Universidad de Los Andes
Facultad de Ingeniería
Escuela de Ingeniería de Sistemas

Diseño de una plataforma de integración para el acceso a bases de datos desde NeuronMaster

Tesis presentada ante la ilustre Universidad de Los Andes como requisito final para optar al título de Ingeniero de Sistemas

Tutor:
Prof. Rafael Rivas

César Enrique Bravo Bravo
C.I. 12.797.663

Mérida, Abril 2000

Dedicatoria

A mis Padres, Víctor y Glady,
A Luz María,
A Víctor Rafael,
Y a Víctor Manuel y Angela Gabriela.

Universidad de Los Andes

Facultad de Ingeniería

Escuela de Ingeniería de Sistemas

Resúmen

Neuron Master es un herramienta para el desarrollo de aplicaciones basadas en redes neuronales que ha venido siendo desarrollado desde 1997 en el Postgrado de Control de Procesos de la Universidad de los Andes. Dicha herramienta contiene un módulo de procesamiento de datos, donde se pueden hacer análisis a las variables involucradas en el entrenamiento de las redes neuronales para lograr mayor calidad de los datos de entrada a las redes, y un módulo de entrenamiento con algoritmos fuera de línea: Perceptrón Discreto y Continuo, Retropropagación, Retropropagación con Momentos, Retropropagación con Tasa de Aprendizaje Variable y Retropropagación con los algoritmos de Levenberg-Marquart.

. Ya que, en la mayoría de los casos, los datos correspondientes a las variables descriptivas de cualquier proceso están almacenados en bases de datos, se hace necesario diseñar una estrategia de integración que permita acceder desde NeuronMaster a servidores de bases de datos por medio de una interfaz que sea poderosa pero a la vez transparente al usuario, labor que es el objetivo principal de este Proyecto de Grado.

Descriptores **Cota**

- Redes Neuronales (Computación) – Investigación- Procesos *
- Bases de Datos Industriales QA76.87 B7

ÍNDICE

ÍNDICE DE FIGURAS

CAPÍTULO 1: Introducción

1.1 Redes Neuronales Artificiales

Las técnicas de Inteligencia Artificial son cada vez más usadas en las diversas áreas de la industria y de investigación académica. El hecho de poder emular en ambientes computacionales la forma de funcionamiento del cerebro humano, el comportamiento de los organismos biológicos y la organización social de los mismos, ha sido objeto de estudio de muchos investigadores en los últimos años [3], dando esto como resultado el desarrollo de diferentes técnicas entre las cuales se cuentan: Redes Neuronales Artificiales, Lógica Difusa, Algoritmos Genéticos, Programación Evolutiva, entre otras.

Una de las áreas que ha sido estudiada más profunda y extensamente dentro de la Inteligencia Artificial es la de Redes Neuronales Artificiales(RNA’s). Las RNA´s son un sistema artificial que trata de modelar la estructura paralela del cerebro, mediante un modelo matemático compuesto por un gran número de elementos de procesamiento interconectados que, como respuesta a entradas externas, procesan información por medio del ajuste en la ponderación de las conexiones entre cada uno de sus elementos. Las dos potencialidades principales de las RNA´s son su estructura masivamente paralela y distribuida, y su habilidad de generalización, que se refiere a aprender de un conjunto de patrones de entrenamiento y luego generalizar el aprendizaje para poder dar respuesta efectiva a patrones que no fueron incluidos en el conjunto original. Estas dos capacidades hacen que para las redes neuronales sea posible resolver problemas complejos (de gran escala) que corrientemente serían insolubles[6].

Cada elemento de una RNA es llamado “neurona adaptativa” o “perceptrón”, y está constituido por los siguientes elementos:

- Un grupo de entradas.
- Un conjunto de pesos.
- Un punto de suma de las entradas multiplicadas por los pesos.
- Una función de activación (discreta ó continua).
- Un algoritmo de entrenamiento.

La estructura de un perceptrón puede ser ilustrada a través de la siguiente figura:

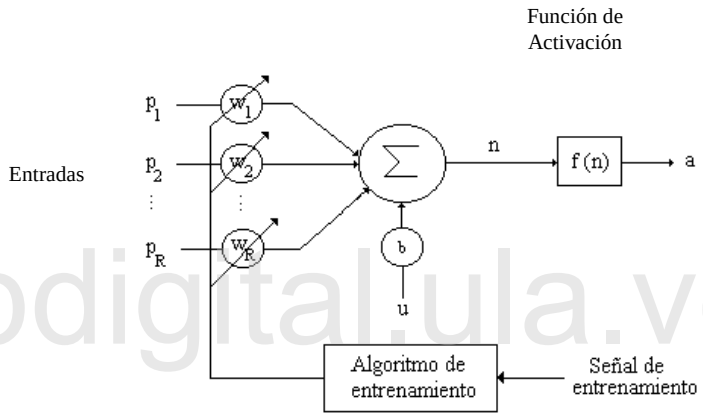


Fig. 1. Estructura de un perceptrón

El proceso de "aprendizaje" de las RNA´s se realiza a través del ajuste de los pesos W_i, lo que se logra mediante el uso de algoritmos de aprendizaje que pueden ser agrupados en dos categorías: *métodos de aprendizaje supervisado*, en donde se dispone de la respuesta correcta para cada patrón de entrenamiento para guiar a la red en el proceso de entrenamiento y encontrar la mejor relación entrada-salida; la segunda categoría es la de los *métodos de aprendizaje no supervisado*, que no requieren la presencia de un supervisor sino que es un proceso de auto-organización, basado en información local, que busca correlaciones con los datos de la red para evaluar el rendimiento de la misma.

A su vez, el proceso de aprendizaje de una red neuronal puede ser en línea o fuera de línea. El aprendizaje en línea supone que cada patrón de entrada de la red modificará la configuración de los pesos, mientras que el aprendizaje fuera de línea supone dos fases, una fase de entrenamiento donde son modificados los pesos de la red, en función de los patrones de entrada y de los algoritmos de entrenamiento, hasta que se cumplan los requerimientos de parada (error mínimo o ciclo máximo), y una fase de implantación donde ya se han obtenido valores estáticos apropiados para los pesos.

Dentro de las aplicaciones que han sido desarrolladas mediante el uso de RNA’s se pueden nombrar modelado de procesos, identificación de procesos, optimización de procesos, monitoreo y control, diagnóstico de procesos, sensores virtuales, fabricación de productos comerciales, entre otros, por lo que en el ambiente industrial las Redes Neuronales han ganado terreno como herramienta de uso común para la resolución de problemas en los que resulta muy complejo o muy costoso utilizar otro tipo de modelos matemáticos[1][3].

1.2 Descripción de NeuronMaster

NeuronMaster es un ambiente para el desarrollo de aplicaciones basadas en redes neuronales realizado en el Postgrado de Ingeniería en Control y Automatización de la Universidad de los Andes, que busca satisfacer las necesidades de las personas que requieran el uso de RNA´s, no sólo en la fase selección de la topología y tipos de algoritmo de entrenamiento de las redes, sino también en la fase de obtención y procesamiento de los datos que sirven como entrada a dichas redes. Las redes neuronales abarcadas en el sistema son totalmente interconectadas del tipo supervisado y fuera de línea.

Los sistemas operativos para los que ha sido desarrollado NeuronMaster son Microsoft Windows 95/98/NT y Linux, utilizando el lenguaje de programación C++ para la codificación de los algoritmos que conforman el sistema. En el caso de la versión desarrollada para Linux se utilizó el lenguaje Tcl/Tk para diseñar la interfaz visual del sistema.

La estructura de NeuronMaster se puede describir a través de los siguientes módulos:

1.2.1. Módulo de Obtención y Procesamiento de Datos:

Los datos que sirven como patrones para cada sesión de entrenamiento de la red neuronal deben estar almacenados en un archivo plano con extensión sugerida ".ntf" (NeuronMaster Training File). Dentro de dicho archivo cada variable de entrada a la red estará organizada en forma de columna, en donde la primera línea de cada columna será el nombre o etiqueta correspondiente a la variable; por lo tanto el archivo contendrá tantas columnas como variables de entrada y salida tenga la red a entrenar. Este proyecto de grado propone el acceso a bases de datos desde NeuronMaster que permita obtener los datos correspondientes a las variables a ser usadas por la red, desde su origen, para almacenarlos automáticamente en los archivos de entrada; esto permitiría optimizar el uso de la aplicación tanto en tiempo como en la exactitud de los datos a procesar, además de permitir una interacción directa con el proceso donde se va a aplicar la red.

Uno de los problemas fundamentales del uso de las redes neuronales es que los datos que se utilicen para el entrenamiento deben ser de alta calidad, es decir, deben poseer gran cantidad de información. Por esta razón, se ha incluido en NeuronMaster un módulo de procesamiento de datos que permite hacer un análisis estadístico de dichos datos, donde

es posible hacer cálculos de tendencia central para cada señal de entrada (media aritmética, media y moda), además de medidas de variación y asimetría (fluctuación, desviación estándar y varianza) y medidas relativas (coeficiente de variación, matriz de varianzas y covarianzas y matriz de correlación).

Una de las características más importante del módulo de procesamiento de datos es la implantación del método de los componentes principales para lograr reducir el número de variables de entrada a la red a un grupo de variables artificiales que contengan un alto porcentaje (mayor al 90%) de la información de las variables originales. Esté método es particularmente útil en aquellos casos en donde se dispone una gran cantidad de variables de entrada que hacen inmanejable el entrenamiento de la red.

1.2.2. *Módulo de Entrenamiento*

Como se mencionó anteriormente NeuronMaster abarca redes totalmente interconectadas con procesos de aprendizaje supervisado y fuera de línea. Además se utilizaron redes con funciones de activación discretas y continuas.

Las redes con función de activación discreta son redes de una sola capa que se utilizan generalmente para resolver problemas de reconocimiento de patrones, y que utilizan el algoritmo de entrenamiento de Rosenblatt.

Las redes con función de activación continua son redes multicapas que utilizan el algoritmo de retropropagación y sus variaciones, como retropropagación con factor de momento, retropropagación con tasa de aprendizaje variable y el algoritmo de Levenberg Marquardt.

Dentro del módulo de entrenamiento se pueden abrir varias sesiones de entrenamiento a la vez, lo que permite entrenar una misma red neuronal con diferentes configuraciones.

1.2.3. Módulo de Aplicación

Una vez entrenada la red neuronal la información resultante del entrenamiento debe ser utilizada para su implantación en procura de generar una solución al problema planteado. NeuronMaster dispone de un módulo que plasma la información de los pesos y la configuración de la red en algoritmos codificados en lenguajes de programación como C++, Fortran y Matlab, leyendo correctamente dicha información para poder acoplarla de manera sencilla al proceso en donde es requerida.

CAPÍTULO 2: Descripción de los SGDB y los métodos de comunicación usados para la construcción de la Interfaz.

En el siglo XX, la tecnología clave ha sido la obtención, procesamiento y distribución de la información[11]. Las comunicaciones son el área de la tecnología que ha tenido mayor desarrollo en las últimas tres décadas, y especialmente en los sistemas de computación ha habido un avance significativo en esta área, siendo el ejemplo más claro de ello el impacto que ha tenido la Internet la sociedad de los últimos años. El acceso rápido y confiable a los datos ha sido uno de los problemas fundamentales estudiado por los desarrolladores de aplicaciones computacionales, ya sea que dichos datos se encuentren en el computador donde se está ejecutando la aplicación que los requiere, o que estén almacenados en uno o en varios computadores remotos.

El desarrollo de las Redes de Area Local (Local Area Networks [LANs]) y de las Redes de Area Amplia (Wide Area Networks [WANs]), ha permitido pasar de la filosofía que consistía en centralizar todos los procesos y toda la información en supercomputadores (mainframes), predominante hasta la mitad de la década de los ochenta, a la filosofía del uso de múltiples computadoras personales interconectadas, donde se distribuye tanto la información como las aplicaciones a ejecutar, a esta última filosofía se le da el nombre de *Sistemas Distribuidos* .

Uno de los resultados más importantes de la filosofía de los sistemas distribuidos es el *modelo cliente-servidor,* que se refiere a definir servidores especializados en determinadas funciones (manejo de archivos, impresoras, bases de datos, etc.), y clientes donde se ejecutan las aplicaciones que requieren acceder a dichos servidores, donde clientes y servidores están conectados entre sí a través de una LAN o de una WAN.

Este proyecto de grado propone una plataforma de integración que permita conectar a NeuronMaster (aplicación cliente) con diversas bases de datos localizadas en servidores de bases de datos, tomando como ejemplos SQL Server y MySQL. A continuación se procede a explicar las características de los Sistemas de Gestión de Bases de Datos (SGBD) estudiados para la realización de este proyecto de grado y, posteriormente, los métodos de acceso a estos SGDB a través de una red de área local.

2.1 Descripción de los Sistemas de Gestión de Bases de Datos a los que se puede acceder desde NeuronMaster

2.1.1 Bases de Datos Relaciones y Lenguaje SQL

La mayoría de los SGBD comerciales usan el modelo relacional como base, debido a la sencillez como dicho modelo representa la información. El modelo relacional organiza los datos en forma de tablas que reciben el nombre de *relaciones*; en dichas tablas los diferentes *campos* o *atributos* se representan en forma de columna y cada fila representa un conjunto de valores de datos relacionados entre sí, llamado *registro o tupla.*

El modelo de base de datos relacional se puede definir de la siguiente forma:

Una base de datos relacional es una base de datos donde todos los datos visibles por el usuario están organizados estrictamente como tablas de valores, y donde todas las operaciones de las bases de datos operan sobre estas tablas[5].

El lenguaje de programación estándar para realizar operaciones en bases de datos relacionales es el SQL (Structured Query Language); aunque hay otros lenguajes utilizados por SGBD relacionales, tales como QUEL y QBE, SQL ha logrado imponerse como el más utilizado por los SGDB comerciales más importantes, ya que sus sentencias son frases

sencillas en inglés que permiten acceder a los datos de forma simple y eficiente. Además, el hecho de que se han logrado establecer estándares de SQL por la ANSI (American National Standards Institute) y por la ISO (International Standards Organization) ha permitido que SQL sea un lenguaje portable entre diferentes plataformas computacionales y que pueda ser utilizado por programas de aplicación, e incluso que el código escrito en SQL pueda insertarse dentro de programas escritos en otros lenguajes de programación de alto nivel como C/C++, Fortran, Basic, etc.

Las características antes mencionadas hacen de SQL la herramienta más eficiente y versátil para la obtención de datos de Bases de Datos Relacionales, por esta razón los SGBD basados en lenguaje SQL son los más utilizados en los ambientes industriales y académicos, ambientes a los que está enfocado NeuronMaster. Por otro lado, SQL es una herramienta idónea para la implantación de sistemas con arquitectura cliente-servidor distribuida, ya que permite la interacción entre los sistemas frontales (clientes) y los sistemas dorsales (servidores de bases de datos).

Por las razones expuestas anteriormente, los SGBD que se estudiarán en este proyecto de grado serán del tipo relacional y basados en lenguaje SQL. A continuación se describen los SGDB utilizados para desarrollar esta investigación.

2.1.2 Microsoft SQL Server

SQL Server es uno de los SGBD más importantes del mercado, ya que permite un alto volumen de procesamiento de transacciones dentro de la arquitectura cliente/servidor, ejecutadas sobre redes Windows NT. Además posee herramientas para ejecutar aplicaciones clientes desde Windows 95/98 e incluso UNIX. SQL Server posee una versión del lenguaje SQL llamada Transact SQL, que posee todas las potencialidades de SQL y

además incluye ciertas extensiones que permiten realizar consultas de forma sencilla que serian muy complejas hacerlas en SQL estándar.

SQL Server permite el acceso a las bases de datos construidas en él a través de API´s (Application Programming Interface) que contienen funciones sencillas para acceso, adición y eliminación de datos, creación y modificación de tablas, entre otras funciones. El acceso a datos almacenados dentro de bases de datos SQL Server se realiza por medio de ODBC (Open DataBase Conectivity), que es una API estándar que ha introducido Microsoft® para el acceso no sólo a SQL Server, sino a una gran variedad de manejadores de bases de datos, tales como Oracle, DB2, Sybase, entre otros.

2.1.3 MySQL

MySQL es un servidor de bases de datos multi-usuarios basado en lenguaje SQL, que consiste en un proceso “demonio”, llamado *mysqld,* que se activa en el servidor, donde se almacenan las bases de datos, que tiene la función de procesar las consultas hechas desde las aplicaciones cliente. MySQL trabaja bajo plataformas UNIX, OS/2 y Windows con arquitectura cliente-servidor.

El acceso a las bases de datos almacenadas en un servidor MySQL desde aplicaciones cliente, puede hacerse a través de API’s que están escritas en lenguaje C++. Las API’s de MySQL están almacenadas dentro de las librerías MySQL, y pueden ser accedidas desde cualquier programa elaborado en lenguaje C++ invocando a la librería “mysql.h”. Una lista de las funciones más importantes de las API’s de MySQL es la que se expone a continuación:

Mysql_connect()	Conecta la aplicación cliente con un servidor MySQL
Mysql_init()	Inicia una estructura del tipo *mysql* que permite la conexión con servidor MySQL
Mysql_query()	Realiza una consulta a una tabla de una base de datos almacenada en un servidor MySQL
Mysql_list_dbs()	Devuelve una lista de las bases de datos almacenadas en un servidor MySQL
Mysql_list_tables()	Devuelve una lista de las tablas de una base de datos almacenada en un servidor MySQL
Mysql_list_fields()	Devuelve una lista de los campos perteneciente a una tabla de una base de datos almacenada en un servidor MySQL
Mysql_store_results()	Almacena los datos resultantes de una consulta en una estructura para que puedan ser utilizados por la aplicación cliente
Mysql_num_rows()	Devuelve el número de filas resultantes en una consulta
Mysql_num_fields()	Devuelve el número de columnas resultantes en una consulta
Mysql_close()	Termina una conexión con un servidor MySQL

2.2 Métodos de Conexión utilizados para acceder a los Servidores de Bases de Datos desde NeuronMaster

El objetivo principal de este proyecto de grado es insertar a NeuronMaster dentro de la arquitectura cliente-servidor, para que el acceso a los datos de interés para el entrenamiento de las redes neuronales artificiales se pueda realizar de una forma rápida y confiable. Para cumplir con este objetivo fue necesario construir una interfaz de integración que permite que una consulta hecha desde NeuronMaster pueda obtener respuesta en un Servidor de Bases de Datos que puede estar operando en el computador donde se está ejecutando NeuronMaster o en cualquier otro computador conectado a él dentro de una red de área local, y que los datos resultantes de dicha consulta sean transmitidos de manera eficiente a NeuronMaster. En la siguiente figura se presenta un esquema de la interfaz de integración propuesta:

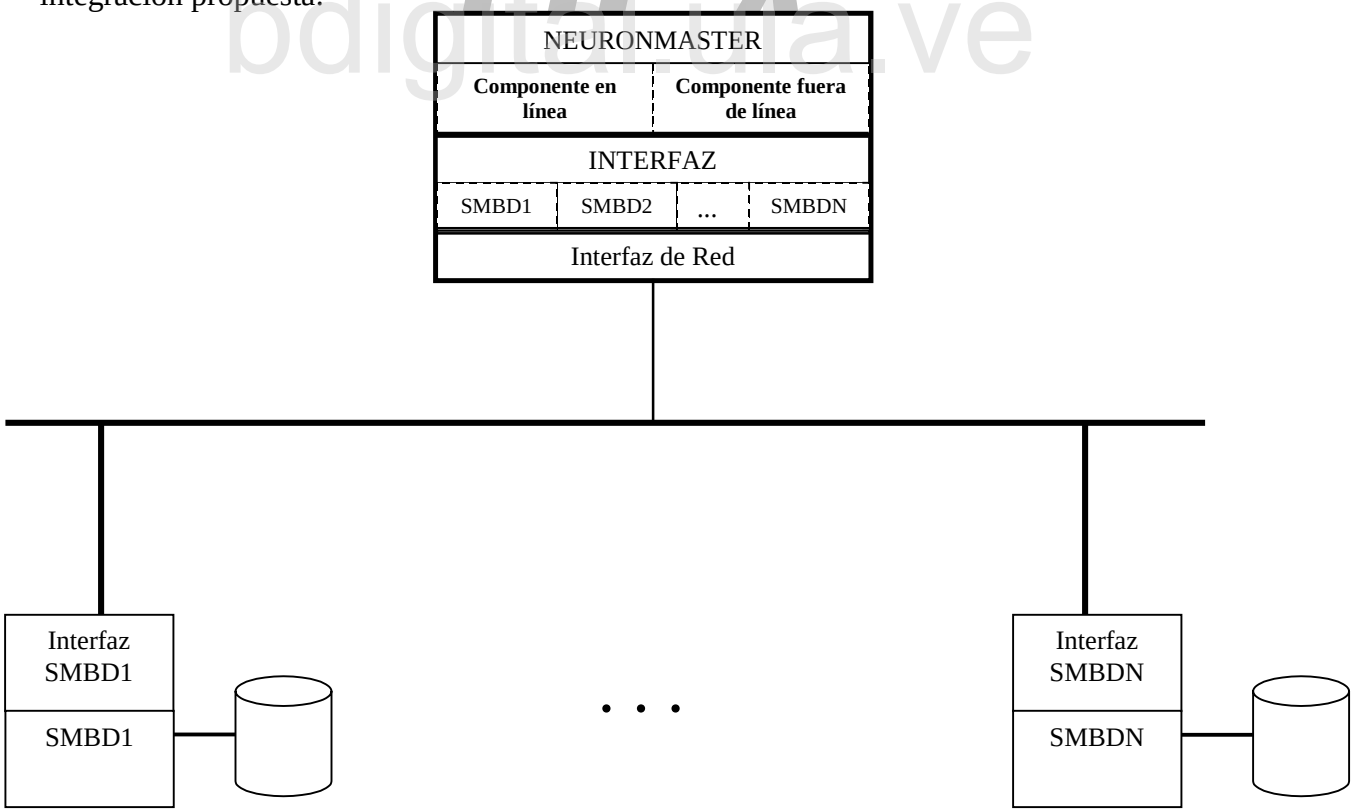


Fig. 2. Esquema de la plataforma de integración entre NeuronMaster y SGDB

Ya que los diferentes SGDB poseen distintas forma de acceso a sus bases de datos, se proponen tres tipos de métodos para la transmisión de las consultas y de la información resultante de dichas consultas entre NeuronMaster y los principales SGDB, dichos métodos son ODBC (Open DataBase Conectivity) que se utiliza en un nivel “alto” de la interfaz ya que en este nivel no es necesario enfrentarse al manejo de transmisión de mensajes a través de la red, RPC (Remote Procedure Call) que se utiliza en un nivel “intermedio” donde los SGDB tienen definidas API’s para el acceso a los datos pero es necesaria la implementación de un protocolo de transmisión de mensajes en la red, y los Sockets utilizado en un nivel “bajo” donde no se dispone de OBC ni de RPC y es necesario definir métodos de obtención y transmisión de datos entre el servidor de bases de datos y la aplicación cliente. Inmediatamente se hace una descripción de cada uno de los métodos de acceso mencionados anteriormente.

2.2.1 ODBC

La Conectividad Abierta a Bases de Datos ú ODBC es un estándar elaborado por Microsoft® que consiste en un conjunto de librerías de enlace dinámico (DLL’s), que permite acceder a un amplio conjunto de servidores de bases de datos mediante programas compilados escritos en lenguaje C o C++ con sentencias de lenguaje SQL embebidas.

ODBC es uno de los métodos más populares para el acceso a SGBD, ya que mediante un conjunto de funciones almacenadas dentro de API´s, se pueden elaborar programas que permitan a una aplicación cliente acceder a la información existente en una base de datos. Entre los servidores de bases de datos que ofrecen conexión a través de ODBC se encuentran Microsoft SQL Server, Oracle, DB2 y Sybase, entre muchos otros. En

este proyecto se presenta la implementación de un módulo de conexión ODBC a bases de datos construidas en Microsoft SQL Server, el cual sirve de modelo para el uso de ODBC con cualquier otro servidor de bases de datos que lo soporte.

Una de las ventajas de ODBC es que separa la aplicación cliente del SGDB, es decir, el programa cliente no debe enterarse con cual SGDB está interactuando, sólo debe conocer bien las funciones albergadas en las API's de ODBC. Está característica permite la conexión a SGDB por usuarios que no necesariamente tienen que ser expertos en el uso de determinado servidor de bases de datos, sólo hace falta que conocer el lenguaje SQL para construir las consultas y el lenguaje C para la implementación de los programas que las ejecutan.

A continuación se describirán algunas de las funciones más importantes del estándar ODBC, las cuales se utilizaron para el módulo ODBC de la interfaz que se desarrolló en esta investigación.

SQLAllocEnv()	Inicia una variable del tipo HENV que permite el manejo de todas las operaciones de acceso al servidor de bases de datos
SQLAllocConnect()	Inicia una variable del tipo HDBC que permite manejar la conexión con un servidor de bases de datos
SQLConnect()	Conecta la aplicación cliente con un servidor de bases de datos
SQLAllocStm()	Inicia una variable del tipo HSTM que provee información de los mensajes de error, la posición del cursor y la información acerca del estado del proceso iniciado por una sentencia SQL.
SQLExecDirect()	Ejecuta una consulta en el servidor de bases de datos, elaborada en lenguaje SQL

SQLNumResultCols()	Retorna el número de columnas resultantes de una consulta
SQLDescribeCol()	Retorna una descripción de cada una de las columnas resultantes de una consulta. La descripción consiste en nombre del campo, tipo de dato, tamaño de la columna (en bytes), número de dígitos decimales, y un campo que indica si la columna contiene registros nulos.
SQLBindCol()	Organiza la información resultante de una consulta en columnas de datos
SQLFreeStm()	Libera la memoria utilizada por la variable HSTM
SQLDisconnect()	Desconecta la aplicación cliente del servidor de bases de datos
SQLFreeConnect()	Libera la memoria utilizada por la variable HDBC

2.2.2 RPC

La llamada a un procedimiento remoto o RPC consiste en que un proceso que tiene lugar en una máquina A pueda llamar a un procedimiento que se ejecute en una máquina B, transmitiendo la información de una máquina a otra en forma de parámetros. Este protocolo permite la transmisión de mensajes entre una máquina y otra, haciendo creer al usuario de la aplicación cliente que está trabajando en un proceso centralizado. Si la máquina donde se ejecutan los procedimientos es una máquina UNIX, como es el caso estudiado dentro de este proyecto, la recepción del mensaje de llamada a los procedimientos que proviene de la máquina cliente está a cargo del proceso portmap, que tiene asociado el puerto 111 y cuya función es asociar un número de puerto a cada proceso RPC; así todas las llamadas a procedimientos remotos serán recibidas en el puerto 111 por el proceso portmap y luego

distribuidas a los puertos correspondientes a los servicios que ejecutarán dichos procedimientos. Un esquema del funcionamiento del protocolo RPC se describe por medio de la siguiente figura:

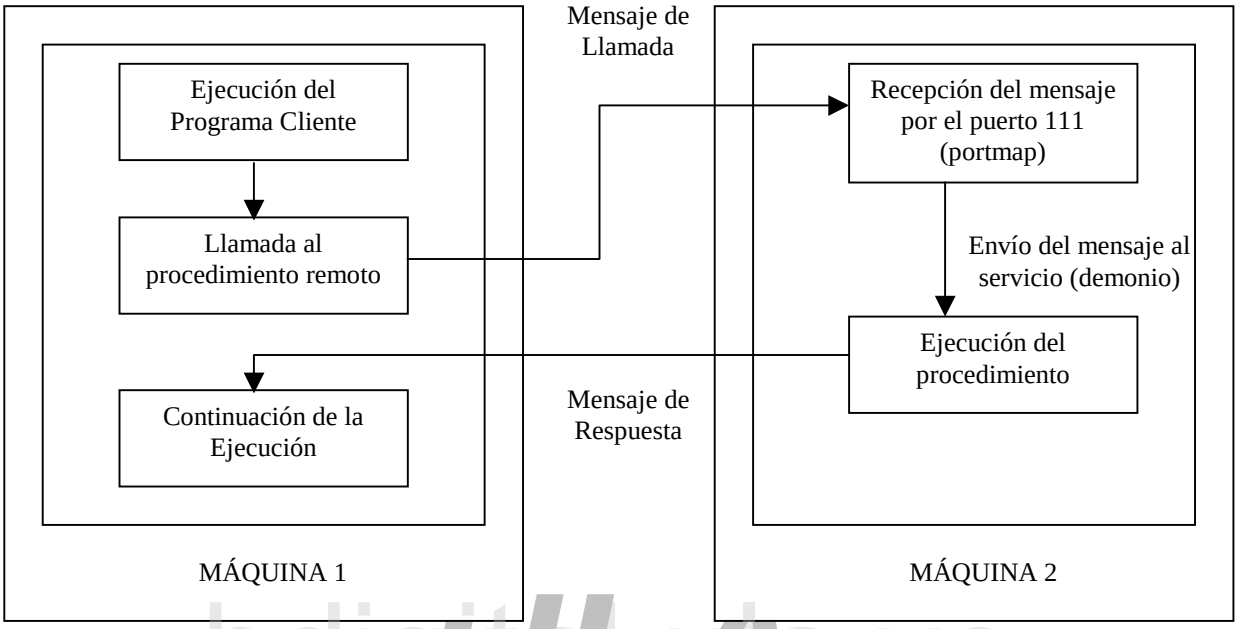


Fig. 3. Esquema de funcionamiento del protocolo RPC

La representación de los datos que se transmiten en los mensajes se hace bajo el protocolo XDR (external data representation), que es una representación estándar de los datos independiente de la estructura física de los mismos, de manera de que la representación de los datos en máquinas de distinto tipo no es un obstáculo para el envío, transmisión y recepción de los mensajes.

Todos los procedimientos que van a ser invocados bajo una estructura RPC deben estar agrupados en un programa, del cual pueden definirse varias versiones para permitir su evolución.

La estructura del protocolo RPC consta de archivo de protocolo con extensión “.x” que contiene la identificación del programa, las versiones del mismo y los procedimientos

agrupados en él. Los programas serán identificados por un número hexadecimal que puede estar comprendido entre 0x00000000 y 0xffffffff; así mismo la identificación de la versión y de los procedimientos se harán por medio de un número entero. En el archivo mencionado anteriormente deben definirse también las estructuras en donde se van a pasar los parámetros y en donde se van retornar el resultado de los procedimientos.

También debe definirse un archivo elaborado en lenguaje C que contenga la implantación de los procedimientos identificados en el archivo de protocolo, cuya estructura no será diferente a la de un procedimiento normal elaborado para ejecución local; por otro lado también se debe escribir un programa cliente que defina las llamadas a los procedimientos (escritos en el archivo que se acaba de mencionar) dentro de la aplicación cliente. En dicho programa se debe declarar una variable del tipo CLIENT, definido en la librería "*rpc.h*", que va permitir la conexión con el servidor donde se van ejecutar los procedimientos, lo que se logrará mediante la llamada a la función "*clnt_create()*"; a la mencionada función se le deben pasar como parámetros una estructura con el nombre del servidor (Host), el nombre del programa que contiene los procedimientos así como el nombre de la versión del programa, y el tipo de protocolo de transmisión de datos (TCP ó UDP). Una vez realizada la conexión con el servidor se podrá hacer las llamadas a los procedimientos remotos igual que si estuviera llamando a procedimientos de ejecución local.

Aunque los archivos donde realmente se hace uso de las funciones del protocolo RPC pueden realizarse por el programador, en este proyecto se utilizó el comando "*rpcgen*" que, utilizando el archivo de protocolo mencionado anteriormente, elabora dichos archivos. Los archivos construidos por el *rpcgen* son los siguientes: un archivo "*svc*", que definirá todas las funciones del protocolo RPC que requiere el servidor; un archivo "*clnt"* que

definirá todas las funciones del protocolo RPC que requiere el programa cliente, y un programa "*xdr*" donde se realiza la definición en protocolo XDR de los datos que se transmitirán por medio de la red.

A continuación se describen brevemente las funciones más importantes del protocolo RPC utilizadas para el módulo RPC de la interfaz, invocadas en los programas clnt, svc y xdr:

Archivo clnt:

clnt_call() Ejecuta la petición de ejecución de un procedimiento de nombre dado, en una máquina de nombre dado, pasándole los parámetros propios del procedimiento en formato XDR.

Archivo svc:

Svcudp_create() Devuelve un puntero a una estructura SVCXPTR que permite la creación del socket y su enlace al puerto UDP de servicio en la máquina servidor.

Svctcp_create() Devuelve un puntero a una estructura SVCXPTR apoyándose en el protocolo TCP.

svc_register() Registra el programa y la versión del programa donde están almacenados los procedimientos a utilizar

Svc_run() Coloca el servidor en espera para la recepción de peticiones del programa cliente

Archivo xdr:

Xdr_pointer	Transforma los tipos de datos descritos en el archivo ".x" a un tipo puntero dentro del protocolo xdr para que los mensajes puedan ser transportados por la red.

En este proyecto se utilizó el protocolo RPC para lograr la conexión entre NeuronMaster y un servidor de bases de datos MySQL que corre bajo plataforma LINUX. La interfaz lograda describe un protocolo para transmisión de datos en la red y a la vez sirve como modelo para que se pueda lograr la conexión con cualquier otro tipo de servidor de bases de datos, ya que lo que podría cambiar es el uso de las API's para cada SGDB, sin que esto signifique un cambio estructural en la interfaz.

2.2.3 Sockets

Un socket es un punto de comunicación por el cual un proceso puede emitir recibir información[9]. Los sockets o enchufes permiten la transmisión de mensajes, en forma de cadenas de bits, entre varias máquinas, convirtiéndose así en una de las partes fundamentales del modelo cliente-servidor. Ya que la transmisión de mensajes con el uso de sockets es en forma de cadenas de bits, al recibir un mensaje se debe realizar un "*cast*" que permita al compilador interpretar el mensaje recibido como el tipo de dato que se requiere, es decir, darle al mensaje el formato del tipo de dato que poseía originalmente, antes de su envío.

El uso de sockets está definido por una serie de funciones llamadas *primitivas,* las que se describen a continuación:

- Socket: crea un socket nuevo y devuelve un número entero que funciona como descriptor del mismo

Sintaxis:

int socket(dominio, tipo, protocolo);

descripción de los parámetros:

dominio: entero que identifica la *familia* que va usar el socket. Entre las familias más conocidas están AF_UNIX, para el dominio UNIX, y AF_INET, para el dominio Internet.

tipo: indica el tipo de propiedades de las comunicaciones en las cuales va a usarse el socket. Los tipos pueden ser: SOCK_DEAM, para servicio no orientado a conexión, SOCK_STREAM, para servicio orientado a conexión, SOCK_RAW (sólo disponible para el *root*), si se necesita acceder directamente a la capa de red, y SOCK_SEQPACKET para las comunicaciones que se encuentran en el dominio XEROX NS.

protocolo: indica el protocolo de la capa de red a utilizar; los protocolos pueden ser UDP y TCP.

- Connect: se usa para establecer una conexión activa con un servidor remoto.

Sintaxis:

int connect(sock, p_adr, l_adr);

descripción de los parámetros:

sock: descriptor del socket.

p_adr: es un puntero a una estructura del tipo sockaddr (incluida en la librería sys/socket.h) que identifica la dirección del socket remoto.

l_adr: longitud de la información que es apuntada por p_adr.

- Send: se utiliza para enviar mensajes a través de un socket.

  Sintaxis:

  int send(sock, msg, long, opc);

  descripción de los parámetros:

  sock: descriptor del socket;

  msg: cadena de bit que contiene el mensaje a enviar.

  long: longitud del mensaje.

  opc: define una opción para el envío del mensaje, usualmente 0

- Recvfrom: se utiliza para recibir datos en un socket.

  Sintaxis:

  int recvfrom(sock, msg, long, opc, p_exp, p_lgexp);

  descripción de los parámetros:

  sock: descriptor del socket.

  msg: mensaje a recibir.

  long: longitud de msg.

  opción: opción para la recepción del mensaje, usualmente 0.

  p_exp: apunta a una estructura del tipo sockaddr que almacena la dirección del socket remoto.

  p_lgexp: longitud de p_exp.

- Bind: se usa para dar un “nombre” a un socket para que el mismo pueda ser identificado por cualquier proceso.

  Sintaxis:

int bind(sock, p_dir, long);

descripción de los parámetros:

sock: descriptor del socket.

p_dir: apunta a una estructura del tipo sockaddr que almacena la dirección del socket.

long: almacena la longitud de p_dir.

- Listen: es usado por los servidores orientados a conexión para escuchar solicitudes de conexión.

  Sintaxis:

  int listen(sock, long);

  descripción de los parámetros:

  sock: descriptor del socket.

  long: número máximo de peticiones pendientes.

- Accept: se usa para que los procesos del cliente tomen conocimiento de un enlace realizado, retornando el descriptor del socket del cliente.

  Sintaxis:

  int accep(sock, p_adr, long);

  descripción de los parámetros:

  sock: descriptor del socket.

  p_adr: apunta a una estructura del tipo sockaddr que contiene la dirección del socket conectado.

  long: longitud de p_adr.

- Write: permite escribir un mensaje sobre un socket conectado (tiene la misma función de la primitiva send).

  Sintaxis:

  int write(sock, msg, long);

  descripción de los parámetros:

  sock: descriptor del socket.

  msg: mensaje a escribir.

  long: longitud de msg.

- Read: permite leer un mensaje de un socket conectado (tiene la misma función de la primitiva rcvfrom).

  Sintaxis:

  int read(sock, msg, long);

  descripción de los parámetros:

  sock: descriptor del socket.

  msg: mensaje a leer.

  long: longitud de msg.

- Shutdown: es usado para liberar un socket que está conectado bajo protocolo TCP.

  Sintaxis:

  int shutdown(sock, sens);

  descripción de los parámetros:

  sock: descriptor del socket.

sens: opción que indica que se desea dejar de recibir mensajes(0), emitir mensajes(1) o ni recibir ni emitir mensajes(2).

- Close: es usado para liberar un socket que está conectado bajo protocolo UDP.

  Sintaxis:

  int close(sock);

  descripción de los parámetros:

  sock: descriptor del socket.

Mediante el uso de las primitivas anteriormente descritas puede desarrollarse un protocolo de envío y recepción de mensajes entre un cliente y un servidor, con la única restricción de que los mensajes deben transmitirse en forma de cadenas de bits.

Aun cuando en este proyecto no se implantó un módulo de la interfaz basado en sockets, en el próximo capítulo se sugiere un macro-algoritmo para la construcción del mismo.

CAPÍTULO 3: Implantación de los métodos de Acceso a Servidores de Bases de Datos desde NeuronMaster

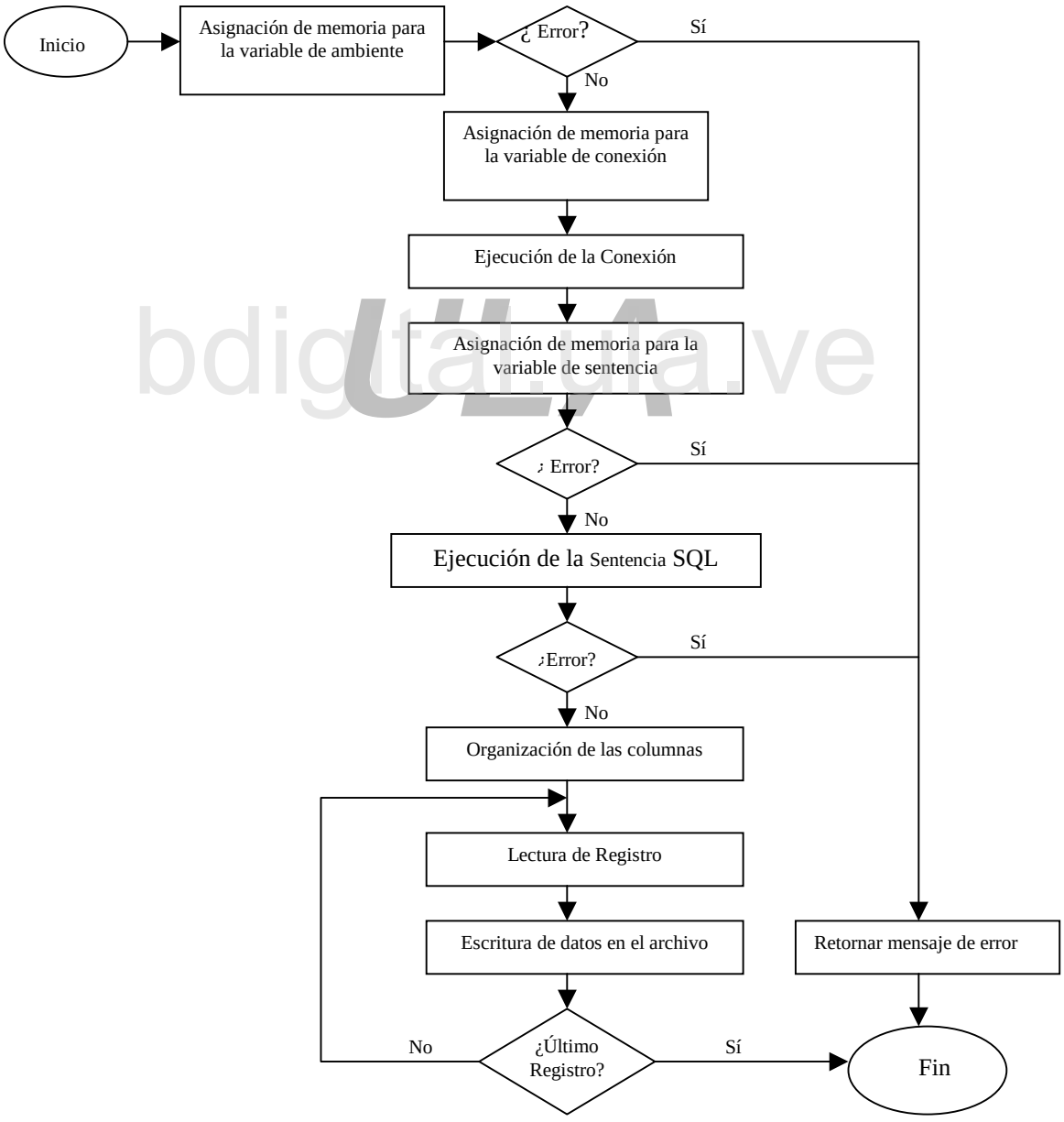
Como se mencionó en le capítulo anterior, la interfaz desarrollada en este proyecto de grado se ha dividido en tres niveles: alto, medio y bajo. En este capítulo se hará una descripción exhaustiva de dichos niveles; en primer lugar se describirá el nivel alto de la interfaz, que consiste en un conjunto de funciones implantadas para el acceso a Bases de Datos mediante el uso del estándar ODBC, utilizando como ejemplo el acceso a bases de datos SQL Server alojadas en un servidor Windows NT. Posteriormente se describirá el nivel medio, que consiste en una metodología para el acceso, por medio de RPC, a datos almacenados en bases de datos alojadas en servidores remotos, tomando como ejemplo el acceso a bases de datos MySQL alojadas en un servidor Linux. Finalmente se sugerirá una metodología para el acceso a servidores de Bases de Datos a través de sockets, que constituye el nivel bajo de la interfaz; en este último caso no se desarrolló un ejemplo de implantación pero, sin embargo, se propone un algoritmo general.

3.1 Nivel Alto: ODBC

En este nivel se desarrolló una función que permite acceder a los datos que estén alojados en cualquier base de datos que soporte el estándar ODBC; particularmente en este proyecto de grado se implantó una función que realiza acceso a datos a un servidor de bases de datos Microsoft® SQL Server®. Se le ha asignado el nombre de “nivel alto” a esta parte de la interfaz, ya que la conexión al SGBD se hace sólo a través de las API´s ODBC y no hace falta elaborar protocolos de comunicación para transmisión de datos en la capa de red, ya que ODBC se encarga de la gestión de datos en la red. Además, con el uso de ODBC no

hace falta conocer la naturaleza del SGDB con el que se hace la conexión (sólo conocer la versión de SQL que soporta), lo que significa que con la función realizada en este nivel de la interfaz puede lograrse la conexión a una gran cantidad de SGDB tales como Oracle®, Sybase®, DB2®, Microsoft® Access®, Microsoft® Visual FoxPro®, entre muchos otros.

A continuación se hace una descripción de la metodología empleada para realizar la función que permite el acceso a datos a través de ODBC:



Para poder iniciar la sesión ODBC es necesario reservar memoria para una variable de ambiente del tipo HENV, que provee acceso la información global del ambiente de conexión ODBC, tal como si la conexión que se va a realizar es válida o los parámetros de una conexión activa. Para lograr realizar la asignación de una variable de ambiente se llama a la función SQLAllocEnv() a la cual se pasa como parámetro la mencionada variable y que puede devolver el mensaje de error SQL_ERROR cuando la función no puede reservar memoria para la variable de ambiente.

Inmediatamente después de que es asignada la variable de ambiente se debe reservar memoria para una variable de conexión del tipo HDBC. Cuando se define un ambiente para la conexión ODBC, bajo el mismo pueden hacerse varias conexiones remotas; como vía para ejecutar se debe asignar memoria para la variable de conexión a través de la función SQLAllocConect(); a dicha función se le pasa como parámetro la variable de conexión.

Para ejecutar la conexión con el SMBD remoto se utiliza la función SQLConnect(), que debe contener como parámetros toda la información referente a la base de datos de donde se desea obtener los datos, así como la información del usuario que va acceder a dichos datos; estos parámetros son: el nombre de la fuente de datos (base de datos de interés), nombre del usuario en el servidor de bases de datos (login) y la contraseña para acceder a dicho servidor (password).

Una vez lograda la conexión con el servidor de bases de datos donde están almacenados los datos a los que se quiere acceder, se debe asignar memoria a una variable de sentencia que va a permitir realizar la consulta SQL al servidor de bases de datos, dicha variable va a albergar toda la información resultante de la ejecución de una consulta SQL hecha por medio de ODBC. Para asignar memoria a la variable de ambiente se utiliza la

función SQLAllocStmt() a la que se pasa como parámetros la variable de conexión y la variable de sentencia.

Ya asignado el espacio en memoria para todas las variables que se necesitan para albergar la información acerca de la conexión con un SMDB, se puede proceder a ejecutar la sentencia SQL que permitirá realizar la consulta que recuperará los datos de la base de datos a la que se hizo la conexión. Esto se logra a través de la función SQLExecDirect() cuyos parámetros son la variable de sentencia, una cadena de caracteres que contenga sentencia SQL y la longitud de dicha cadena de caracteres. Esta función devuelve la información resultante de la consulta en la variable de sentencia y los mensajes de estado de la consulta en una variable del tipo RETCODE.

Cuando ya se ha ejecutado la consulta y se han recuperado los datos de interés, los mismos se organizan en columnas a través de la función SQLBindCol() para posteriormente escribir estas columnas en el archivo de entrenamiento que va a servir de entrada a la sesión de entrenamiento de redes neuronales que se desea planificar. Los parámetros de la función SQLBindCol() son la variable de sentencia, el número de la columna, el tipo de datos a almacenar en la columna, el descriptor de la columna, el tamaño del buffer a escribir en la columna (en bytes) y un puntero que aloja el indicador de longitud del buffer a escribir.

Luego de definir cuantas columnas se utilizarán para alojar la data recuperada, se procede a leer cada registro, uno por uno, a través de la función SQLFetch(). Una vez leídos los datos y organizados en columnas, se procede a escribirlos en el archivo de entrenamiento señalado por el usuario para culminar la función de acceso a datos que conforma el nivel alto de la interfaz desarrollada.

3.2 Nivel Medio: RPC

El nivel medio de la interfaz se implementó para la lograr la conexión con servidores de bases de datos que requieren de un protocolo de transmisión de datos a través de la red, ya sea porque no soporten el estándar ODBC y/o porque la plataforma donde se ejecutan es diferente a la plataforma donde se ejecuta la versión actual de NeuronMaster (Microsoft Windows®).

En el caso estudiado en este proyecto de grado se implantó una serie de funciones para acceso a datos, por medio de llamadas a procedimientos remotos, a bases de datos alojadas en un servidor MySQL bajo plataforma Linux. MySQL contiene una serie de librerías que constan de un conjunto de funciones para acceso a las bases de datos por medio de programas escritos en lenguaje C/C++; dichas librerías se encuentran en los directorios /mysql/include y /mysql/lib del servidor Linux donde está activa la base de datos MySQL, así pues los programas que se desarrollen para el acceso a datos por medio de las librerías antes mencionadas también deben estar guardados en el servidor Linux. Por está razón hace falta hacer llamadas a procedimientos que se ejecuten de forma remota (RPC) para que, desde el computador donde se está ejecutando NeuronMaster, puedan hacerse llamadas a los procedimientos de acceso a las bases de datos MySQL. El protocolo de red utilizado para el desarrollo de este nivel de la interfaz es el UDP ya que se considera que los datos serán transmitidos en redes de área local.

El diseño de un protocolo RPC requiere la construcción de los procedimientos que van a ejecutarse en el servidor remoto, los cuales deben estar agrupados en un solo programa al que se le da el nombre de “programa servidor”, y el desarrollo del programa que hará la llamada a dichos procedimientos desde la aplicación que requiera su uso, a esta aplicación se le llama “programa cliente”. Además el protocolo requiere una serie de

programas, que fueron explicados en el capítulo anterior, en donde se hará uso de las funciones del protocolo RPC para que se haga efectiva la comunicación entre el cliente y el servidor, además de transformar los datos que se van a transmitir a un formato genérico que permita que se manejen entre distintas plataformas; dichos programas son el svc el xdr y el clnt (ver Capítulo 2).

El desarrollo de este nivel de la interfaz propone un protocolo genérico que permita el acceso a cualquier base de datos que contenga librerías (API's) para el acceso a datos desde programas desarrollados en lenguaje C/C++ y que estén alojadas en sistemas que soporten RPC, sin tener que hacer cambios sustanciales en la estructura del protocolo, sólo se deben hacer cambios en las funciones de acceso a datos, las que, en líneas generales, poseen la misma forma para cualquier SGDB. Esto significa que el archivo de protocolo no debe sufrir ningún cambio en el caso de que se necesite acceder a un tipo distinto de servidor de base de datos, es decir, se desarrolló una API genérica para el acceso a datos desde NeuronMaster.

A continuación se hará la descripción del protocolo RPC implantado, haciendo la división entre los programas implementados para el servidor y los implementados para el cliente.

3.2.1 Programa Servidor

El primer paso para construir el programa servidor es escribir un programa que contenga la declaración de los métodos remotos así como el nombre del programa que los va a contener y la versión del mismo, con sus correspondientes números identificadores. Además, este programa debe contener la declaración de todas las estructuras que van a servir para pasar los parámetros a las funciones y para devolver el resultado de los métodos.

El protocolo RPC sólo admite dos parámetros por procedimiento: la estructura donde debe estar almacenada toda la información que se le debe pasar al procedimiento, y una estructura del tipo svc_req en la que debe ir almacenada la información acerca de la conexión RPC.

El archivo de protocolo que se utilizó para declarar los procedimientos y las estructuras utilizadas en la interfaz lleva el nombre de cmysql.x y tiene la siguiente estructura:

```
/*************************************************************************
Estructuras para almacenar datos de fecha y hora de adquisición de los datos
*************************************************************************/

struct date{
        short dia;
        short mes;
        unsigned int ano;
};

struct time{
        short hora;
        short min;
};

/*************************************************************************
Estructura para almacenar cada columna de datos solicitada mediante una consulta sql
*************************************************************************/

struct dato{
        struct date fecha;
        struct time hora;
        float *valor;
};

typedef dato *vec;
typedef char *cadena;


/*************************************************************************
Estructura para almacenar todos los registros resultantes de la ejecución de una consulta sql
*************************************************************************/
```

```
struct data{
        short num_total_paquete;
        short num_paquete;
        cadena *nombres;
        short num_proceso;
        struct dato *datos;
        short err;
        short tam_buffer;
        short num_campos;
};

/*************************************************************************
Estructura para almacenar la información que se le pasará como parámetro a los
procedimientos que se ejecutarán en el servidor
*************************************************************************/

struct info{
        char *Host;
        char *User;
        char *Password;
        char *BD;
        char *Tabla;
        char *Query;
        short num_paquete;
        short num_proceso;
        short tam_buffer;
};

/*************************************************************************
Declaración del programa, la versión, y los procedimientos
*************************************************************************/

program CMYSQLSERV { //Nombre del Programa
    version CMYSQLVERS {//Nombre de la versión
        data OBTENERTABLAS_SERV(info)=1;//Método de petición de tablas de una BD
        data OBTENERCAMPOS_SERV(info)=2;//Método de petición de campos de una
                                                    //Tabla
        data CONSULTA_SERV(info)=3;//Método que ejecuta la sentencia SQL en el
                                            //servidor
        data GETBUFFER_SERV(info)=4;//Método que permite obtener los datos
                                            //resultantes de una consulta que se hayan
                                            //podido mandar en un solo bloque.
                } = 1; //Número de la versión
} = 0x20000002; // Número del programa.

/************************** Fin del Programa **************************/
```

El protocolo elegido para la transmisión de los datos a través de la capa de red es UDP, por lo tanto ya que en algunas consultas la cantidad de datos obtenidos no va a poder enviarse de una sola vez a través de la red (la máxima cantidad de bytes de transmisión que soporta RPC en protocolo UDP es de 8 Kb), es necesario repartir la información en paquetes que serán enviados de manera secuencial por medio del procedimiento GETBUFFER_SERV().

Como puede verse en el esquema anterior en la estructura “info” se almacena toda la información necesaria para la comunicación con el servidor de bases de datos: nombre del servidor (Host), nombre y contraseña del usuario que tiene acceso al servidor (User y Password) , nombre de la base de datos a la que se quiere acceder (BD), nombre de la Tabla de donde se desea obtener los datos (Tabla), cadena que contiene la sentencia SQL a ejecutar (Query), número de paquete que se desea obtener (num_paquete), número de proceso asociado a la sentencia ejecutada (num_proceso) y el número de registros que contiene cada paquete (tam_buffer).

La estructura “data” contiene toda la información que se desea obtener cuando se ejecuta una sentencia SQL: el número de paquetes en que fue dividida la información recolectada por la consulta ejecutada (num_total_paquete), el número de paquete que se está recibiendo actualmente (num_paquete), una estructura que contenga los nombres de las tablas que contiene la base de datos a la que se quiere acceder y los nombres de los campos de cada tabla (nombres), una estructura que almacene los datos resultantes de una consulta (datos), el número de proceso que asignado a la consulta ejecutada (num_proceso), el número de registros que tiene el paquete transmitido (tam_buffer), la cantidad de campos que fueron consultados (num_campos) y el código de error reportado por el procedimiento (err).

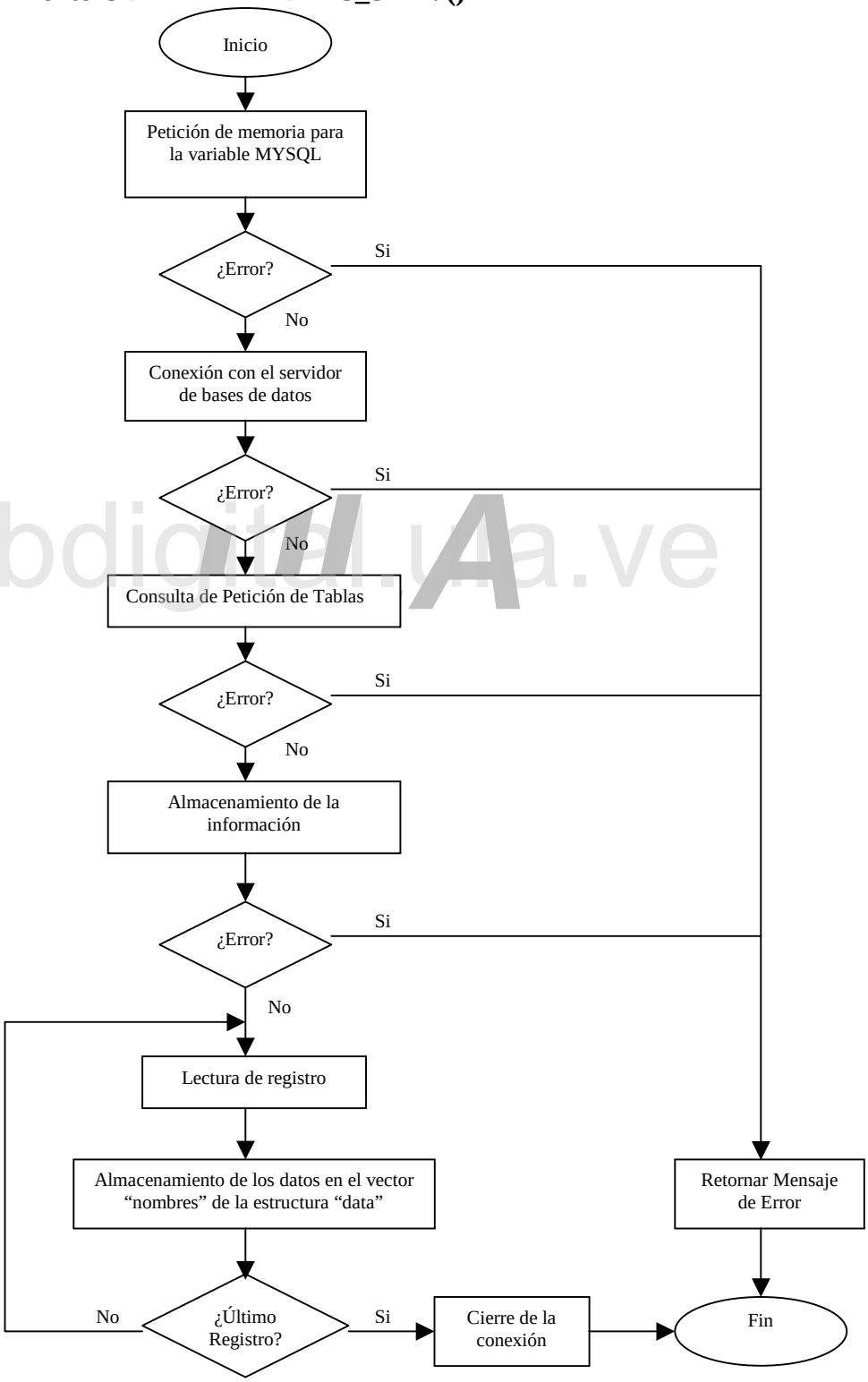
Se construyeron cuatro procedimientos para acceder a los datos: OBTENERTABLAS_SERV() que ejecuta una consulta que solicita los nombres de todas las tablas que contiene la base de datos a la que se quiere acceder, OBTENERCAMPOS_SERV() que ejecuta una consulta que extrae el nombre de todos los campos de una tabla específica, CONSULTA_SERV() que cumple la función más importante del protocolo, ejecutando la sentencia SQL que permite acceder a los datos solicitados por NeuronMaster, y GETBUFFER_SERV() que permite, en caso de que la información no se pueda enviar de una sola vez, solicitar los paquetes restantes de información. Cuando es necesario enviar la información por partes en el procedimiento CONSULTA_SERV() se dispone la información en varios paquetes, enviando como respuesta el primer paquete de información y almacenando los demás paquetes en archivos cuyo nombre está estructurado de la siguiente forma: la palabra “query” concatenada con el número de proceso asignado a la consulta, el número de paquete (desde el 001 hasta 999) y finalmente la extensión “.dat” (ejemplo: query1001.dat). De esta manera los paquetes que pudieron ser enviados por el procedimiento CONSULTA_SERV() pueden ser accedidos posteriormente por el procedimiento GETBUFFER_SERV(). El tamaño de cada paquete de información es definido dividiendo el tamaño total de la información recolectada (en bytes) entre 8Kb que es la tasa máxima de información que se puede enviar mediante RPC.

Cada procedimiento, excepto GETBUFFER_SERV() que no ejecuta consultas, debe contener un conjunto de variables que permitirán la conexión con la base de datos. Estas variables son de los tipos: MYSQL que permite recolectar toda la información relativa a la conexión con el servidor, MYSQLRES que permite almacenar la información resultante de una consulta y MYSQLROW que permite almacenar la información de cada registro de la consulta ejecutada.

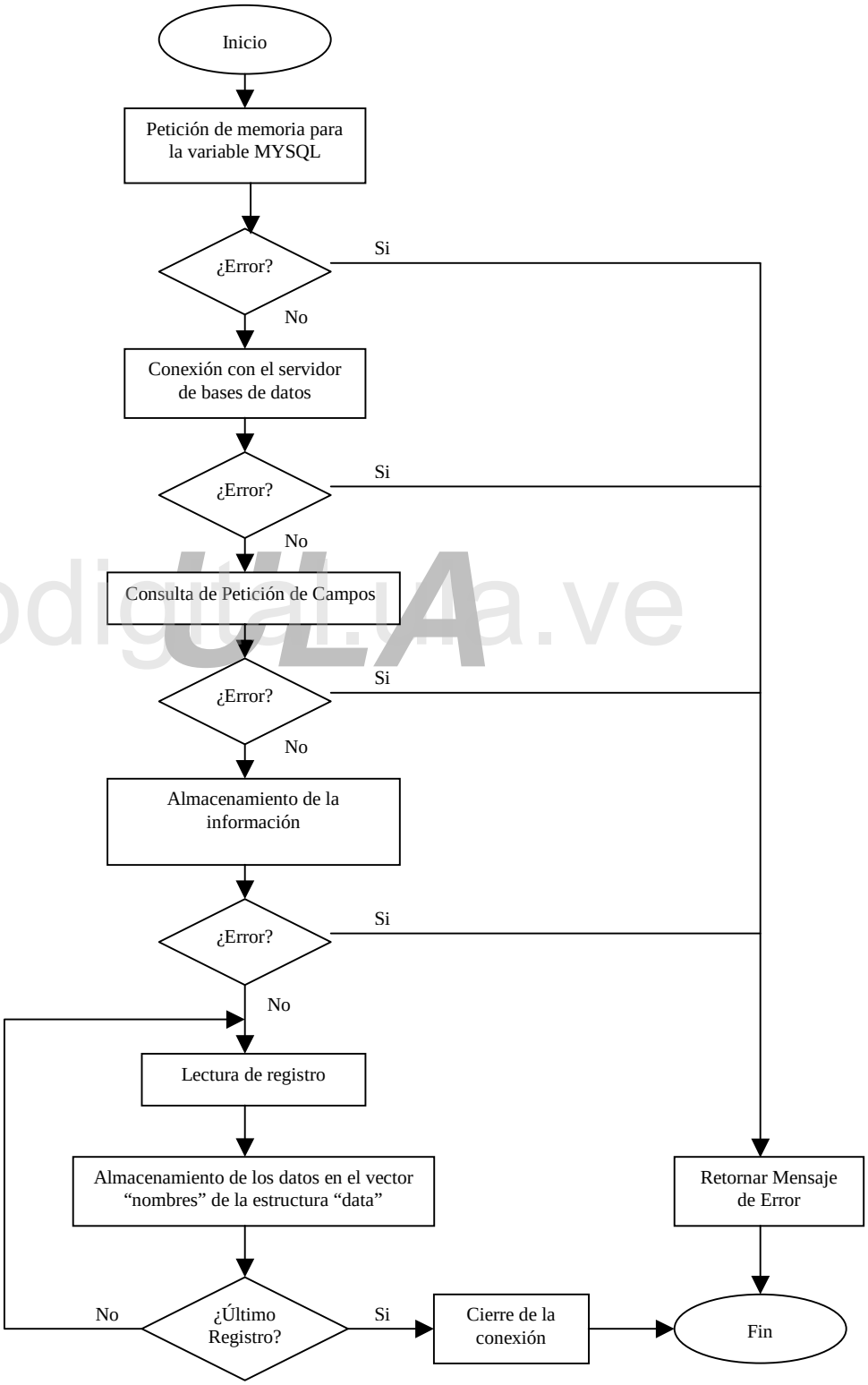
Todos los métodos que ejecutan consultas siguen la siguiente metodología: en primer lugar se reserva memoria para la variable MYSQL y enseguida se procede a iniciar esta variable mediante la función mysql_init(), luego se realiza la conexión con la base de datos de interés mediante la función mysql_real_connect() cuyos parámetros deben ser la variable MYSQL y información relativa al nombre del servidor, nombre y password del usuario que está realizando la conexión y el nombre de la base de datos. Una vez lograda la conexión se procede a ejecutar la consulta mediante la función mysql_query() (excepto en el método OBTENERCAMPOS_SERV() en donde se usa la función mysql_list_fields()) a la que se le deben pasar como parámetros la variable MYSQL y el texto correspondiente a la consulta SQL que se desea ejecutar, por ejemplo, para la función OBTENERTABLAS_SERV() la consulta que se debe ejecutar es “SHOW TABLES”. El resultado de la consulta ejecutada debe almacenarse en una variable del tipo MYSQLRES mediante el uso de la función mysql_store_result(), para luego leer, mediante la función mysql_fetch_row(), uno a uno los registros almacenados y organizarlos dentro de una estructura que se pueda devolver como resultado de la función; en las funciones OBTENERTABLAS_SERV() y OBTENERCAMPOS_SERV() los datos se organizarán en un vector de cadenas de caracteres, pues el resultado de las consultas realizadas por estos métodos será el nombre de las tablas de una base de datos (OBTENERTABLAS_SERV()) o de los campos pertenecientes a una tabla (OBTENERCAMPOS_SERV()); por otro lado, en la función CONSULTA_SERV() los datos se organizarán en una matriz donde las columnas representarán cada uno de los campos y las filas representarán cada uno de los registros recuperados. Como se mencionó anteriormente, la información recuperada será dividida en paquetes, donde el primer paquete será almacenado en la matriz que se va a retornar como resultado de la función, y los registros restantes serán almacenados en una

matriz auxiliar para luego ser en escritos en archivos que podrán ser recuperados por el cliente mediante el uso de la función GETBUFFER_SERV(). A continuación se muestran los diagramas de flujo que esquematizan cada uno de los procedimientos antes descritos:

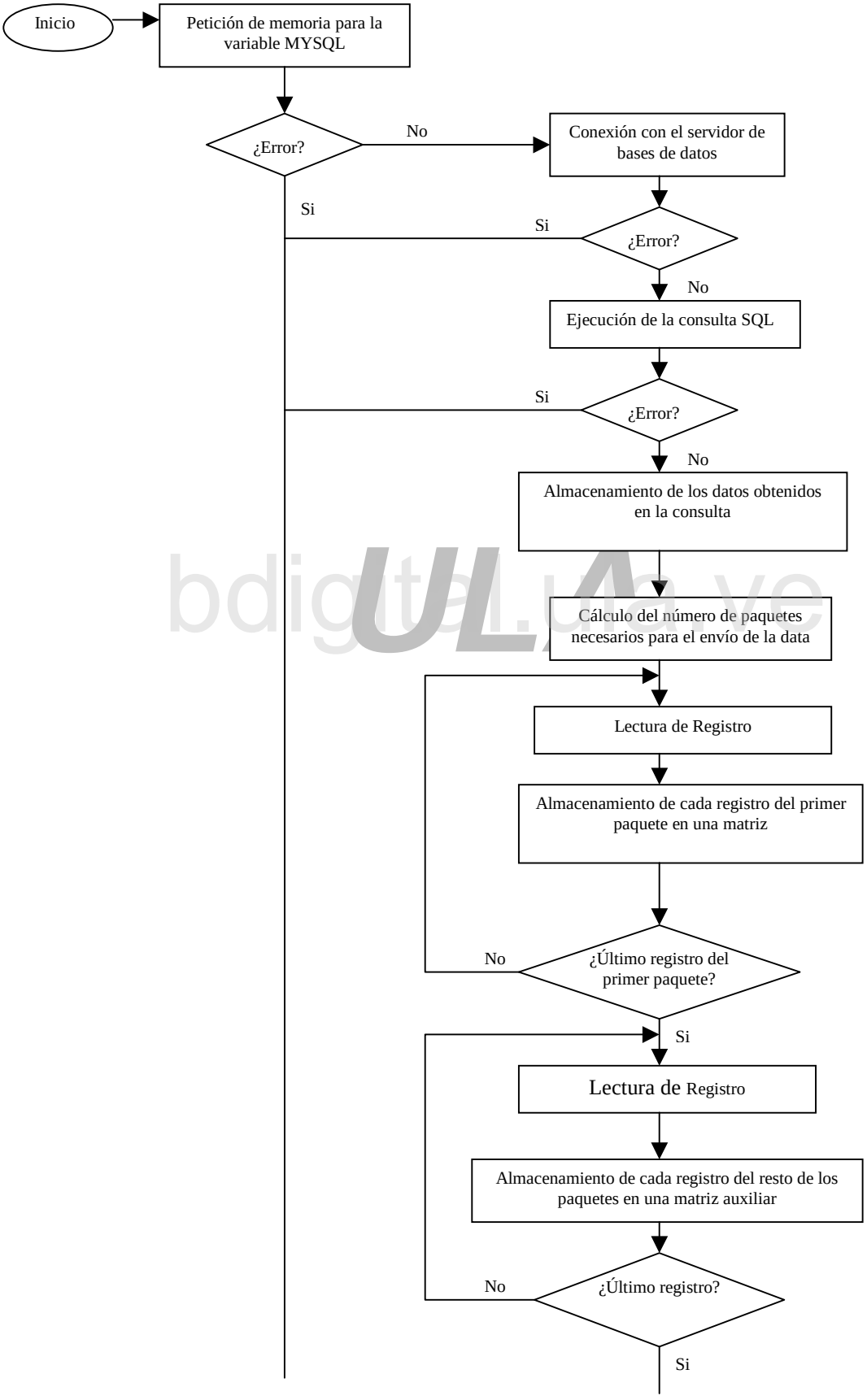
3.2.1.1 Procedimiento OBTENERTABLAS_SERV()

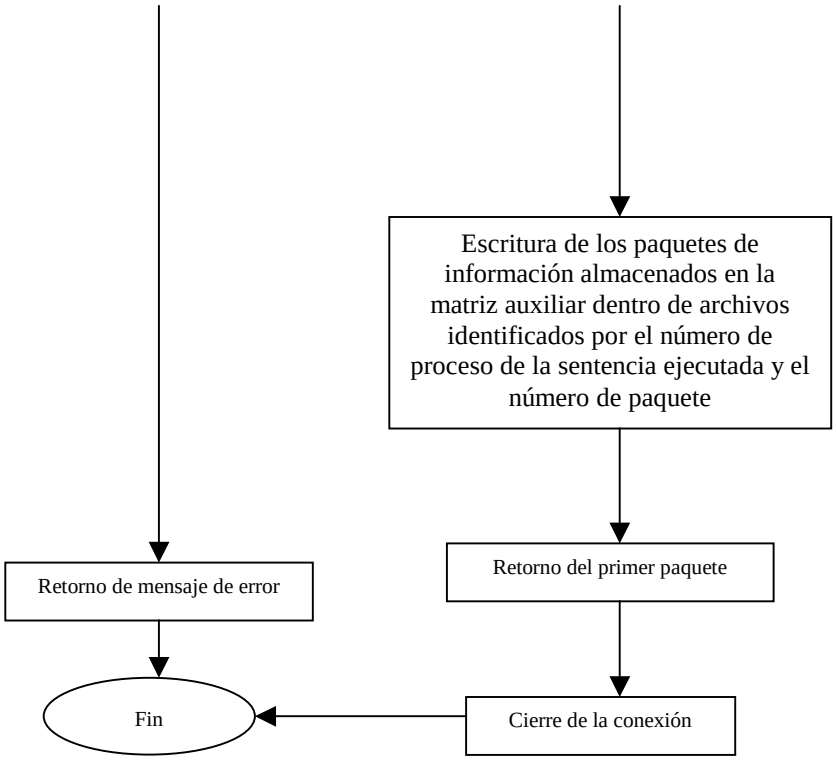


3.2.1.2 Procedimiento OBTENERCAMPOS_SERV()

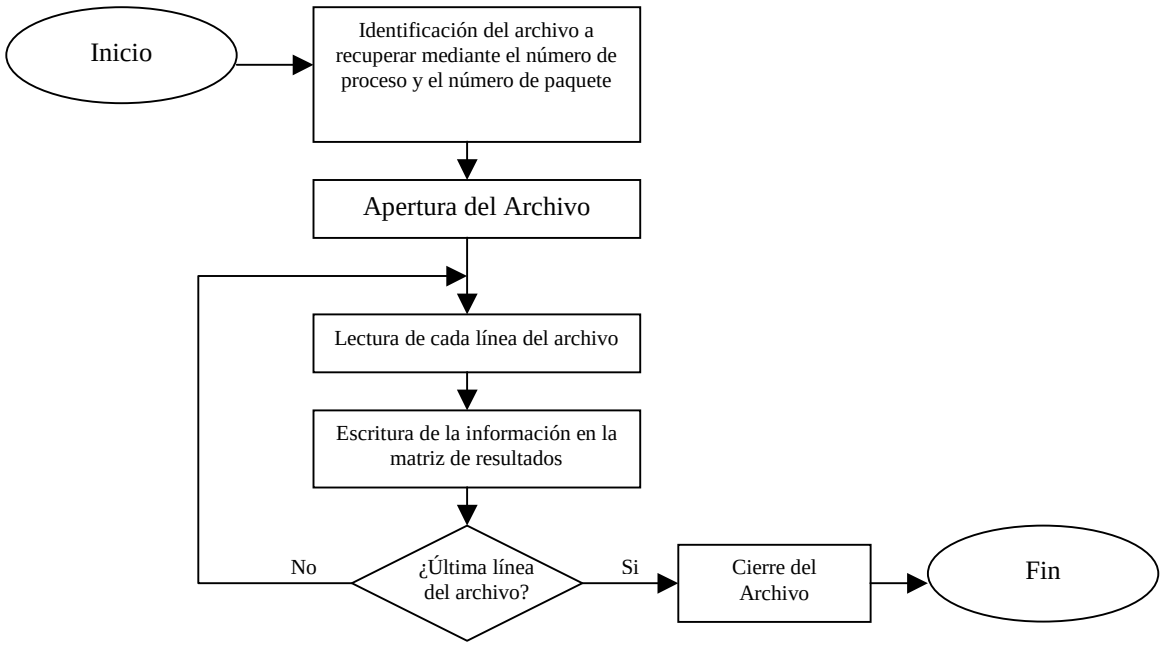


3.2.1.3 Procedimiento CONSULTA_SERV()





3.2.1.4 Procedimiento GETBUFFER_SERV()

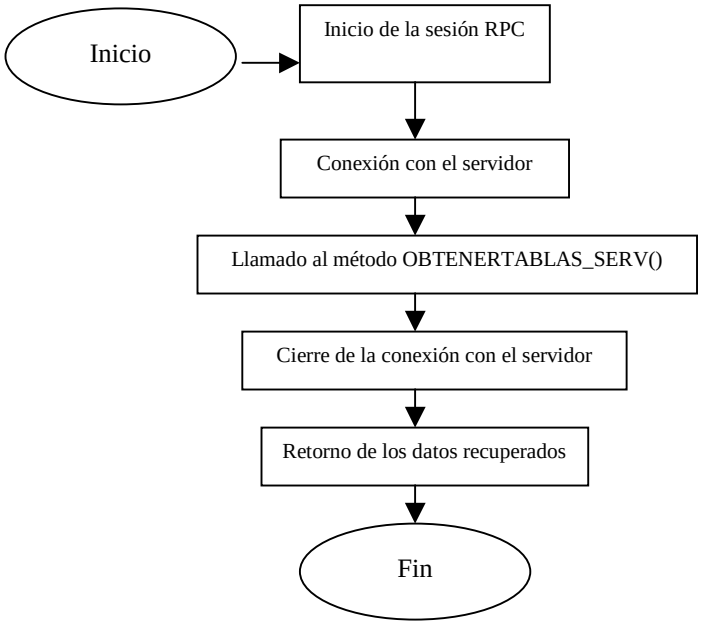


3.2.2 Programa Cliente

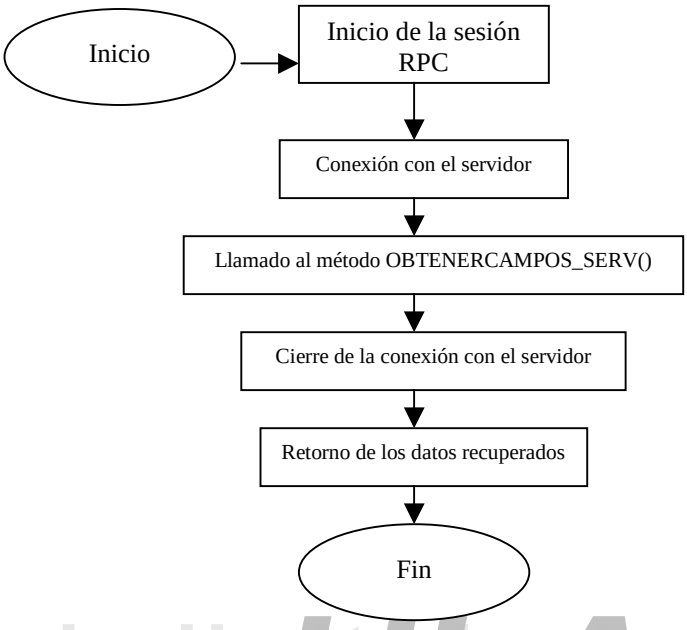
El programa cliente consta de tres funciones: ObtenerTablas_clnt() que solicita las tablas de la base de datos a la que se quiere acceder, ObtenerCampos_clnt() que solicita los campos de la tabla elegida y Consulta_clnt() que realiza la consulta SQL para la obtención de los datos que se requieren. A todas estas funciones se les deben pasar como parámetro la información relativa al nombre del servidor, nombre y password del usuario y nombre de la base de datos. A la función ObtenerCampos_clnt() se le debe añadir un parámetro con el nombre de la tabla de la que se requieren los campos y a la función Consulta_clnt() se le pasa una estructura que contenga todos los datos anteriores y los datos relativos a la consulta SQL a realizar. Todas las funciones deben hacer el llamado a las funciones rpc_nt_init() para iniciar la sesión RPC, clnt_create() para iniciar la conexión y rpc_nt_exit() para terminar la conexión y cerrar la sesión.

En los siguientes diagramas de flujo se esquematiza el funcionamiento de las funciones anteriormente mencionadas:

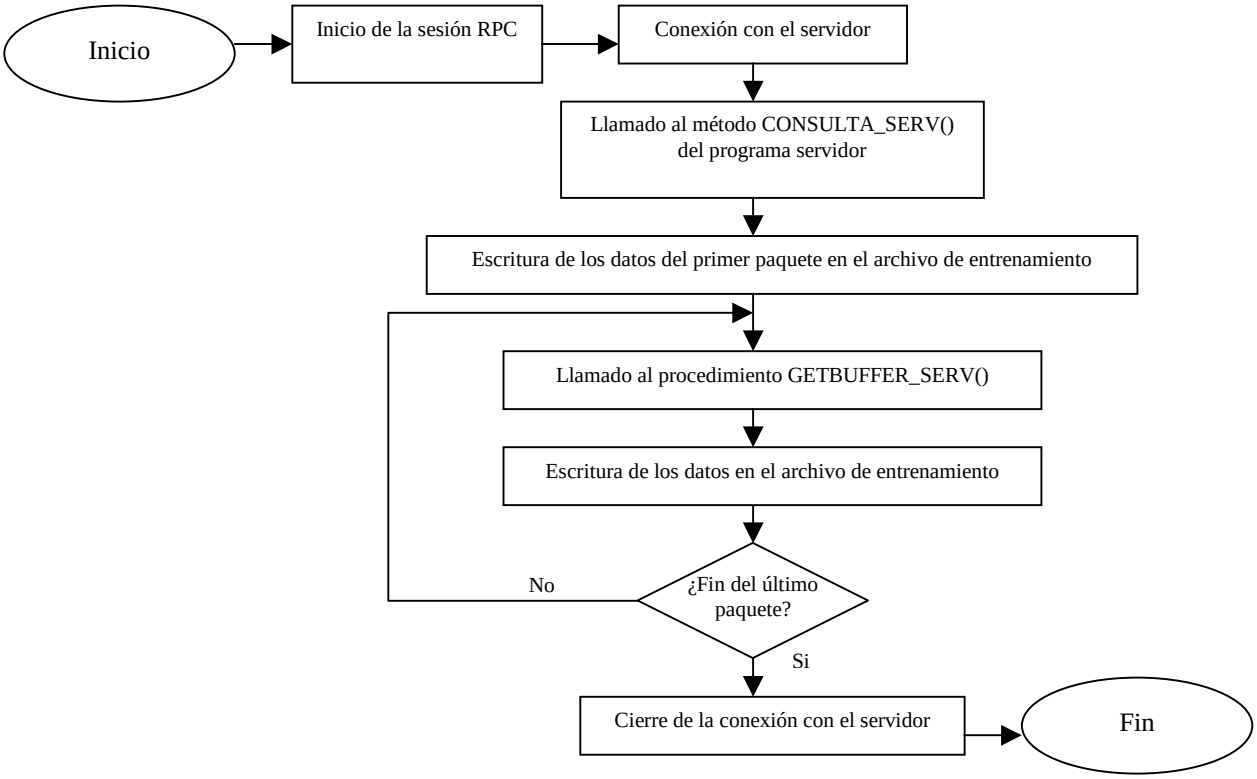
3.2.2.1 Función ObtenerTablas_clnt()



3.2.2.2 Función ObtenerCampos_clnt()



3.2.2.3 Función Consulta_clnt()



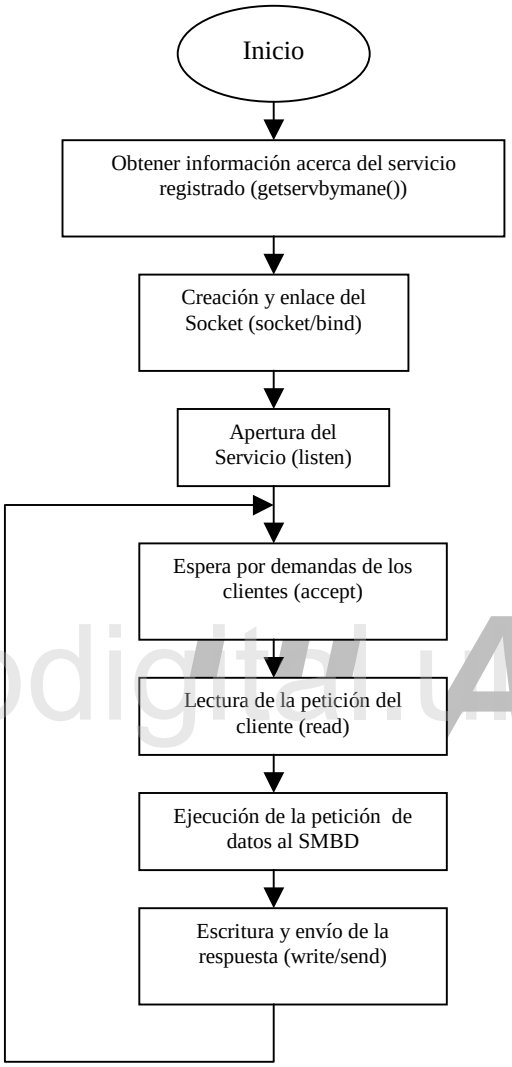
3.3 Nivel Bajo: Sockets

Existen sistemas de computación que no soportan RPC, o utilizan otro tipo de medio de transmisión de datos a través de la capa de red, como por ejemplo sistemas que corren en máquinas VAX, sistemas QNX, redes Novell, etc. Para estos casos es necesario utilizar Sockets para lograr la conexión entre servidores y la transmisión efectiva de los datos. Para NeuronMaster puede ser útil disponer de una metodología para lograr la conexión con este tipo de sistemas, ya que existen SMDB, utilizados sobretodo en ambientes industriales, que funcionan bajo estos sistemas con los cuales puede haber interés de establecer conexiones para la obtención de información.

Hasta ahora NeuronMaster se ha dirigido a funcionar en ambientes computacionales que soportan ODBC o RPC, por lo que no ha habido la necesidad de diseñar un método de conexión para sistemas que no dispongan de estos métodos. Sin embargo, previendo el uso de NeuronMaster en ambientes industriales donde puedan existir los sistemas mencionados, se propone una metodología para establecer conexiones del tipo cliente-servidor a través de Sockets. Como requisito para poder implantar este método de conexión se debe poder acceder a los datos almacenados en los SMBD de interés por medio de sentencias SQL embebidas en programas escritos en lenguaje C.

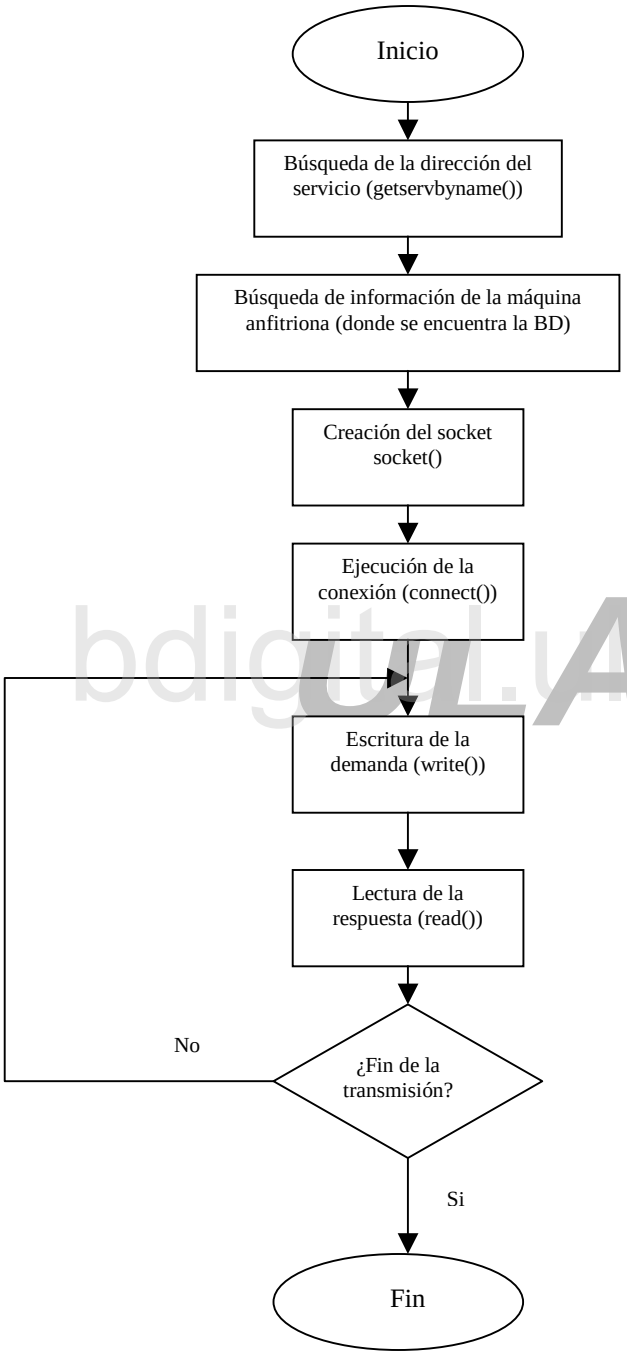
Igual que en la metodología seguida para realizar conexión mediante RPC, la metodología basada en Sockets requiere de la construcción de un programa cliente, donde se ejecutarán las peticiones de datos por parte de NeuronMaster, y de un programa servidor, donde se ejecutan las consultas a las bases de datos. En los siguientes diagramas de flujo se presentarán los pasos a seguir para establecer la conexión mediante Sockets, especificando la estructura del programa servidor y del programa cliente.

3.3.1 Programa Servidor



Como puede observarse en el diagrama de flujo anterior, el programa servidor es un proceso que siempre debe estar activo en la máquina donde se encuentra el SMBD, atento a las peticiones de los clientes. El programa servidor supone la creación de un “servicio” que debe ser registrado junto con el protocolo a utilizar y el número de puerto a usar en el archivo /etc/service, ubicado en la máquina servidor.

3.3.2 Programa Cliente



Capítulo 4: Diseño de la Interfaz con el Usuario

NeuronMaster es un sistema que además de ofrecer un potente entorno de trabajo para el entrenamiento de redes neuronales, busca que su interfaz sea de fácil uso. Por esta razón se diseñó una interfaz en forma de ventanas que proporcionen al usuario la posibilidad de interactuar con el sistema de forma directa por medio de cuadros de diálogo, estructuras de árbol (tal como las que usan los exploradores de internet), menús, botones de acción, y otros controles. En este sentido, la interfaz diseñada para el acceso a bases de datos desde NeuronMaster está organizado en forma de ventanas consecutivas que se abren a medida de que se van cumpliendo los pasos necesarios para la conexión con el servidor y la ejecución de la consulta con la que se obtendrán los datos que se requieren para el entrenamiento de las redes neuronales.

En principio se debe tener acceso a la información referente a los servidores de bases de datos con los que se puede realizar una conexión: nombre del host, dirección IP del servidor, nombre de la base de datos y el método de conexión (RPC, ODBC, Sockets). Para lograr obtener esta información al momento que se quiera obtener información de una base de datos desde NeuronMaster se debe escribir dicha información en un archivo al que se le ha dado el nombre “neuronmaster.ini” donde estarán almacenados los datos de cada base de datos a la que se pueda acceder; la información referente a cada base de datos estará separada por dos palabras reservadas al principio y al final de cada bloque de información: al principio debe ir la palabra DB_INIT y al final debe ir escrita la palabra DB_END. Así pues, la estructura del archivo “neuronmaster.ini” debe ser de la siguiente forma:

DB_INIT

Nombre: Nombre de la base de datos
Método de Acceso: Tipo de Acceso (RPC, ODBC, Sockets)
Host: Nombre del Host
Dirección IP: Número de la dirección IP

DB_END

DB_INIT

.
.
.

DB_END

.
.
.

Para lograr la autenticación del usuario que va a acceder al servidor de bases de datos, cada vez que se realice una conexión se presentará una pantalla que solicita a dicho usuario su nombre de usuario y su respectiva contraseña.

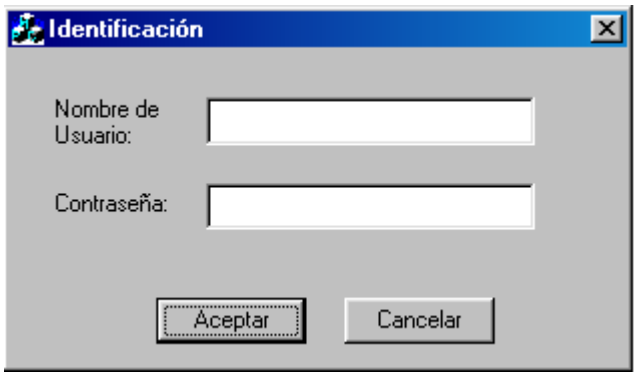


Fig. 4 Pantalla de autenticación

Para poder iniciar la sesión de conexión a bases de datos desde NeuronMaster se ha dispuesto de un item en el menú de herramientas de la pantalla principal del sistema llamado “Conexión a Bases de Datos” y un hipervínculo en una página html que aparece al iniciar la aplicación, como se puede observar a la siguiente figura:

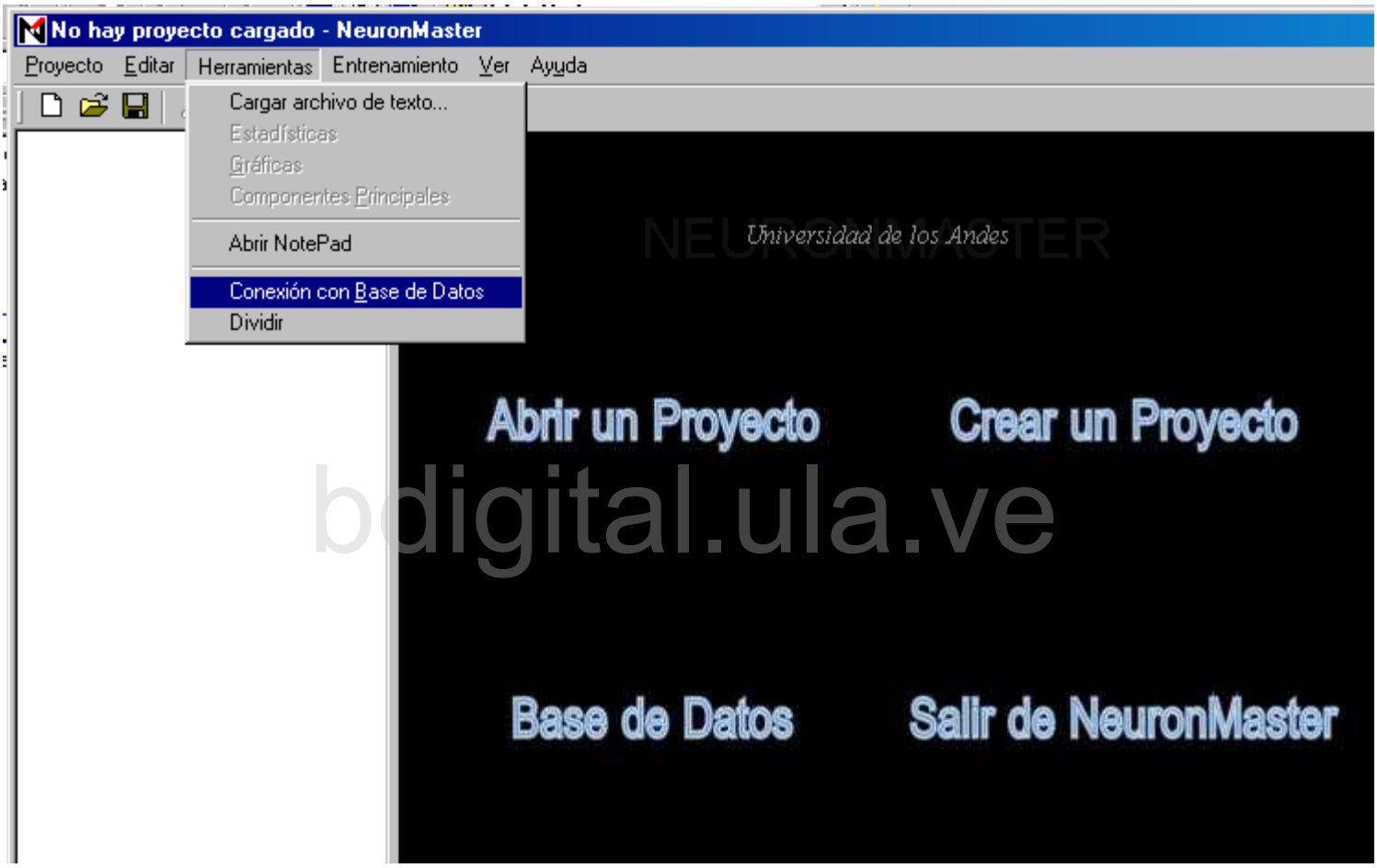
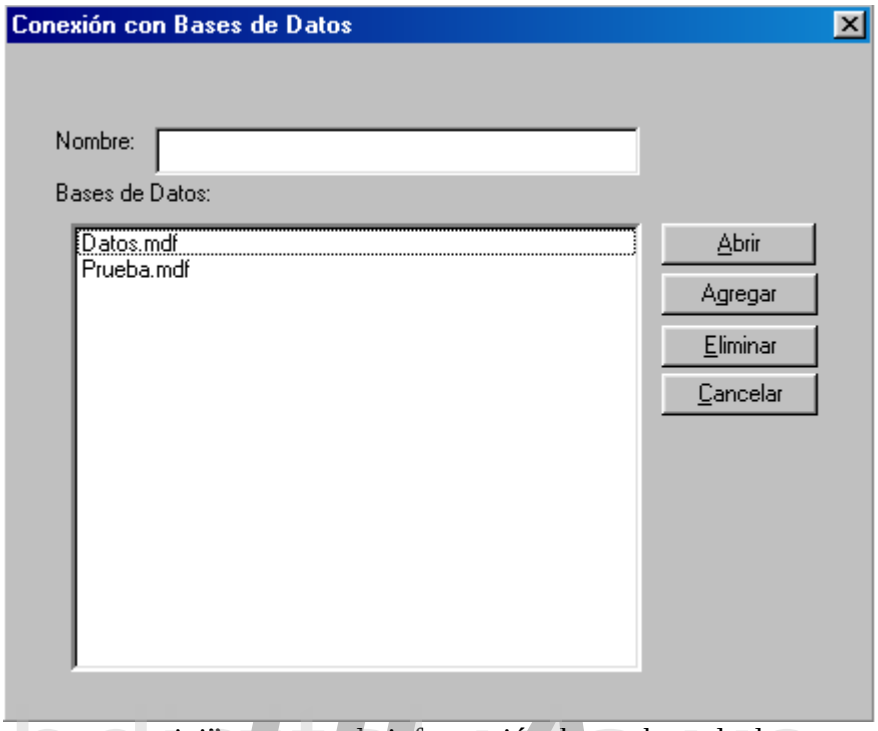


Fig. 5 Acceso a la conexión a bases de datos

Una vez que se pulsa el item de conexión a bases de datos en el menú o el hipervínculo en la pantalla de entrada a NeuronMaster se accede a un cuadro de dialogo donde aparece la lista de las bases de datos registradas en el archivo “neuronmaster.ini”. En este cuadro de dialogo es posible abrir una conexión entre NeuronMaster y una base de

datos elegida de la lista, eliminar la información de una base de datos elegida de la lista en



del archivo “neuronmaster.ini” o agregar la información de una base de datos nueva.

Fig. 6 Cuadro de dialogo de conexión a bases de datos

Si se pulsa el botón “Agregar” en este cuadro de dialogo, aparece una pantalla en donde se le pide al usuario la información necesaria para la conexión con la nueva base de datos, es decir toda la información requerida por el archivo “neuronmaster.ini”. Cuando se culmina de llenar la información referente a la base de datos nueva y se pulsa el botón aceptar, dicha información es escrita en el archivo “neuronmaster.ini”.

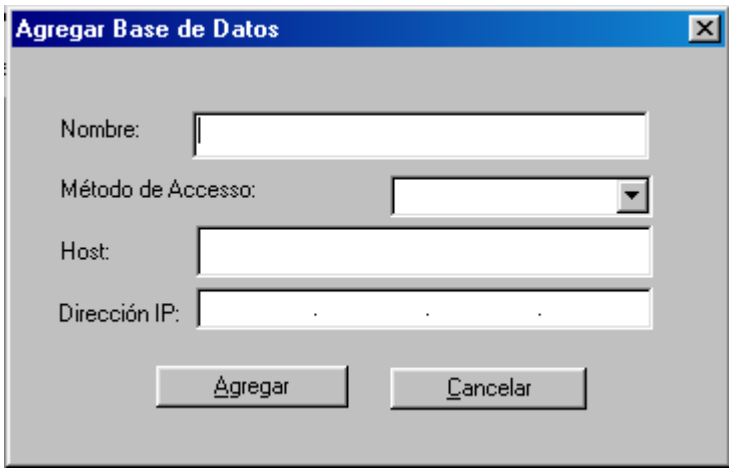


Fig. 7 Cuadro de dialogo de adición de una nueva Base de Datos

Si por el contrario se elige el botón “Abrir” del cuadro de dialogo de “Conexión con Bases de Datos” se abre un nuevo cuadro de dialogo donde aparecen dos listas, una con todas las tablas que pertenecen a la base de datos elegida, y otra que en principio estará vacía donde se reflejaran los campos de la tabla que sea elegida de la primera lista.

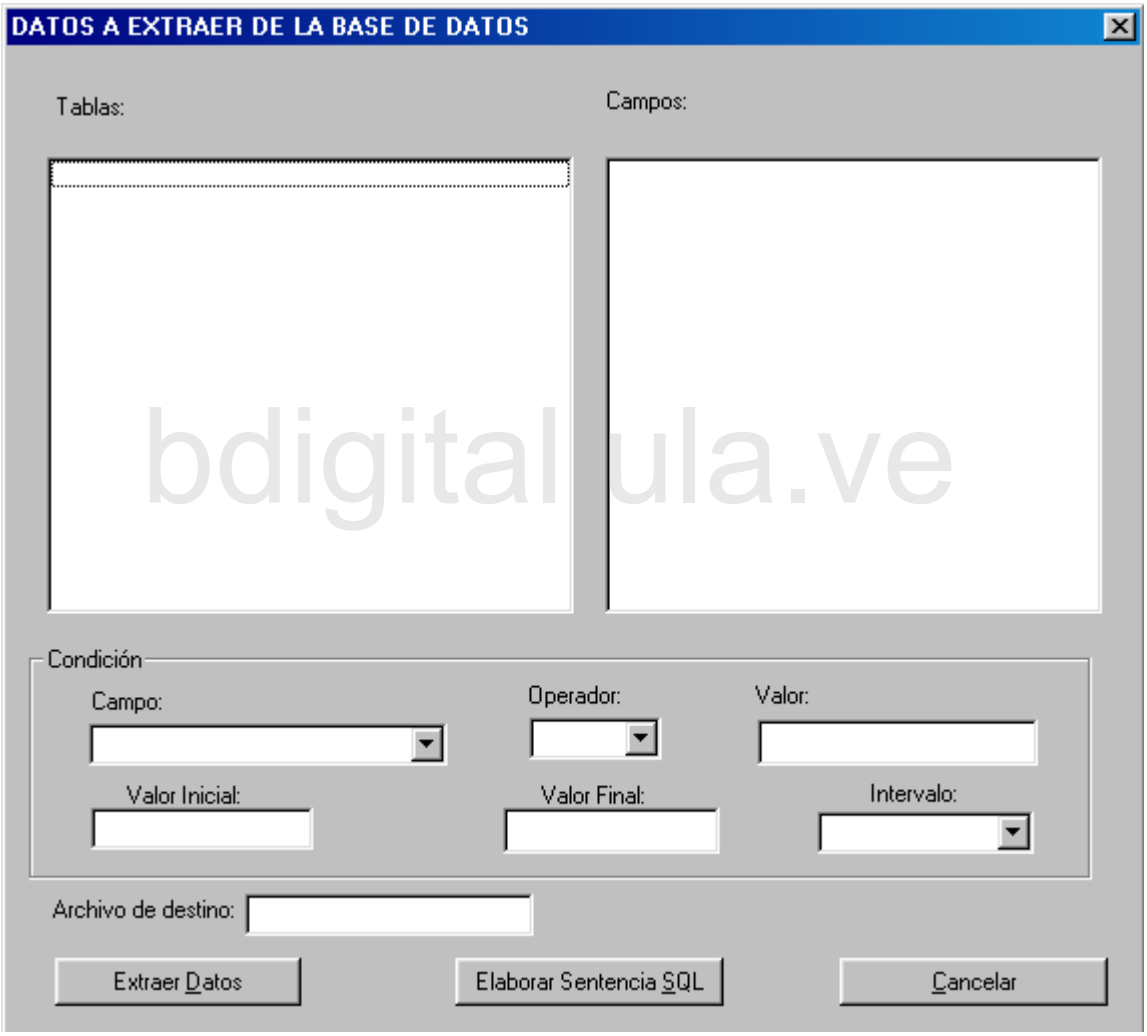
Además, aparecen cuadros de texto donde con los que se puede elegir la condición de una sentencia SQL que tendrá la siguiente estructura:

SELECT <Campos> FROM <Tablas> WHERE <Condición>

Los campos y las tablas son elegidos de las listas que aparecen en el cuadro de dialogo, y la condición puede armarse de la siguiente forma:

<Campo> operador <Valor>

donde los operadores pueden ser: igual que (=), mayor que(>), menor que(<), mayor o igual que(>=) , menor igual que(<=) o entre (en este caso se deben definir los valores inicial y final entre los que debe estar el campo seleccionado y si el campo que va a definir la condición almacena medidas de tiempo se debe definir el intervalo para la toma de los datos). En el caso de que la sentencia SQL prediseñada no pueda satisfacer las expectativas



del usuario se da la opción de elaborar una sentencia SQL propia con sólo pulsar el botón “Elaborar Sentencia SQL”. Una vez construida la sentencia SQL a ejecutar se debe pulsar el botón “Extraer datos” y los datos obtenidos se almacenarán en el archivo cuyo nombre debe ser introducido por el usuario en el cuadro de texto llamado “Archivo de Destino”.

Fig. 8 Cuadro de dialogo de ejecución de sentencia SQL prediseñada

Fig 9. Cuadro de dialogo de ejecución de sentencia SQL diseñada por el usuario

Cada botón de acción de la interfaz de conexión a bases de datos de NeuronMaster está asociado a las funciones que ejecutan las consultas para obtención de datos, explicadas en el capítulo anterior.

La interfaz de conexión a bases de datos del sistema NeuronMaster ha sido diseñada con la idea de hacer transparente al usuario todo el proceso de interacción con los SMBD, la capa de red y con todos los factores que deben involucrarse para lograr que la comunicación entre NeuronMaster y los servidores sea efectiva.

Capítulo 5: Conclusiones y Recomendaciones

5.1 Conclusiones

Se diseñó una interfaz completa para el acceso a servidores de bases de datos desde el sistema para el desarrollo de aplicaciones basadas en redes neuronales NeuronMaster, que permite obtener información almacenada en bases de datos alojadas tanto en el servidor donde se esté ejecutando NeuronMaster, como en servidores remotos, para su uso como patrones de entrada y salida en sesiones de entrenamiento de redes neuronales artificiales. Esto tiene como resultado mejoras con relación al tiempo de captura y confiabilidad en la obtención de la información, y en la organización de los datos de interés en los archivos de entrenamiento.

Se implantó un método efectivo de obtención de datos desde servidores de bases de datos SQL Server usando el estándar ODBC, que no sólo está limitado a interactuar con este tipo de SMBD sino con cualquier SMBD que soporte el estándar ODBC, haciendo posible la conexión de NeuronMaster con la mayoría de los servidores de bases de datos comerciales.

Se implantaron funciones de acceso a servidores de bases de datos MySQL usando como método de conexión RPC, mediante el diseño de una capa de protocolo de transmisión de datos a través de redes de área local (LAN), que puede ser extendido al uso en redes de área amplia y al uso en Internet. Estas funciones proponen una plantilla para la transmisión de datos entre NeuronMaster y cualquier servidor de bases de datos que esté funcionando bajo plataformas que soporten RPC, suponiendo esto un aporte importante para la herramienta, ya que se dispone de un módulo genérico de conexión a servidores remotos.

Se presentó una metodología para el acceso a servidores de bases de datos por medio de Sockets, cuya implantación podría ser de gran importancia en el caso de que NeuronMaster requiera ejecutarse en sistemas computacionales que no soporten ODBC ni RPC, o que utilicen cualquier otro tipo de métodos para la transmisión de datos a través de la capa de red. Esta metodología se presenta considerando que en ambientes industriales es común encontrar ambientes computacionales con las características antes mencionadas.

Toda esta plataforma de acceso a datos se esconde detrás de una interfaz con el usuario de fácil uso, que hace transparente todos los procesos de conexión con los servidores y de petición y captura de datos.

5.2 Recomendaciones

La implantación de la metodología propuesta para la conexión a servidores de bases de datos a través de Sockets sería un aporte importante al sistema NeuronMaster, ya que permitiría abarcar una serie de SMBD que son utilizados con frecuencia en ambientes industriales y académicos.

Es importante la prueba de los métodos implantados en este proyecto de grado con SMBD distintos a los utilizados, para estudiar el rendimiento y alcance de la interfaz desarrollada.

Bibliografía

1. Bose, N.K. y P. Liang. **Neural Networks Fundamentals**. MacGraw-Hill. 1996
2. Coffman, Gayle. **SQL Server 7. Manual de Referencia.** Osborne MacGraw-Hill. 1999.
3. Colina, Eliezer y Rivas, Francklin. **Introducción a la Inteligencia Artificial.** PostGrado en Ingeniería de Control y Automatización. Universidad de los Andes.
4. Gallant, Stephen. **Neural Network Learning and Expert Systems.** The MIT Press. 1993.
5. Groff, James. **Guía de SQL. LAN Times.** MacGraw-Hill. 1998.
6. Haykin, Simon. **Neural Networks. A Comprehensive Foundation.** IEEE Press.1994.
7. Kruglinski, David. **Programación Avanzada con Microsoft Visual C++.** MacGraw-Hill. 1997.
8. Navathe, Shamkant. **Sistemas de Bases de Datos. Conceptos Fundamentales.** Addison Wesley. 1997.
9. Rifflet, Jean-Marie. **Comunicaciones en UNIX.** MacGraw-Hill. 1990.
10. Rosen, Kenneth y Otros. **UNIX Sistema V.** MacGraw-Hill. 1997.
11. Tanenbaum, Andrew. **Redes de Computadoras.** Tercera Edición. Prentice Hall. 1997.
12. Tanenbaum, Andrew. **Sistemas Operativos Distribuidos.** Prentice Hall. 1996.
13. Sitte, Joaquin. Neurocomputing. A Brief Introduction. CeCalcULA. Universidad de los Andes. 1999.
14. Soukup, Ron. **A fondo Microsoft SQL Server 7.0.** MacGraw-Hill.1999.
15. **MySQL Clients Tools and API´s**. MySQL Documentation. www.mysql.com.

ANEXOS

```
/*************************************************************************
Función para acceso a Datos a través de ODBC “conex_odbc.cpp”
Proyecto NeuronMaster
Nivel Alto: ODBC
Desarrollado por: César Enrique Bravo
*************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#include <string.h>
#include <sql.h>
#include <sqlext.h>

#define NUM_COL 50
#define MAX_LENGTH 1024

/*
Códigos de Error
        0: Transacción realizada con éxito
        1: Falla al reservar memoria para la variable de ambiente
        2: Falla al reservar memoria para la variable de conexión
        3: Falla al realizar la conexión
        4: Falla al ejecutar la consulta SQL
        5: Falla al leer un registro
*/

int get_datos_odbc(data_source, user_id, user_pwd, tag_name, query, vres, tam)
        char *data_source;
        char *user_id;
        char *user_pwd;
        char *tag_name;
        char *query;
        float *vres[NUM_COL];
        int *tam[NUM_COL];
{
        HENV henv;
        HDBC hdbc;
        HSTMT hstmt;
        RETCODE rc;
        int num_tot_col, num_col;
        char *nomb_col[NUM_COL];
        int Tam_Buffer;
        int *Tam_Nombre;
        int *DataType;
        unsigned int *Tam_Col;
        int *DecimalDigits;
        int *Nullable;
        SDWORD tipo_col[NUM_COL];

        if (SQLAllocEnv(&henv) != SQL_SUCCESS)
                return (1);

        if (SQLAllocConnect(henv,&hdbc)!= SQL_SUCCESS)
                return (2);
```

```
        rc= SQLConnect(hdbc,data_source,SQL_NTS,user_id,SQL_NTS, user_pwd, SQL_NTS);

        if ((rc != SQL_SUCCESS) && (rc != SQL_SUCCESS_WITH_INFO)
                return (3);

        SQLAllocStmt(hdbc,&hstmt);

        if ((rc = SQLExecDirect(hstmt, consulta, SQL_NTS)) != SQL_SUCCESS)
                return (4);

        rc = SQLNumResultCols(hstmt,&num_col);

        for (int i=0;i<num_col;i++){

                rc =
SQLDescribeCol(hstmt,i+1,nomb_col[i],MAX_LENGTH,Tam_Nombre,DataType,Tam_Col,DecimalDigits,
Nullable);

                rc += SQLBindCol(hstmt, i+1, *DataType, nomb_col[i], Tam_Col, &tipo_col[i]);

        }

        *tam=0;

        for (i=0;i<num_col;i++){

                while (1){

                        rc = SQLFetch(hstmt);

                        if (rc == SQL_ERROR || rc == SQL_SUCCESS_WITH_INFO)
                                return (5);

                        if (rc == SQL_SUCCESS || rc == SQL_SUCCESS_WITH_INFO){

                                vres[i][*tam]=nomb_col[i];

                                (*tam)++;

                        }else
                                break;
                }

        SQLFreeStmt(hstmt, SQL_DROP);
        SQLDisconnect(hdbc);
        SQLFreeConnect(hdbc);
        return (0);
}

}
```

```
/*************************************************************************
Archivo de protocolo cmysql.x
Proyecto NeuronMaster
Nivel Medio: RPC
Desarrollado por: César Enrique Bravo
*************************************************************************/


/*************************************************************************
ESTRUCTURAS PARA ALMACENAR DATOS DE FECHA Y HORA DE ADQUISICIÓN DE LOS DATOS
*************************************************************************/

struct date{
        short dia;
        short mes;
        unsigned int ano;
};

struct time{
        short hora;
        short min;
};

/*************************************************************************
ESTRUCTURA PARA ALMACENAR CADA COLUMNA DE DATOS QUE ES SOLICITADA MEDIANTE UNA CONSULTA SQL
*************************************************************************/

struct dato{
        struct date fecha;
        struct time hora;
        float *valor;
};

typedef dato *vec;
typedef char *cadena;


/*************************************************************************
ESTRUCTURA PARA ALMACENAR TODOS LOS REGISTROS RESULTANTES DE LA EJECUCIÓN DE UNA CONSULTA SQL
*************************************************************************/

struct data{
        short num_total_paquete;
        short num_paquete;
        cadena *nombres;
        short num_proceso;
        struct dato *datos;
        short err;
        short tam_buffer;
        short num_campos;
};

/*************************************************************************
```

```
ESTRUCTURA PARA ALMACENAR LA INFORMACIÓN QUE SE LE PASARÁ COMO PARÁMETRO
A LOS PROCEDIMIENTOS QUE SE EJECUTARÁN EN EL SERVIDOR
**************************************************************************/

struct info{
        char *Host;
        char *User;
        char *Password;
        char *BD;
        char *Tabla;
        char *Query;
        short num_paquete;
        short num_proceso;
        short tam_buffer;
};

/**************************************************************************
DECLARACIÓN DEL PROGRAMA, LA VERSIÓN, Y LOS PROCEDIMIENTOS
**************************************************************************/

program CMYSQLSERV { //Nombre del Programa
        version CMYSQLVERS {//Nombre de la versión
                data OBTENERTABLAS_SERV(info)=1;//Método de petición de tablas de una BD
                data OBTENERCAMPOS_SERV(info)=2;//Método de petición de campos de una Tabla
                data CONSULTA_SERV(info)=3;//Método que ejecuta la sentencia SQL en el
                                           //servidor
                data GETBUFFER_SERV(info)=4;//Método que permite obtener los datos
                                           //resultantes de una consulta que se hayan
                                           //podido mandar en un solo bloque.
                } = 1; //Número de la versión
} = 0x20000002; // Número del programa.

/***************************** Fin del Programa ****************************/
```

```
/*************************************************************************
Funciones de acceso a un servidor de bases de datos MySQL “cmysql.c”
Proyecto NeuronMaster
Nivel Medio: RPC
Programa Servidor
Desarrollado por: César Enrique Bravo
*************************************************************************/



#include <stdio.h>
#include "cmysql.h"
#include <malloc.h>
#include <mysql.h>
#include<string.h>
#include <fstream.h>

#define TAM_BUF 1024
typedef vec *vec1;
typedef float *dat;


data *gettablas_1_svc(info *par, struct svc_req* s_req){
        MYSQL* mysql = NULL;
        MYSQL*res = NULL;
        int i,j, num_registros;
        MYSQL_RES *respuesta;
        MYSQL_ROW registro;
        static struct data result;

        mysql= (MYSQL *) malloc (sizeof(MYSQL));
        res= mysql_init(mysql);

        if (res == NULL){
                result.err=1;
                return(&result);
        }

        res = mysql_real_connect(mysql,par->Host,par->User,par->Password,par->BD,0,NULL,0);

        if (res == NULL){
                result.err=2;
                return (&result);
        }

        if ( mysql_query(&mysql,"SHOW TABLES") ){
                result.err=3;
                return (&result);
        }

        if ( (respuesta = mysql_store_result(mysql)) == NULL) {
                result.err=4;
                return(&result);

        }else{
```

```
                num_registros = mysql_num_rows(respuesta);
                result.nombres = (char **) malloc (sizeof(char *)*num_registros);

                for(j=1;j<=num_registros;j++)
                        result.nombres[j] = (char *) malloc (sizeof(char)*1024);

                i=0;

                while(registro = mysql_fetch_row(p_res)){
                        strcpy(result.nombres[i],registro[0]);
                        i++;
                }

        }

        mysql_free_result(respuesta);
        mysql_close(mysql);
        return(&result);
}

data *getcampos_1_svc(info *par, struct svc_req* s_req){
        MYSQL* mysql = NULL;
        MYSQL*res = NULL;
        int i,j, num_registros;
        MYSQL_RES *respuesta;
        MYSQL_ROW registro;
        static struct data result;

        mysql= (MYSQL *) malloc (sizeof(MYSQL));
        res= mysql_init(mysql);

        if (res == NULL){
                result.err=1;
                return(&result);
        }

        res = mysql_real_connect(mysql,par->Host,par->User,par->Password,par->BD,0,NULL,0);

        if (res == NULL){
                result.err=2;
                return (&result);
        }

        if ( mysql_list_fields(&mysql, par->Tabla, NULL) ){
                result.err=3;
                return (&result);
        }

        if ( (respuesta = mysql_store_result(mysql)) == NULL) {
                result.err=4;
                return(&result);

        }else{

                result.nombres = (char **) malloc (sizeof(char *)*num_registros);
```

```
		for(j=1;j<=num_registros;j++)
			result.nombres[j] = (char *) malloc (sizeof(char)*1024);

		i=0;

		while(registro = mysql_fetch_row(p_res)){

			strcpy(result.nombres[i],registro[0]);
			i++;
		}

	}

	mysql_free_result(respuesta);
	mysql_close(mysql);
	result.err=0;
	return(&result);
}

data *consulta_1_svc(info *par,struct svc_req* s_req){
	MYSQL* mysql = NULL;
	MYSQL*res = NULL;
	int i,j,k,l,m, tam_carga, tam_buffer, num_buffers, num_campos_buffer, num_proceso,
num_registros, num_campos;
	int reg_arch;
	MYSQL_RES *respuesta;
	MYSQL_ROW registro;
	static struct data result,result_aux;
	ifstream f;
	char nombre_arch[100];

	mysql= (MYSQL *) malloc (sizeof(MYSQL));
	res= mysql_init(mysql);

	if (res == NULL){
		result.err=1;
		return(&result);
	}

	res = mysql_real_connect(mysql,par->Host,par->User,par->Password,par->BD,0,NULL,0);

	if (res == NULL){
		result.err=2;
		return (&result);
	}

	if ( mysql_query(&mysql, par->Query) ){
		result.err=3;
		return (&result);
	}

	if ( (respuesta = mysql_store_result(mysql)) == NULL) {
		result.err=4;
		return(&result);

	}else{
```

```
num_registros = mysql_num_rows(respuesta);
num_campos = result.num_campos =mysql_num_fields(respuesta);
tam_carga=num_registros*4;
num_buffers = result.num_total_paquete = tam_carga/778;
result.num_paquete = 1;
//tam_buffer = tam_carga/num_buffers;
tam_buffer = 778;
num_campos_buffer = result.tam_buffer = tam_buffer/4;
result.datos=(vec1 *) malloc(sizeof(vec1)*num_campos));
result_aux.datos=(vec1 *) malloc(sizeof(vec1)*num_campos));
reg_arch=num_registros-num_campos_buffer;

for (l=0;l<num_campos;l++){
        result.datos[l].valor=(float *) malloc( sizeof(float)*num_campos_buffer )
        result_aux.datos[l].valor=(float *) malloc( sizeof(float)*reg_arch )
}

for (j=0;j<num_campos_buffer;j++){
        registro = mysql_fetch_row(p_res))
        for(i=0;i<num_campos;i++)
                result.datos[i].valor[j]=registro[i];
}

for (k=j;k<num_registros;k++){
        registro = mysql_fetch_row(p_res))
        for(i=0;i<num_campos;i++)
                result_aux.datos[i].valor[k] = registro[i];
}
j=0;

for (m=1;m<=num_buffers;m++){
        num_proceso = result.num_proceso = getpid();
        strcpy(nombre_arch,'query');
        strcat(nombre_arch,num_proceso);

        if(m<10){
                strcat(nombre_arch,'00');
        else
                if (m<100)
                        strcat(nombre_arch,'0');
        }

        strcat(nombre_arch,m);
        strcat(nombre_arch,'.dat');

        f.open(nombre_arch);

        for (k=j;k<(num_campos_buffer + j);k++){
                for(i=0;i<num_campos;i++)
                        f<<result_aux.datos[i].valor[k]<<'\t';
                f<<endl;
        }
        j=k;
        f.close();
```

```
                }


        mysql_free_result(respuesta);
        mysql_close(mysql);
        return(&result);
}


data *getbuffer_1_svc(info *par,struct svc_req* s_req){

        char nombre_arch[100];
        char cadena[1024], valor[100];
        static struct data result;
        int i=0,j=0, k=0, l=0;

        strcpy(nombre_arch,'query');
        strcat(nombre_arch,par->num_proceso);
        if(m<10){
                strcat(nombre_arch,'00');
        else
                if (m<100)
                        strcat(nombre_arch,'0');
        }

        strcat(nombre_arch,par->num_paquete);
        strcat(nombre_arch,'.dat');

        f.open(nombre_arch);

        for (i=0;i<par->tam_buffer;i++){
                l=0;
                f.getline(cadena,1024);
                k=0;
                while(cadena[k]!='\n'){
                        j=0;
                        valor[j]=cadena[k];
                        valor[j+1]='\0';
                        k++;
                        while(cadena[k]!='\t'){
                                j++;
                                valor[j]=cadena[k++];
                                valor[j+1]='\0';
                        }
                        l++;
                        result.datos[l].valor[i]=(float *) valor;
                }
        }

        return(&result);
}
```

```
/*************************************************************************
Funciones de acceso a un servidor de bases de datos MySQL “cmysql.c”
Proyecto NeuronMaster
Nivel Medio: RPC
Programa Cliente
Desarrollado por: César Enrique Bravo
*************************************************************************/

#include <rpc/rpc.h>
#include <fstream.h>
#include "cmysql_client.h"
#include "cmysql.h"

//Métodos de Conexión a BD MYSQL

data *ObtenerTablas_clnt(char *Host, char *User, char *Password,char *BD)
{
        CLIENT *clnt;
        struct data *rep;
        struct info parametros;
//      int i;
        strcpy(parametros.Host,Host);
        strcpy(parametros.User,User);
        strcpy(parametros.Password,Password);
        strcpy(parametros.BD,BD);
        clnt = clnt_create(parametros.Host,CMYSQLSERV,CMYSQLVERS,"udp");
        if (clnt == NULL) {
                clnt_pcreateerror(parametros.Host);
                exit (1);
        }
        rep = gettablas_serv_1(&parametros,clnt);
        return(rep);
}



data* ObtenerCampos_clnt(char *Host, char *User, char *Password,char *BD,char *Tabla){
        CLIENT *clnt;
        struct data *rep;
        struct info parametros;
        int i;
        strcpy(parametros.Host,Host);
        strcpy(parametros.User,User);
        strcpy(parametros.Password,Password);
        strcpy(parametros.BD,BD);
        strcpy(parametros.Tabla,Tabla);
        clnt = clnt_create(parametros.Host,CMYSQLSERV,CMYSQLVERS,"tcp");
        if (clnt == NULL) {
                clnt_pcreateerror(parametros.Host);
                exit (1);
        }
        rep = getcampos_serv_1(&parametros,clnt);
        return(rep);
}
```

```
int Consulta_clnt(struct info parametros,char *nombre_arch){
        CLIENT *clnt;
        struct data *rep;
        int i,j, k, num_buffers,num_act_buffer;
        ifstream f;

        clnt = clnt_create(parametros.Host,CMYSQLSERV,CMYSQLVERS,"tcp");
        if (clnt == NULL) {
                clnt_pcreateerror(parametros.Host);
                return(-1);
        }
        rep = consulta_serv_1(&parametros,clnt);
        if (rep->err != 0)
          return(rep->err);
        f.open(nombre_arch);
        if (f.fail())
          return(5);

        for(i=0;i<rep->tam_buffer;i++){
          for(j=0;j<rep->num_campos;j++){
                f<<rep->datos[j]->valor[i]<<'\t';
          f<<endl;
        }

        parametros.tam_buffer=rep->tam_buffer;
        parametros.num_campos=rep->num_campos;
        parametros.num_proceso=rep->num_proceso;

        for(k=1;k<rep->num_total_paquete;k++){
          parametros.num_paquete=k+1;
          rep = getbuffer_1(&parametros,clnt);
          for(i=0;i<rep->tam_buffer;i++){
       for(j=0;j<rep->num_campos;j++){
         f<<rep->datos[j]->valor[i]<<'\t';
       f<<endl;
      }
        }

        f.close();
        return(0);
}
```

```
/*
  Archivo “cmysql.h”
 * Please do not edit this file.
 * It was generated using rpcgen.
 */

#ifndef _CMYSQL_H_RPCGEN
#define _CMYSQL_H_RPCGEN

#include <rpc/rpc.h>


#ifdef __cplusplus
extern "C" {
#endif


struct date {
        short dia;
        short mes;
        u_int ano;
};
typedef struct date date;

struct time {
        short hora;
        short min;
};
typedef struct time time;

struct dato {
        struct date fecha;
        struct time hora;
        float *valor;
};
typedef struct dato dato;

typedef dato *vec;

typedef char *cadena;

struct data {
        char tag_ID[20];
        short num_total_paquete;
        short num_paquete;
        cadena *nombres;
        short num_proceso;
        struct dato *datos;
        short err;
        short tam_buffer;
        short num_campos;
};
typedef struct data data;

struct info {
        char *Host;
```

```
        char *User;
        char *Password;
        char *BD;
        char *Tabla;
        char *Query;
        short num_paquete;
        short num_proceso;
        short tam_buffer;
};
typedef struct info info;

#define CMYSQLSERV 0x20000002
#define CMYSQLVERS 1

#if defined(__STDC__) || defined(__cplusplus)
#define GETTABLAS 1
extern  data * gettablas_1(info *, CLIENT *);
extern  data * gettablas_1_svc(info *, struct svc_req *);
#define GETCAMPOS 2
extern  data * getcampos_1(info *, CLIENT *);
extern  data * getcampos_1_svc(info *, struct svc_req *);
#define CONSULTA 3
extern  data * consulta_1(info *, CLIENT *);
extern  data * consulta_1_svc(info *, struct svc_req *);
#define GETBUFFER 4
extern  data * getbuffer_1(info *, CLIENT *);
extern  data * getbuffer_1_svc(info *, struct svc_req *);
extern int cmysqlserv_1_freeresult (SVCXPRT *, xdrproc_t, caddr_t);

#else /* K&R C */
#define GETTABLAS 1
extern  data * gettablas_1();
extern  data * gettablas_1_svc();
#define GETCAMPOS 2
extern  data * getcampos_1();
extern  data * getcampos_1_svc();
#define CONSULTA 3
extern  data * consulta_1();
extern  data * consulta_1_svc();
#define GETBUFFER 4
extern  data * getbuffer_1();
extern  data * getbuffer_1_svc();
extern int cmysqlserv_1_freeresult ();
#endif /* K&R C */

/* the xdr functions */

#if defined(__STDC__) || defined(__cplusplus)
extern  bool_t xdr_date (XDR *, date*);
extern  bool_t xdr_time (XDR *, time*);
extern  bool_t xdr_dato (XDR *, dato*);
extern  bool_t xdr_vec (XDR *, vec*);
extern  bool_t xdr_cadena (XDR *, cadena*);
extern  bool_t xdr_data (XDR *, data*);
extern  bool_t xdr_info (XDR *, info*);
```

```
#else /* K&R C */
extern bool_t xdr_date ();
extern bool_t xdr_time ();
extern bool_t xdr_dato ();
extern bool_t xdr_vec ();
extern bool_t xdr_cadena ();
extern bool_t xdr_data ();
extern bool_t xdr_info ();

#endif /* K&R C */

#ifdef __cplusplus
}
#endif

#endif /* !_CMYSQL_H_RPCGEN */
```

```
/*
  Archivo “cmysql_svc.c”
 * Please do not edit this file.
 * It was generated using rpcgen.
 */

#include "cmysql.h"
#include <stdio.h>
#include <stdlib.h>
#include <rpc/pmap_clnt.h>
#include <string.h>
#include <memory.h>
#include <sys/socket.h>
#include <netinet/in.h>

#ifndef SIG_PF
#define SIG_PF void(*)(int)
#endif

static void
cmysqlserv_1(struct svc_req *rqstp, register SVCXPRT *transp)
{
	union {
		info gettablas_1_arg;
		info getcampos_1_arg;
		info consulta_1_arg;
		info getbuffer_1_arg;
	} argument;
	char *result;
	xdrproc_t _xdr_argument, _xdr_result;
	char *(*local)(char *, struct svc_req *);

	switch (rqstp->rq_proc) {
	case NULLPROC:
		(void) svc_sendreply (transp, (xdrproc_t) xdr_void, (char *)NULL);
		return;

	case GETTABLAS:
		_xdr_argument = (xdrproc_t) xdr_info;
		_xdr_result = (xdrproc_t) xdr_data;
		local = (char *(*)(char *, struct svc_req *)) gettablas_1_svc;
		break;

	case GETCAMPOS:
		_xdr_argument = (xdrproc_t) xdr_info;
		_xdr_result = (xdrproc_t) xdr_data;
		local = (char *(*)(char *, struct svc_req *)) getcampos_1_svc;
		break;

	case CONSULTA:
		_xdr_argument = (xdrproc_t) xdr_info;
		_xdr_result = (xdrproc_t) xdr_data;
		local = (char *(*)(char *, struct svc_req *)) consulta_1_svc;
		break;

	case GETBUFFER:
```

```
                _xdr_argument = (xdrproc_t) xdr_info;
                _xdr_result = (xdrproc_t) xdr_data;
                local = (char *(*)(char *, struct svc_req *)) getbuffer_1_svc;
                break;

        default:
                svcerr_noproc (transp);
                return;
        }
        memset ((char *)&argument, 0, sizeof (argument));
        if (!svc_getargs (transp, _xdr_argument, (caddr_t) &argument)) {
                svcerr_decode (transp);
                return;
        }
        result = (*local)((char *)&argument, rqstp);
        if (result != NULL && !svc_sendreply(transp, _xdr_result, result)) {
                svcerr_systemerr (transp);
        }
        if (!svc_freeargs (transp, _xdr_argument, (caddr_t) &argument)) {
                fprintf (stderr, "unable to free arguments");
                exit (1);
        }
        return;
}

int
main (int argc, char **argv)
{
        register SVCXPRT *transp;

        pmap_unset (CMYSQLSERV, CMYSQLVERS);

        transp = svcudp_create(RPC_ANYSOCK);
        if (transp == NULL) {
                fprintf (stderr, "cannot create udp service.");
                exit(1);
        }
        if (!svc_register(transp, CMYSQLSERV, CMYSQLVERS, cmysqlserv_1, IPPROTO_UDP)) {
                fprintf (stderr, "unable to register (CMYSQLSERV, CMYSQLVERS, udp).");
                exit(1);
        }

        transp = svctcp_create(RPC_ANYSOCK, 0, 0);
        if (transp == NULL) {
                fprintf (stderr, "cannot create tcp service.");
                exit(1);
        }
        if (!svc_register(transp, CMYSQLSERV, CMYSQLVERS, cmysqlserv_1, IPPROTO_TCP)) {
                fprintf (stderr, "unable to register (CMYSQLSERV, CMYSQLVERS, tcp).");
                exit(1);
        }

        svc_run ();
        fprintf (stderr, "svc_run returned");
        exit (1);
        /* NOTREACHED */
```

```
}
```

```
/*
  Archivo “cmysql_clnt.c”
 * Please do not edit this file.
 * It was generated using rpcgen.
 */

#include <memory.h> /* for memset */
#include "cmysql.h"

/* Default timeout can be changed using clnt_control() */
static struct timeval TIMEOUT = { 25, 0 };

data *
gettablas_1(info *argp, CLIENT *clnt)
{
	static data clnt_res;

	memset((char *)&clnt_res, 0, sizeof(clnt_res));
	if (clnt_call (clnt, GETTABLAS,
		(xdrproc_t) xdr_info, (caddr_t) argp,
		(xdrproc_t) xdr_data, (caddr_t) &clnt_res,
		TIMEOUT) != RPC_SUCCESS) {
		return (NULL);
	}
	return (&clnt_res);
}

data *
getcampos_1(info *argp, CLIENT *clnt)
{
	static data clnt_res;

	memset((char *)&clnt_res, 0, sizeof(clnt_res));
	if (clnt_call (clnt, GETCAMPOS,
		(xdrproc_t) xdr_info, (caddr_t) argp,
		(xdrproc_t) xdr_data, (caddr_t) &clnt_res,
		TIMEOUT) != RPC_SUCCESS) {
		return (NULL);
	}
	return (&clnt_res);
}

data *
consulta_1(info *argp, CLIENT *clnt)
{
	static data clnt_res;

	memset((char *)&clnt_res, 0, sizeof(clnt_res));
	if (clnt_call (clnt, CONSULTA,
		(xdrproc_t) xdr_info, (caddr_t) argp,
		(xdrproc_t) xdr_data, (caddr_t) &clnt_res,
		TIMEOUT) != RPC_SUCCESS) {
		return (NULL);
	}
	return (&clnt_res);
}
```

```
data *
getbuffer_1(info *argp, CLIENT *clnt)
{
        static data clnt_res;

        memset((char *)&clnt_res, 0, sizeof(clnt_res));
        if (clnt_call (clnt, GETBUFFER,
                (xdrproc_t) xdr_info, (caddr_t) argp,
                (xdrproc_t) xdr_data, (caddr_t) &clnt_res,
                TIMEOUT) != RPC_SUCCESS) {
                return (NULL);
        }
        return (&clnt_res);
}
```

```
/*
  Archivo “cmysql_xdr.c”
 * Please do not edit this file.
 * It was generated using rpcgen.
 */

#include "cmysql.h"

bool_t
xdr_date (XDR *xdrs, date *objp)
{
	register int32_t *buf;

	 if (!xdr_short (xdrs, &objp->dia))
		return FALSE;
	 if (!xdr_short (xdrs, &objp->mes))
		return FALSE;
	 if (!xdr_u_int (xdrs, &objp->ano))
		return FALSE;
	return TRUE;
}

bool_t
xdr_time (XDR *xdrs, time *objp)
{
	register int32_t *buf;

	 if (!xdr_short (xdrs, &objp->hora))
		return FALSE;
	 if (!xdr_short (xdrs, &objp->min))
		return FALSE;
	return TRUE;
}

bool_t
xdr_dato (XDR *xdrs, dato *objp)
{
	register int32_t *buf;

	 if (!xdr_date (xdrs, &objp->fecha))
		return FALSE;
	 if (!xdr_time (xdrs, &objp->hora))
		return FALSE;
	 if (!xdr_pointer (xdrs, (char **)&objp->valor, sizeof (float), (xdrproc_t) xdr_float))
		return FALSE;
	return TRUE;
}

bool_t
xdr_vec (XDR *xdrs, vec *objp)
{
	register int32_t *buf;

	 if (!xdr_pointer (xdrs, (char **)objp, sizeof (dato), (xdrproc_t) xdr_dato))
		return FALSE;
	return TRUE;
```

```
}

bool_t
xdr_cadena (XDR *xdrs, cadena *objp)
{
        register int32_t *buf;

         if (!xdr_pointer (xdrs, (char **)objp, sizeof (char), (xdrproc_t) xdr_char))
                 return FALSE;
        return TRUE;
}

bool_t
xdr_data (XDR *xdrs, data *objp)
{
        register int32_t *buf;

        int i;
         if (!xdr_vector (xdrs, (char *)objp->tag_ID, 20,
                sizeof (char), (xdrproc_t) xdr_char))
                 return FALSE;
         if (!xdr_short (xdrs, &objp->num_total_paquete))
                 return FALSE;
         if (!xdr_short (xdrs, &objp->num_paquete))
                 return FALSE;
         if (!xdr_pointer (xdrs, (char **)&objp->nombres, sizeof (cadena), (xdrproc_t) xdr_cadena))
                 return FALSE;
         if (!xdr_short (xdrs, &objp->num_proceso))
                 return FALSE;
         if (!xdr_pointer (xdrs, (char **)&objp->datos, sizeof (dato), (xdrproc_t) xdr_dato))
                 return FALSE;
         if (!xdr_short (xdrs, &objp->err))
                 return FALSE;
         if (!xdr_short (xdrs, &objp->tam_buffer))
                 return FALSE;
         if (!xdr_short (xdrs, &objp->num_campos))
                 return FALSE;
        return TRUE;
}

bool_t
xdr_info (XDR *xdrs, info *objp)
{
        register int32_t *buf;

         if (!xdr_pointer (xdrs, (char **)&objp->Host, sizeof (char), (xdrproc_t) xdr_char))
                 return FALSE;
         if (!xdr_pointer (xdrs, (char **)&objp->User, sizeof (char), (xdrproc_t) xdr_char))
                 return FALSE;
         if (!xdr_pointer (xdrs, (char **)&objp->Password, sizeof (char), (xdrproc_t) xdr_char))
                 return FALSE;
         if (!xdr_pointer (xdrs, (char **)&objp->BD, sizeof (char), (xdrproc_t) xdr_char))
                 return FALSE;
         if (!xdr_pointer (xdrs, (char **)&objp->Tabla, sizeof (char), (xdrproc_t) xdr_char))
                 return FALSE;
         if (!xdr_pointer (xdrs, (char **)&objp->Query, sizeof (char), (xdrproc_t) xdr_char))
```

```
                return FALSE;
        if (!xdr_short (xdrs, &objp->num_paquete))
                return FALSE;
        if (!xdr_short (xdrs, &objp->num_proceso))
                return FALSE;
        if (!xdr_short (xdrs, &objp->tam_buffer))
                return FALSE;
       return TRUE;
}
```