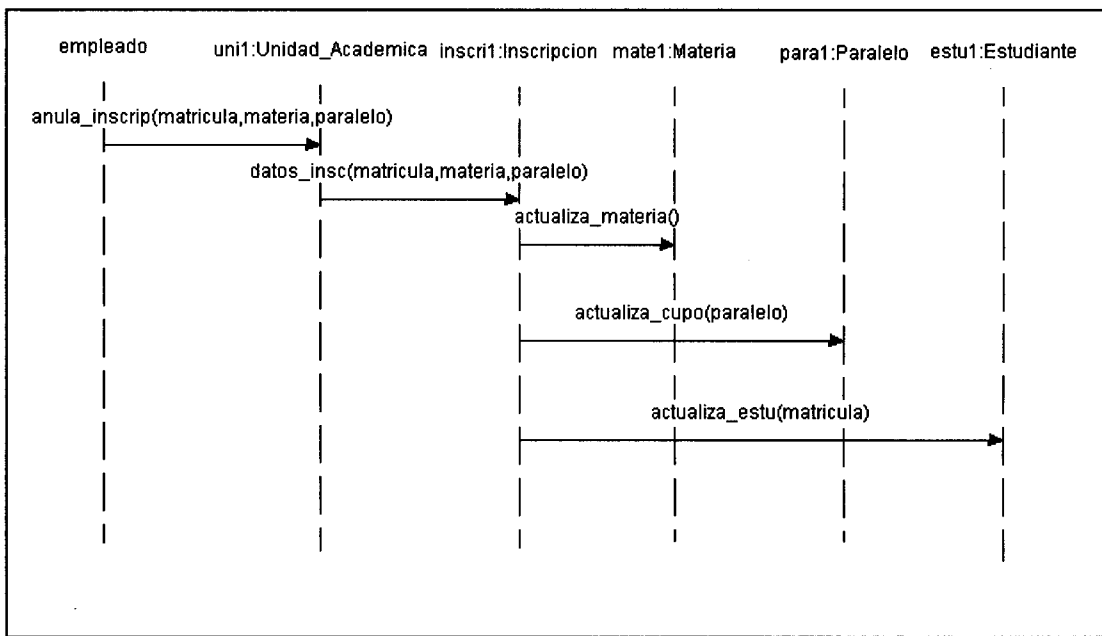


Figura No. 15 Diagrama de Interacción de Objetos 2.1



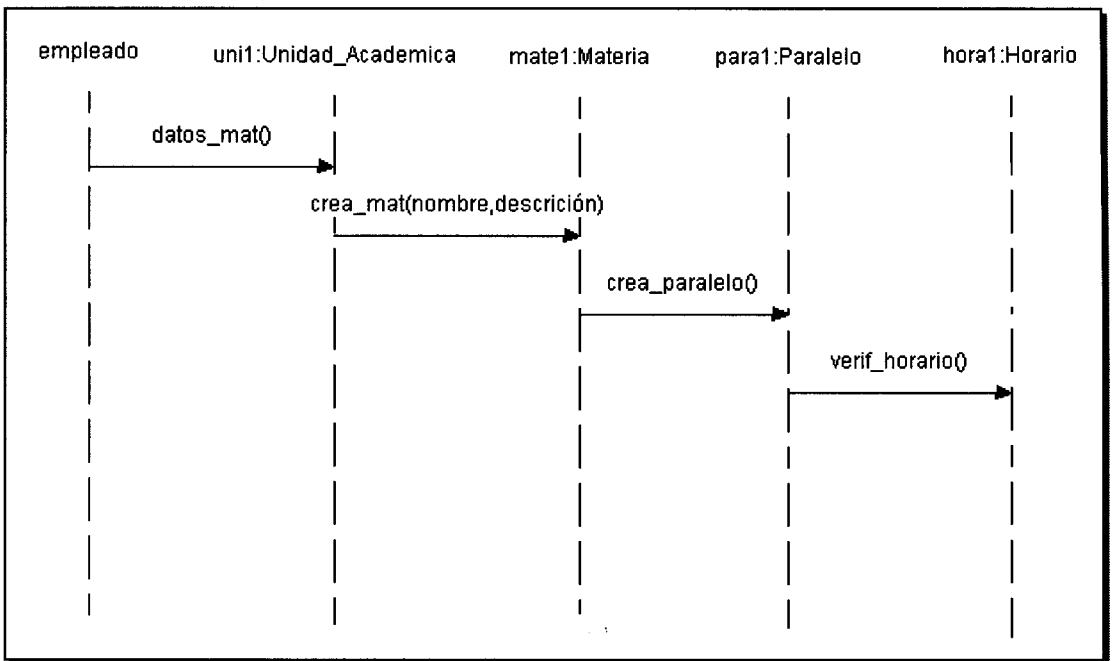
### Escenario 3.1 Ingreso de Materias

Asunciones:

- El empleado llena la forma de creación correctamente.
- La materia no existe

Resultados:

- Se crea un nuevo código de materia.



**Figura No. 16 Diagrama de Interacción de Objetos 3.1**

### Escenario 4.1 Anulación de Materias

Asunciones:

- La materia existe.

Resultados:

- Se anulan todas las inscripciones a los paralelos de la materia eliminada.
- Se actualizan datos de inscripciones de estudiantes.

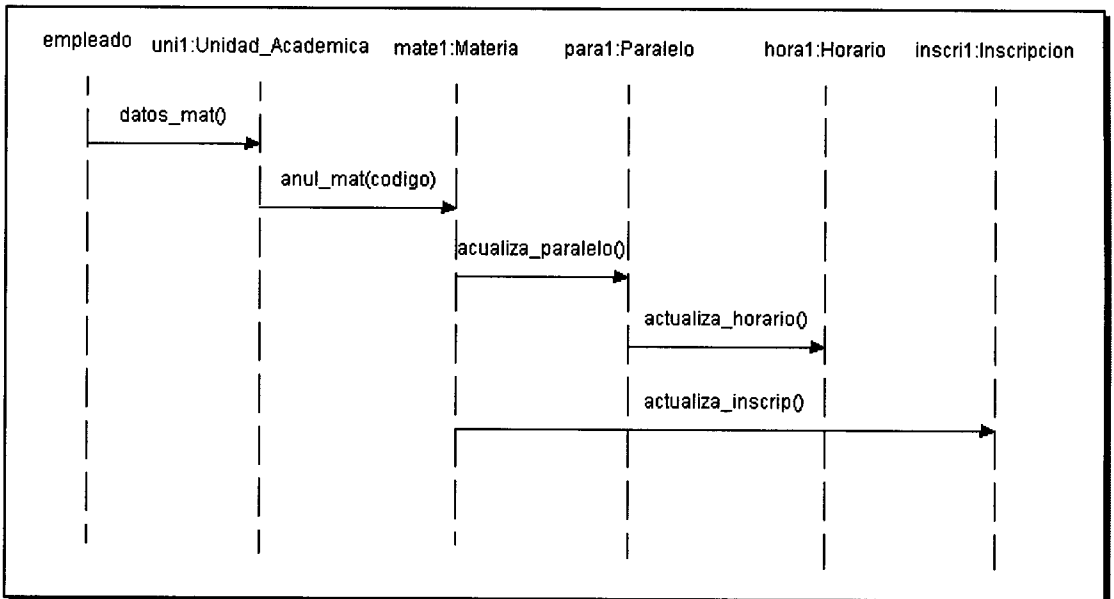


Figura No. 17 Diagrama de Interacción de Objetos 4.1



### Escenario 10.1 Ingreso de Notas

Asunciones:

- La inscripción en el paralelo existe

Resultados:

- Se actualiza la inscripción.

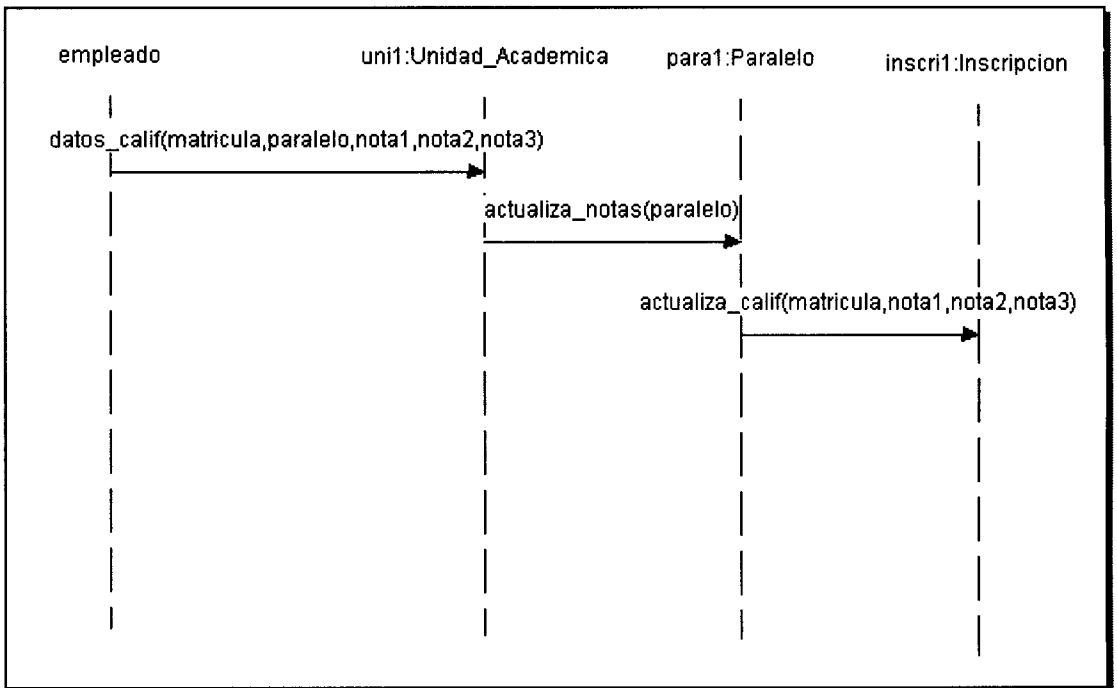


Figura No. 18 Diagrama de Interacción de Objetos 4.1

#### 4.4.4 Descripción de Clases

##### Clase : Materia

###### Atributos

- materia
- Nombre
- descripción

###### Métodos

- String que\_materia()
- String bs\_materia()
- String bus\_materia()
- Boolean eli\_materia()
- Boolean ing\_materia()

##### Clase : Deuda

###### Atributos

- matricula
- valor

###### Métodos

- String bus\_deudas()
- Boolean eli\_deudas()

### **Clase : Estudiante**

#### Atributos

- apellido
- cédula
- dirección
- email
- matricula
- nombre
- teléfono

#### Métodos

- String bus\_estudiante()
- String que\_estudiante()
- Boolean ing\_estudiante()
- Boolean eli\_estudiante()

### **Clase : Horario**

#### Atributos

- horario
- nombre
- descripción

#### Métodos

- String que\_horario(int)

- String bs\_ horario ()
- String bus\_ horario ()
- Boolean eli\_ horario ()
- Boolean ing\_ horario ()

### **Clase : Profesor**

#### Atributos

- apellido
- cédula
- dirección
- email
- matricula
- nombre
- teléfono

#### Métodos

- String bus\_profesor()
- String bs\_profesor()
- String que\_profesor()
- Boolean ing\_profesor()
- Boolean eli\_profesor()

## **Clase : Inscripción**

### Atributos

- cod\_paralelo
- materia
- matricula
- nota\_1
- nota\_2
- nota\_3
- paralelo
- promedio

### Métodos

- String bs\_inscripción()
- String bus\_inscripcion()
- String que\_inscripcion()
- boolean ing\_inscripcion()
- boolean eli\_inscripcion()
- boolean ing\_notas()

## **Clase : Paralelo**

### Atributos

- cod\_paralelo

- cupo\_asig
- cupo\_disp
- horario
- materia
- paralelo
- profesor

#### Métodos

- String bs\_paralelo()
- String bus\_paralelo()
- boolean eli\_paralelo()
- boolean ing\_paralelo()
- int que\_cod\_paralelo()



## 4.5 Modelo de Interfaces de Usuario

### 4.5.1 Flujo de Ventanas

La figura muestra la interacción de las ventanas en el sistema, cada ventana puede ser accesada desde cualquier otra ventana del sistema.

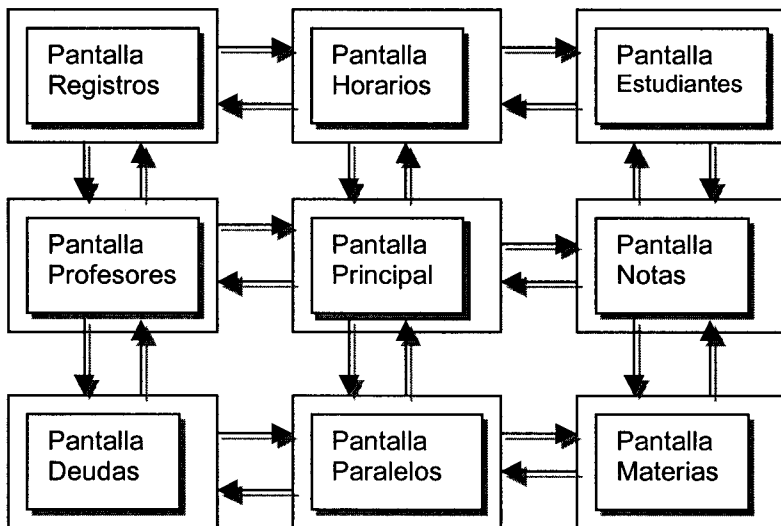


Figura No. 19 Flujo de Ventanas

## 4.5.2 Layouts

Pantalla Principal:

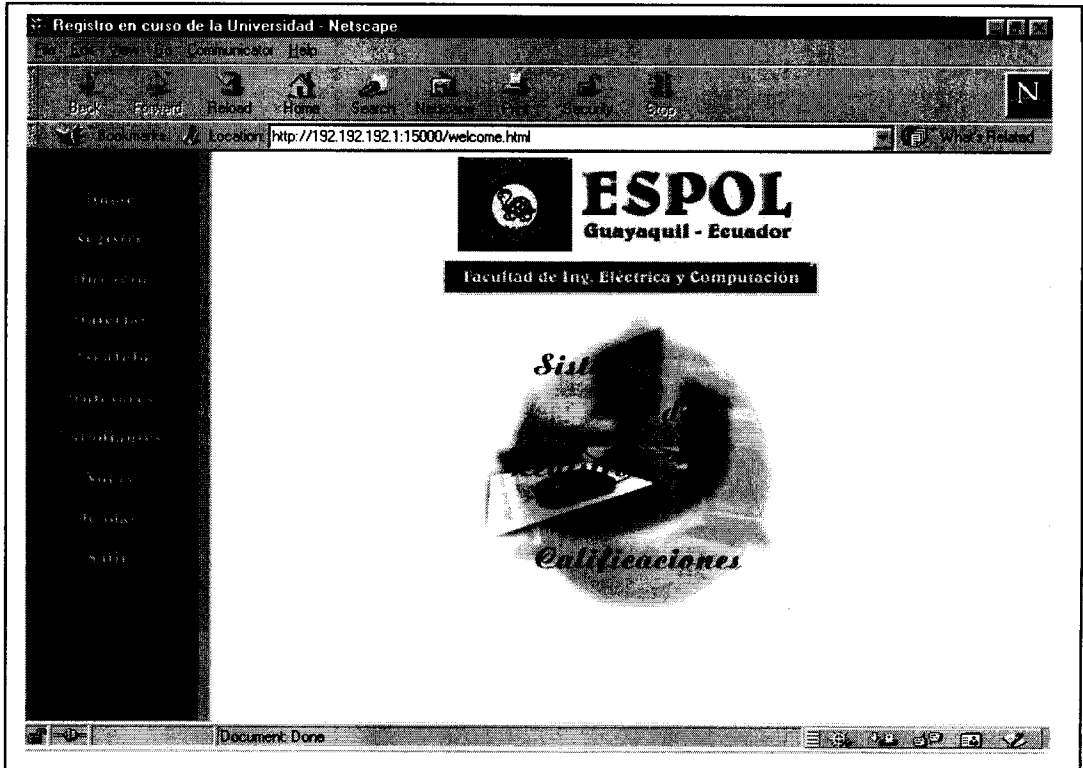
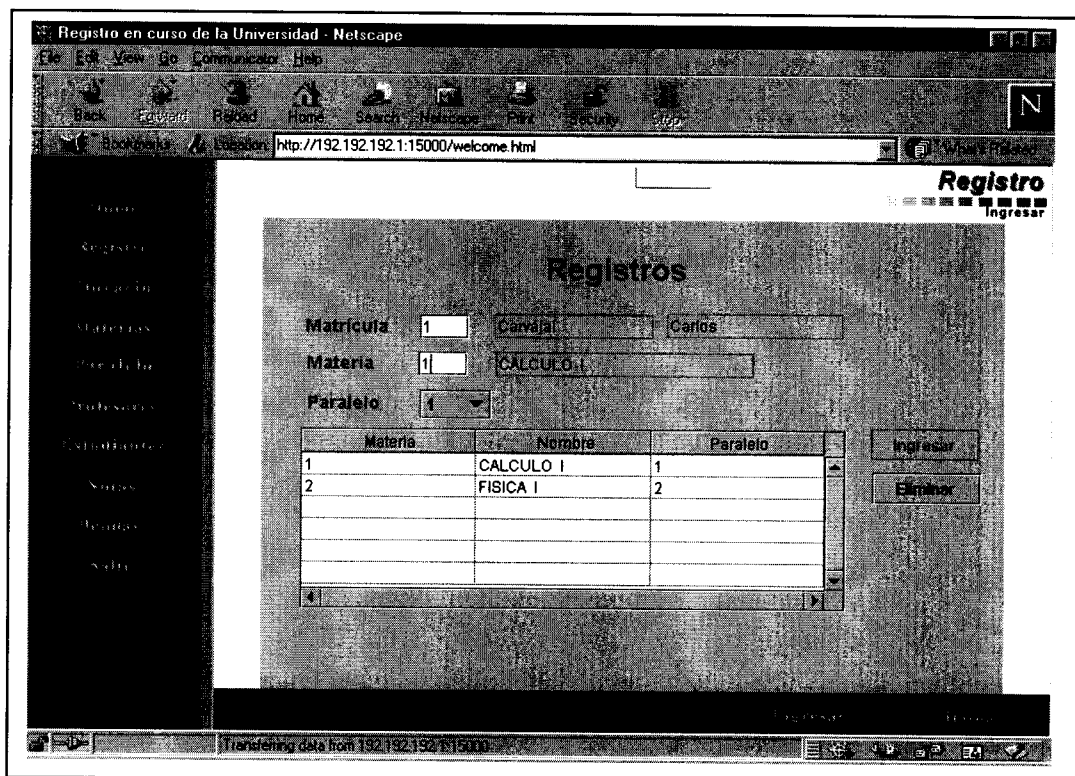


Figura No.20 Pantalla Principal

Esta pantalla muestra el menú inicial, que permite al usuario la utilización de cada una de las transacciones del sistema.

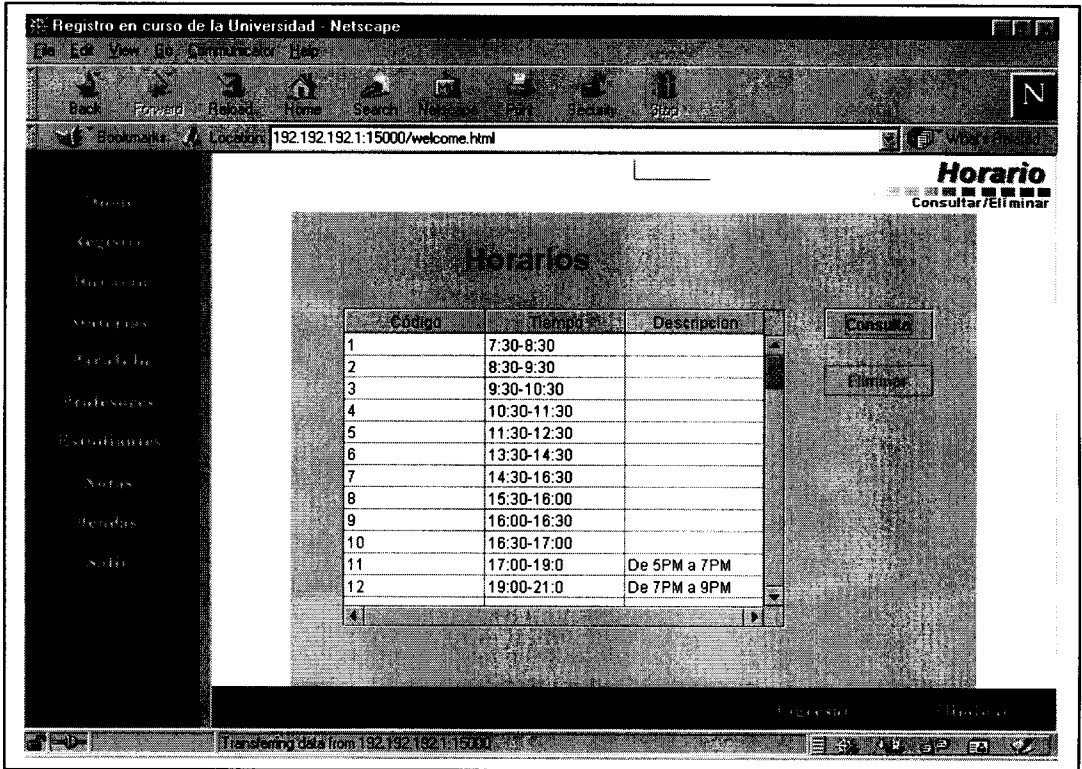
## Pantalla Registros



**Figura No. 21 Pantalla Registros**

Esta pantalla permite el ingreso de las inscripciones a los paralelos que el estudiante requiera. Los botones "Ingresar" y "Eliminar" se habilitan y deshabilitan según sea el caso.

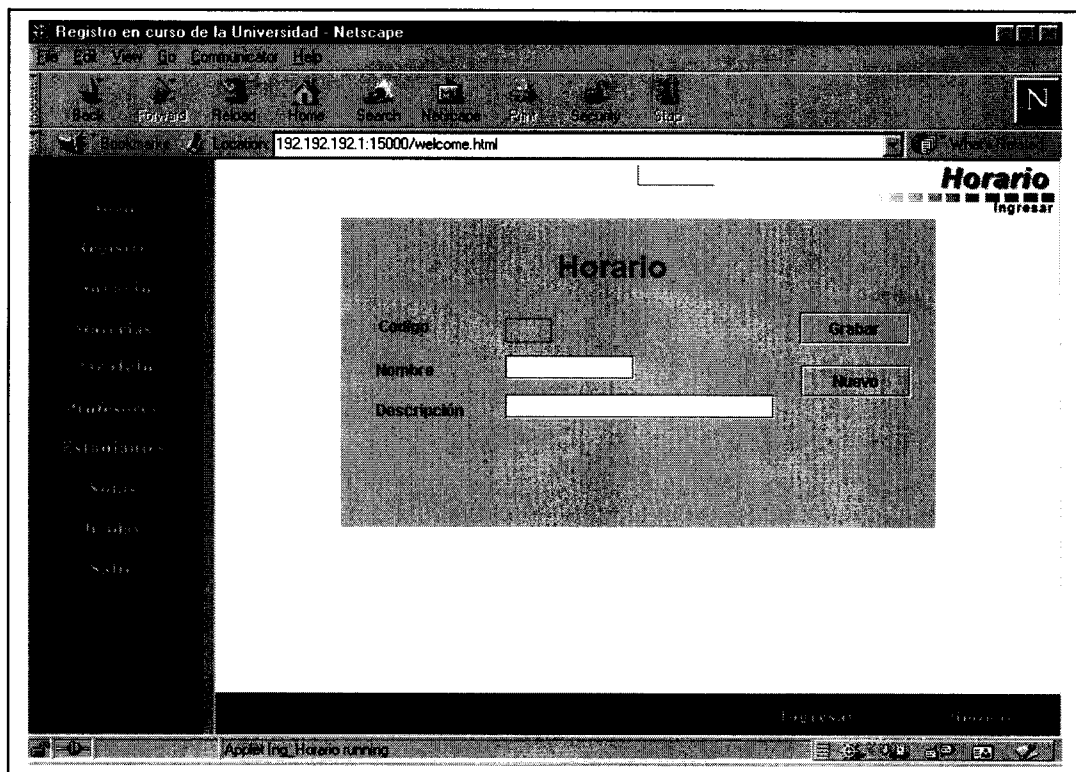
## Pantalla Horarios



**Figura No.22 Pantalla Horarios**

Esta pantalla permite la consulta y eliminación de horarios.

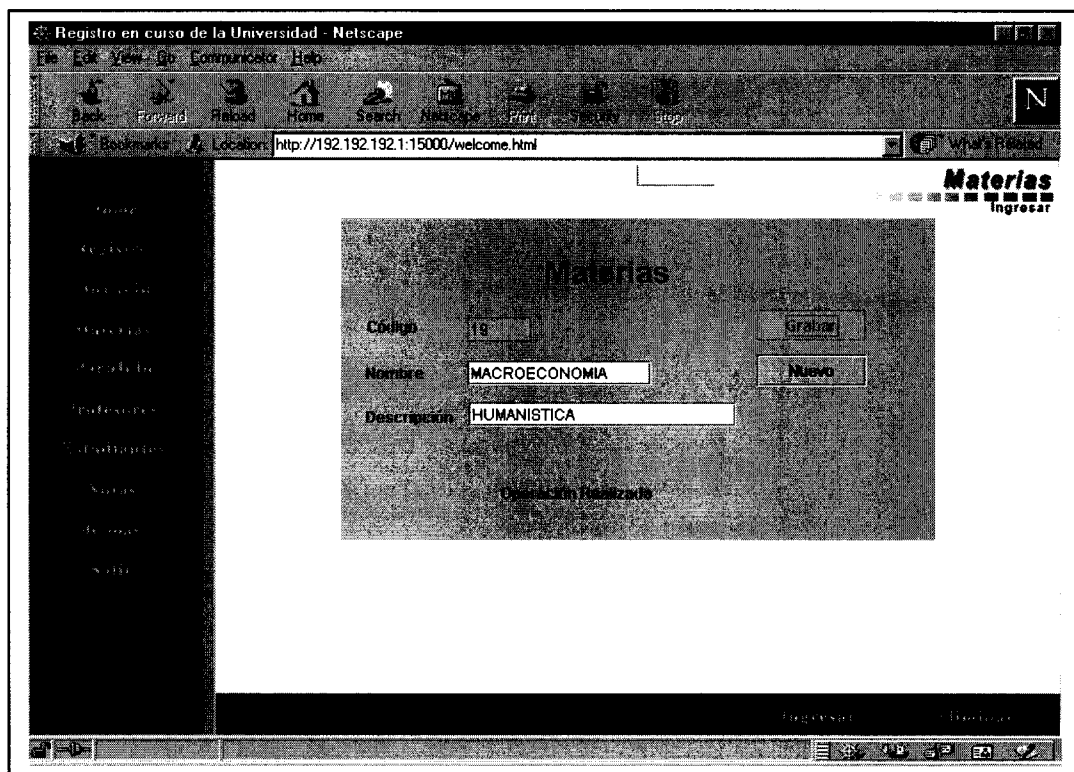
## Pantalla Horario



**Figura No.23 Pantalla Horario**

Esta pantalla permite el ingreso de un nuevo horario en el que se dictaran los diferentes cursos ofrecidos en la universidad.

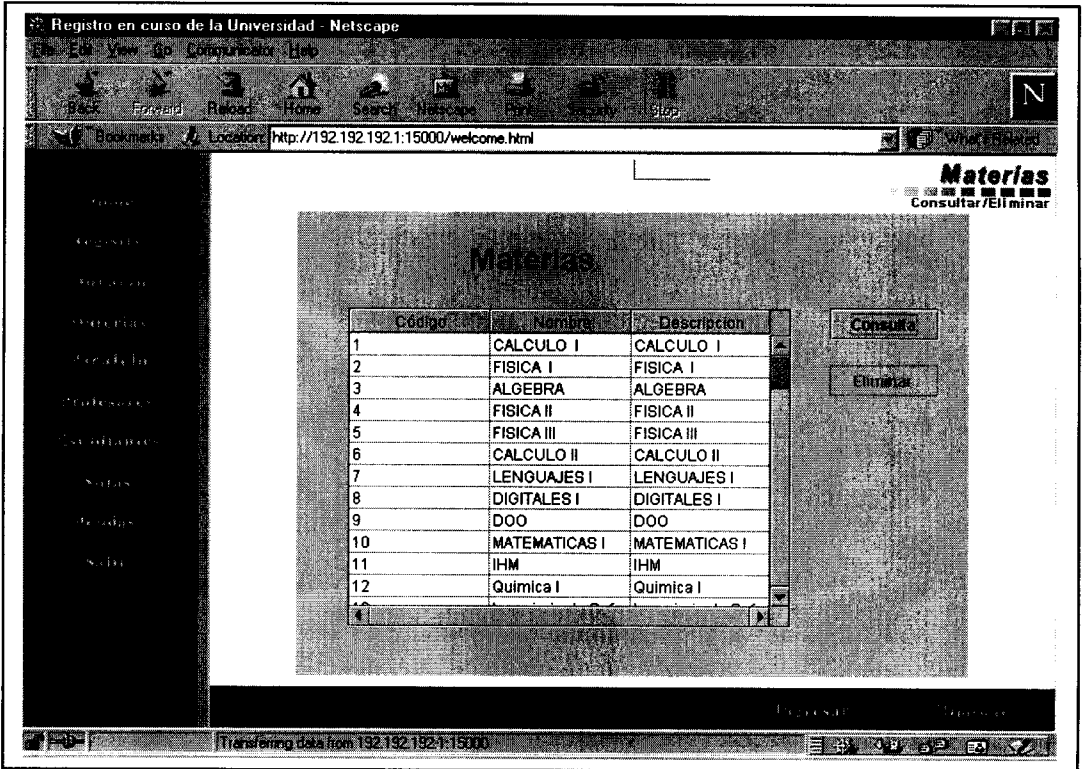
## Pantalla Materias (Ingreso)



**Figura No.24 Pantalla Materias**

Esta pantalla permite la creación de una nueva materia, asignando el sistema un código automático para la materia recién creada.

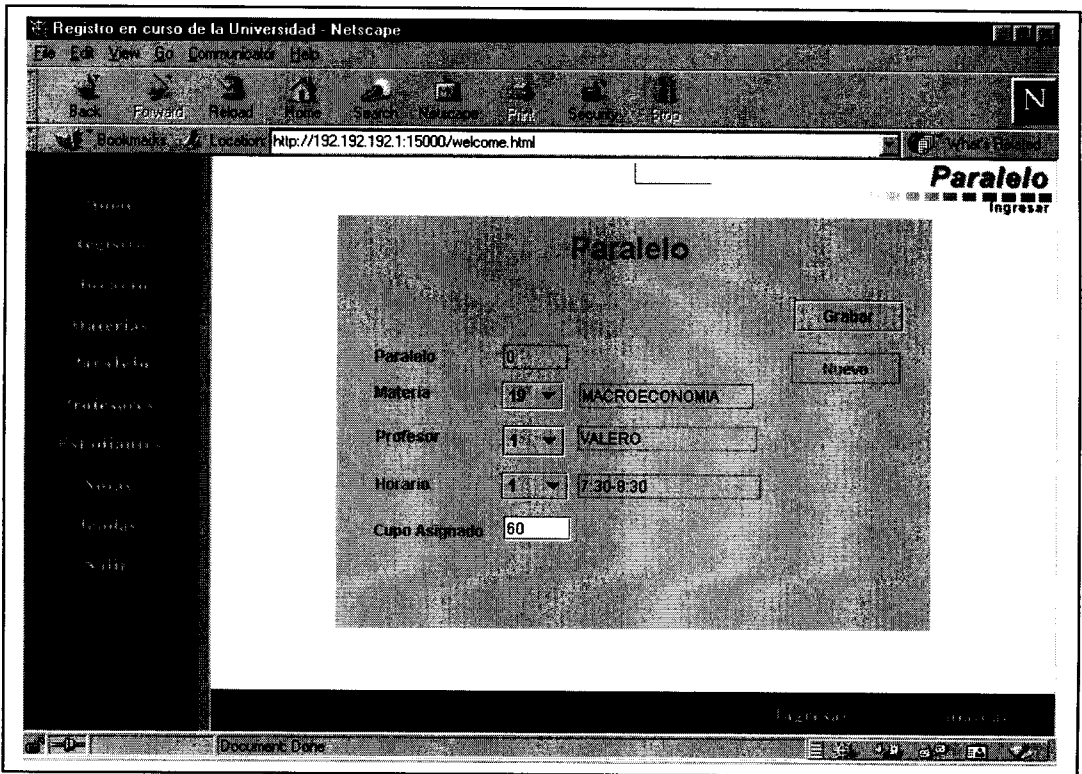
## Pantalla Materias



**Figura No.25 Pantalla Materias**

Esta pantalla permite la consulta y eliminación de materias.

## Pantalla Paralelo



**Figura No.26 Pantalla Paralelo**

Esta pantalla permite el ingreso de cada uno de los paralelos correspondientes a una materia determinada, además del profesor que dictara el curso.



## Pantalla Paralelos

The screenshot shows a Netscape browser window with the following details:

- Browser Title: Registro en curso de la Universidad - Netscape
- Address Bar: http://192.192.192.1:15000/welcome.html
- Page Title: Paralelos
- Table Data:

Código	Paralelo	Materia	Profesor	Horario	Asignado	Disponible
1	1	CALCUL...	VALERO ...	7:30-8:30	60	57
2	1	FISICA I	CAICEDO ...	8:30-9:30	60	59
3	1	ALGEBRA	MONSAL...	9:30-10:30	60	60
4	1	FISICA II	LUDMILA ...	10:30-11:...	60	60
5	1	FISICA III	Jordan C...	11:30-12:...	60	60
6	1	CALCUL...	Pelaez E...	13:30-14:...	60	60
7	1	LENGUAJ...	VERA DU...	15:30-16:...	60	60
8	2	CALCUL...	VERA DU...	15:30-16:...	60	60
9	3	CALCUL...	Pelaez E...	13:30-14:...	60	59
10	2	FISICA I	Pelaez E...	9:30-10:30	60	59
11	3	FISICA I	LUDMILA ...	7:30-8:30	60	60
12	4	DIGITALE	VERA DU...	11:30-12:...	60	60

Buttons: Consulta, Eliminar

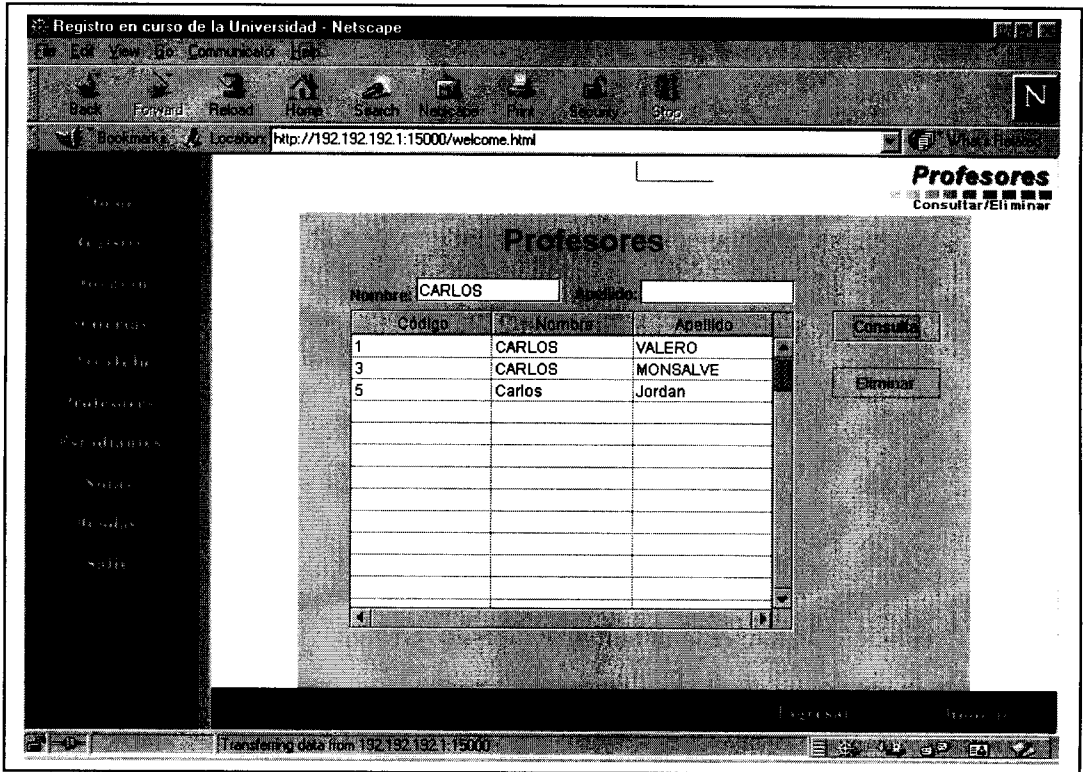
Status Bar: Transferring data from 192.192.192.1:15000

**Figura No.27 Pantalla Paralelos**

Esta pantalla permite la consulta y eliminación de un paralelo de alguna materia.



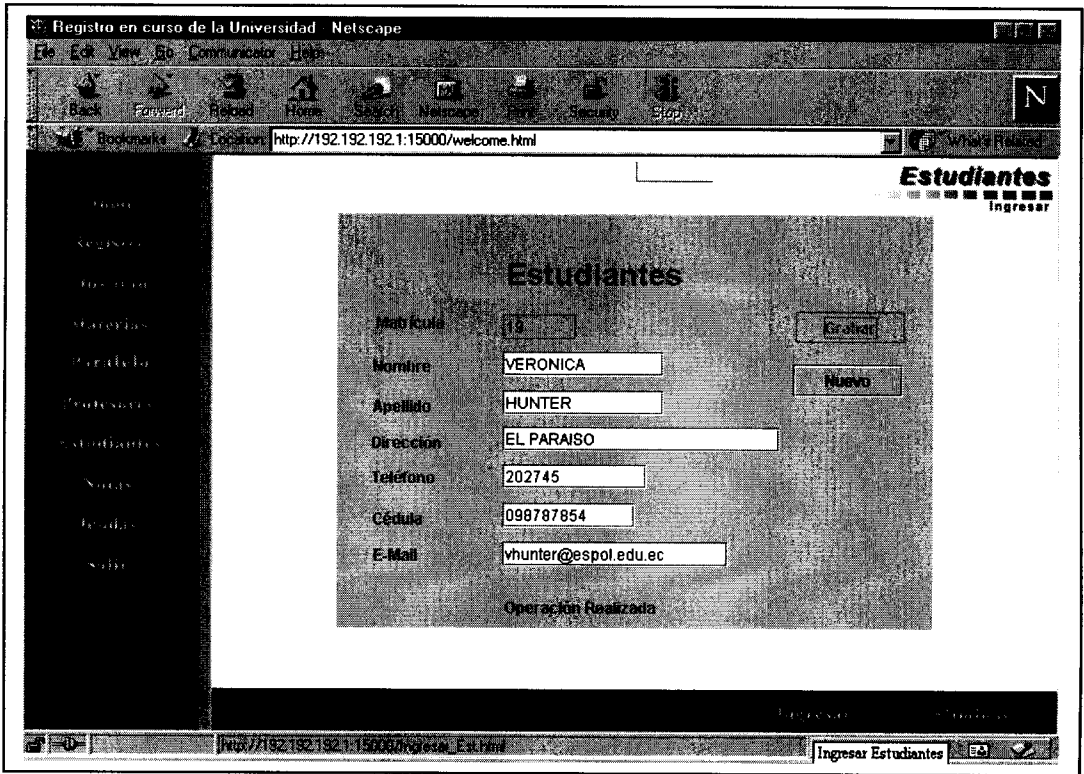
## Pantalla Profesores



**Figura No.29 Pantalla Profesores**

Esta pantalla permite la consulta y eliminación de profesores.

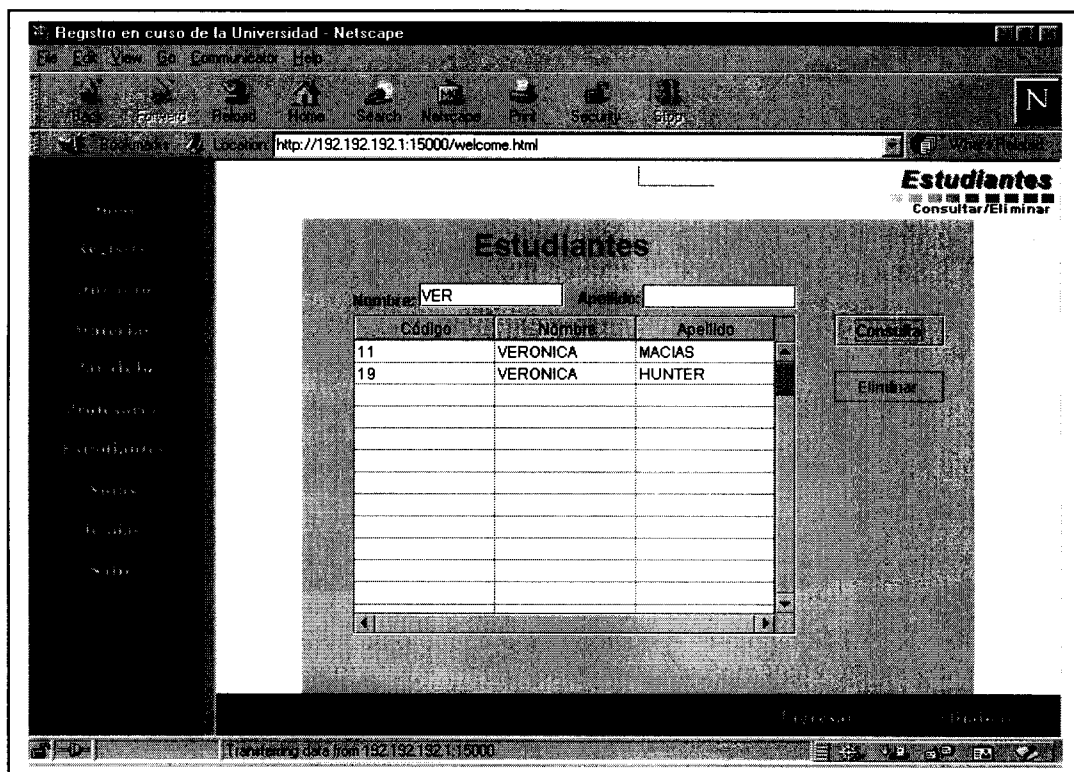
## Pantalla Estudiantes (Ingreso)



**Figura No.30 Pantalla Estudiantes(Ingreso)**

Esta pantalla permite hacer el ingreso de un nuevo estudiante, además genera automáticamente su número de matricula.

## Pantalla Estudiantes



**Figura No.31 Pantalla Estudiantes(Consulta/Eliminación)**

Esta pantalla permite la consulta y eliminación de los estudiantes.

## Pantalla Notas (Ingreso)

Registro en curso de la Universidad - Netscape

192.192.192.1:15000/welcome.html

**Notas Ingresar**

Materia: 1 CALCULO I

Parcial: 1

Parcial 1: 100 Parcial 2: 80 Mejoramiento: 50

Ma.	Nombre	Parcial 1	Parcial 2	Mejorami...	Promedio
1	Carvajal Carlos	100	80	50	76
8	Zambrano Nubia	82	60	100	80

Actualizar

Starting Java... done

**Figura No.32 Pantalla Notas(Ingreso)**

Esta pantalla permite el ingreso/actualización de las calificaciones de un estudiante.

## Pantalla Notas (Consulta)

The screenshot shows a Netscape browser window titled "Registro en curso de la Universidad - Netscape". The address bar displays "192.192.192.1:15000/welcome.html". The main content area is titled "Notas" and contains a form with "Matrícula" (2) and "Nombre" (Ricardo Alvarez). Below the form is a table of grades:

Código	Materia	Nota 1	Nota 2	Nota 3	Promedio
1	CALCUL...	100	60	60	73
2	FISICA I	100	100	100	100

**Figura No.33 Pantalla Notas(Consulta)**

Esta pantalla permite consultar las notas de un estudiante en todas las materias que esta registrado.

## Pantalla Deudas

Registro en curso de la Universidad - Netscape

Location: 192.192.192.1:15000/welcome.html

**Deudas**  
Consultar/Eliminar

### Estudiantes Deudores

Código	Apellido	Nombre	Monto
13	NARVAEZ	MARIUXI	1000

Consulta  
Eliminar

Transferring data from 192.192.192.1:15000

Figura No.34 Pantalla Deudas

Esta pantalla permite consultar las deudas que tienen los estudiantes.



## 4.6 Modelos de Diseño

### 4.6.1 Arquitectura del sistema

La aplicación esta basada en la Arquitectura Cliente/Servidor, para operar en una Red de Area Local y en el Internet.

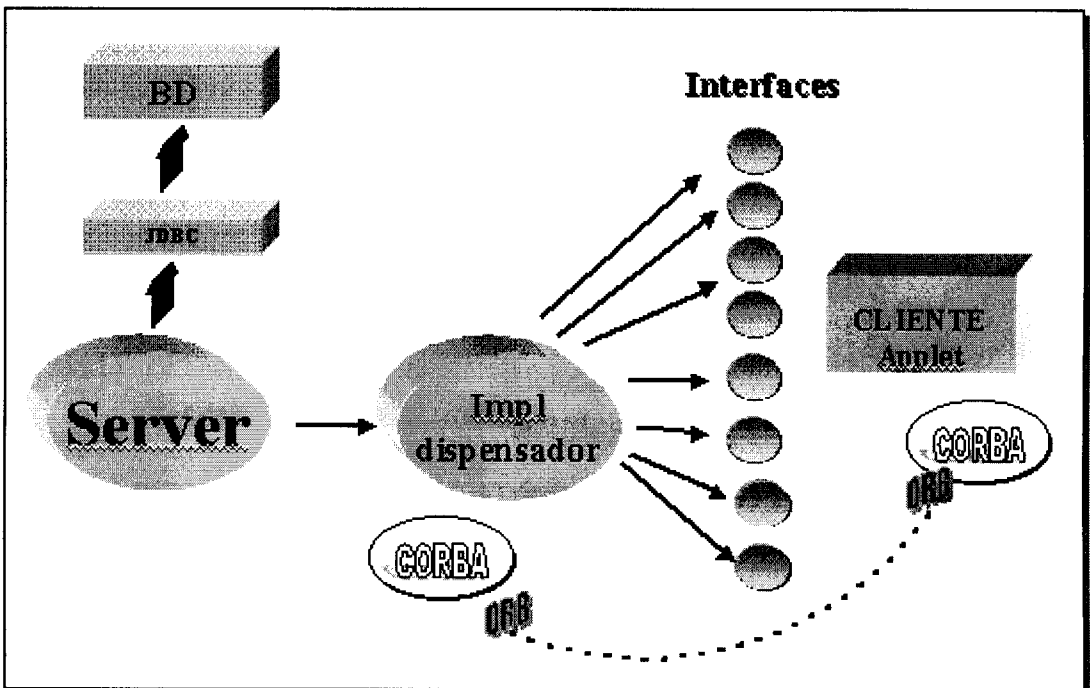


Figura No.35 Arquitectura del sistema

La arquitectura es de tres capas:

La primera capa es el cliente (browser). Este muestra las páginas HTML enviadas por el Web Server, además también envía la información que se

ingresa el usuario al Web Server para procesar las transacciones a través de los applets. Al momento de conectarse a la pagina principal del sitio web, el cliente se encarga de obtener del servidor todos los stubs necesarios para realizar las llamadas a los objetos servidores.

La segunda capa es el Web Server y el Proceso servidor, encargado de levantar las interfaces de los objetos servidores y el dispensador de objetos. El Web Server envía las páginas web con contenido estático o dinámico al cliente. El cliente se comunica con el Web Server por medio del protocolo estándar conocido como HTTP, mientras que los objetos CORBA se comunican usando el protocolo IIOP.

La tercera capa muestra el método de persistencia utilizado, contiene la base de datos y la capacidad transaccional. Los recursos de esta capa son accesados a través de la capa intermedia utilizando JDBC.

## **4.7 Implementación**

### **4.7.1 Ambiente de Desarrollo**

Para la implementación de este sistema fue necesario utilizar las siguientes herramientas:

- Visual age for Java v2.0
- Visibroker for Java v3.4
- JDK v1.1.7
- Browser que soporte Applets de Java y
- Applets Java

#### **4.7.1.1 Visual Age**

Provee un ambiente de desarrollo orientada a objeto, siendo además una poderosa herramienta de programación visual. Genera código puramente Java y posee facilidades para el desarrollo de aplicaciones usando CORBA. Las herramientas de programación permiten definir el esquema visual de los applets o aplicaciones con la selección de los componentes desde una paleta de controles Java.

#### **4.7.1.2 Visibroker for Java**

Soporta métodos de invocaciones CORBA. Los métodos en servidor pueden ser invocados por una aplicación Cliente Java o por applets dentro de un browser. El compilador IDL Visibroker genera el código skeleton para los objetos servidores Java; esto también genera los stubs Java por el lado del Cliente.

El Visibroker para Java soporta métodos de invocaciones Corba tanto estáticos como dinámicos. implementa completamente el protocolo IIOP.

El Visibroker para Java viene con un servicio de nombramiento de objeto llamado OSAgent. Múltiples OSAgents corriendo en la misma red se localizan unos a otros y automáticamente particionan el namespace entre ellos. El ORB soporta tanto funciones servidoras como clientes, lo cual significa que algún cliente puede implementar callbacks.

#### **4.7.1.3 Java Development Kit (JDK)**

Java 1.1.7 es la implementación de JavaSoft, para hacer que el lenguaje Java trabaje más eficientemente y seguro en ambientes distribuidos.

Sus principales características que lo diferencian son: Mejoras al lenguaje, especialmente en la parte del SWING (Interface gráfica); los archivos JAR, que combinan archivos de los applets en uno solo comprimido; los API de seguridad y, el JDBC, que es el driver de conexión a bases de Datos.

#### 4.7.1.4 SQL

Es la herramienta encargada de organizar, administrar y recuperar datos almacenados en una base de datos relacionales. El nombre SQL (Structured Query Language: Lenguaje estructurado de consulta), es un lenguaje que se utiliza para interactuar con una base de datos. La base de datos utilizada es MsSQLServer 7.0 soporta el uso de stored procedures, que es lo que se utilizó para implementar la lógica del negocio.



#### 4.7.2 Código fuente

##### Procesos Servidores

##### Nomenclatura:

Los métodos con prefijo “que” devuelven una descripción de la entidad en función de un código, el prefijo “bs” nos devuelve una serie de códigos de acuerdo a lo requerido, el prefijo “bus” brinda la información requerida en las consultas, y finalmente “ing” y “eli” procesan los ingresos y eliminaciones.

##### Paquete Servidor

En este paquete se implementa el código que trabajará con la base de datos mediante llamadas a Stored Procedures y las respuestas obtenidas las convertirá en el formato requerido por los procesos clientes.

##### Paquete : Implementation

En estas clases está implementado el código de las interfaces, métodos y

atributos definidos en el archivo idl, y básicamente su tarea consiste en invocar a sus clases correspondientes en el paquete SERVIDOR y pasar los parámetros necesarios a los atributos de los objetos servidores. Proveen la codificación requerida para CORBA.

Clase : Impl\_materia

Clase : Impl\_horario

Clase : Impl\_deuda

Clase : Impl\_estudiante

Clase : Impl\_profesor

Clase : Impl\_inscripcion

Clase : Impl\_paralelo

En esta misma capa encontramos la clase Impl\_dispensador que es instanciada una vez por el proceso servidor principal (Server.java), el dispensador se encarga de atender las llamadas de todos los clientes, administra una serie de objetos servidores (Impl\*) que los asignará a los clientes en demanda.

```

//Registro.idl
module Registro
{
    interface re_horario
    {
        boolean ing_horario(in long horario, in string nombre, in string
descripción);
        boolean eli_horario(in long horario);
        string bs_horario();
        string bus_horario();
        string que_horario(in long horario);
        attribute long horario;
    };

    interface re_paralelo
    {
        boolean ing_paralelo(in long paralelo,in long materia, in long
profesor, in long horario,
            in long cupo_asig, in long cupo_disp);
        boolean eli_paralelo(in long cod_paralelo);
        string bs_paralelo(in long materia);
        string bus_paralelo();
        long que_cod_paralelo(in long materia,in long paralelo);
        attribute long cod_paralelo;
        attribute long paralelo;
    };

    interface re_materia
    {
        boolean ing_materia(in long materia, in string nombre, in string
descripción);
        boolean eli_materia(in long materia);
        string bs_materia();
        string bus_materia();
        string que_materia(in long materia);
        attribute long materia;
    };

    interface re_estudiante
    {
        boolean ing_estudiante(in long matricula,in string nombre, in
string apellido,
            in string dirección, in string teléfono, in string cédula,
            in string email);
        boolean eli_estudiante(in long matricula);
    };
}

```



```

    string bus_estudiante(in string apellido,in string nombre);
    string que_estudiante(in long matricula);
    attribute long estudiante;
};

interface re_inscripcion
{
    boolean ing_inscripcion(in long cod_paralelo, in long matricula);
    boolean eli_inscripcion(in long cod_paralelo, in long matricula);
    boolean ing_notas(in long matricula,in long materia,in long
nota_1,in long nota_2,
                    in long nota_3);
    string bus_notas(in long matricula);
    string bs_inscripcion(in long matricula);
    string bus_inscripcion(in long materia, in long paralelo);
    boolean que_inscripcion(in long matricula,in long materia);
};

interface re_profesor
{
    boolean ing_profesor(in long profesor, in string nombre, in string
apellido,
                    in string dirección, in string teléfono, in string
cédula,
                    in string email);
    boolean eli_profesor(in long profesor);
    string bs_profesor();
    string bus_profesor(in string apellido,in string nombre);
    string que_profesor(in long profesor);
    attribute long profesor;
};

interface re_deuda
{
    string bus_deuda();
    boolean eli_deuda(in long matricula);
};

```



```

interface re_interface
{
    attribute                long                índice;
    re_materia                rsv_materia();
    re_estudiante            rsv_estudiante();
    re_profesor              rsv_profesor();
    re_paralelo              rsv_paralelo();
    re_inscripcion          rsv_inscripcion();
    re_horario               rsv_horario();
    re_deuda                 rsv_deuda();
};

interface re_dispensador
{
    re_interface             rsv_interface();
    void                     libe_interface(in re_interface pv_var);
};
};

```

## **CONCLUSIONES y RECOMENDACIONES**

1. El esquema de seguridades de CORBA hace posible la realización de transacciones seguras a través de Internet.
2. Esta tecnología hace transparente al desarrollador la distinción entre cliente y servidores ya que los clientes también pueden ser servidores.
3. Debido a que no existe en nuestro medio proveedores de esta tecnología (CORBA), el desarrollo de aplicaciones de este tipo podría tener un índice de error relativamente alto.
4. Para ejecutar es necesario cumplir con los requerimientos mínimos de hardware y software, que son importantes en términos de consumo de recursos.
5. El producto se tiene amplias perspectivas a futuro.

## **BIBLIOGRAFÍA**

1. ORFALI R., HARKEY D. & EDWARDS J., Essential Client / Server Survival Guide (2da. Edicion; New York: John Wiley & Sons. Inc, 1996).
2. Developing Object-Oriented Software, An Experience-Based Approach (IBM Object Oriented Technology Center, Prentice Hall, 1997).