



UNIVERSIDAD DEL AZUAY

**FACULTAD DE CIENCIAS DE LA
ADMINISTRACIÓN**

ESCUELA DE INGENIERÍA DE SISTEMAS

**“INTERFASE DE COMUNICACIONES PARA CENTRALILLAS
TELEFÓNICAS AVAYA E80 CONTROLLER CON UNA PC,
DESARROLLADO EN JAVA”**

**MONOGRAFÍA PREVIA A LA OBTENCIÓN DEL
TÍTULO DE INGENIERÍA DE SISTEMAS**

AUTORES:

**ROBERTO FELIPE ARÍZAGA NÚÑEZ
JORGE ENRIQUE ARREAGA ORDÓÑEZ**

DIRECTOR:

ING. FABIÁN CARVAJAL

**CUENCA, ECUADOR
2006**

DEDICATORIA

Dedico la culminación de esta monografía
a mis queridos padres, hermanos y sobrinos,
que día a día son la fortaleza de vida
que me impulsa a seguir adelante en la lucha
por conseguir mis anhelos y proyectos.

Felipe Arízaga Núñez

Dedico todo este esfuerzo a mi familia,
ya que sin el apoyo de ellos, no hubiese podido
alcanzar esta meta tan anhelada
que es una gran satisfacción
para mi vida personal y profesional

Jorge Arreaga Ordóñez

AGRADECIMIENTOS

Agradezco a mi Amado Dios
por brindarme todos los medios necesarios para culminar
satisfactoriamente una etapa más de mi vida,
y a todos mis familiares y amigos que de una u otra manera
han estado a mi lado animándome a seguir adelante.

Felipe Arízaga Núñez

Mis más sinceros agradecimientos a la Universidad del Azuay,
institución que le lleno de sabiduría a mi vida,
tanto profesional como personal.
A toda mi familia, ya que con paciencia
y colaboración, estuvieron siempre a mi lado.
Y a todos mis compañeros quienes me colaboraron
para la resolución de todas mis inquietudes.

Muchas Gracias
Jorge Arreaga Ordóñez

ÍNDICE DE CONTENIDOS

Dedicatoria	ii
Agradecimientos	iii
Índice de Contenidos	iv
Índice de Ilustraciones y Cuadros	vii
Resumen	viii
Abstract	ix
Introducción	1
Capítulo I: La Comunicación Serial	2
1.1. Introducción	2
1.2. Características	2
1.3. El Protocolo RS-232.....	5
1.3.1. Señales del Protocolo RS-232	5
1.3.2. Componentes Físicos	7
1.3.2.1. Tipos de Cables	7
1.3.2.2. Conectores DB9	10
1.3.2.3. Conectores RJ45	11
1.4 Conclusiones	12
Capítulo II: La Centralilla Telefónica Avaya E80 Controller	13
2.1. Introducción	13
2.2. Generalidades	14
2.3. Módulos del Sistema	14
2.4. Capacidad del Sistema	15
2.5. Modos de Funcionamiento	16
2.5.1. Modo Principal	17
2.5.2. Modo Híbrido	17
2.6. Reporte de Llamadas	19
2.6.1. Registro por Llamada	20

2.6.2.	Parámetros de Comunicación	22
2.7.	Conclusiones	22
Capitulo III: Lenguaje de Programación Java		23
3.1.	Introducción	23
3.2.	Objetivos del Diseño de Java	23
3.3.	Características de Java	24
3.4.	Bibliotecas de Clases Java	25
3.4.1.	API de Comunicaciones	25
3.4.1.1.	Instalación	26
3.4.1.2.	Características	26
3.4.1.3.	Clases y Métodos Java para Control de Puertos de Comunicación	27
3.4.2.	El API JDBC	35
3.4.3.	Cargando el controlador JDBC	35
3.4.4.	Establecer la Conexión	37
3.4.5.	Creación de Sentencias	38
3.4.6.	Ejecución de Consultas	39
3.5.	Conclusiones	40
Capitulo IV: El Gestor de Base de Datos MySQL		41
4.1	Introducción	41
4.2.	Características	41
4.3.	Conectarse al Servidor MySQL	42
4.4.	Comandos Básicos de MySQL	43
4.5.	Crear y Usar Una Base de Datos	46
4.6.	Crear Tablas	47
4.7.	Agregar, Modificar y Eliminar Registros de Una Tabla	49
4.8.	Recuperar y Ordenar Información de Una Tabla	50
4.9	Conclusiones	54
Capitulo V: Análisis y Diseño del Proyecto		55
5.1.	Introducción	55

5.2.	Ambiente de Desarrollo del Proyecto	55
5.3.	Definición de Entidades y sus Atributos	55
5.4.	Modelo Entidad–Relación	57
5.5.	Construcción del Cableado	58
5.6.	Programación de la Interfase	59
5.7.	Programación de las Consultas	61
5.8.	Conclusiones	67
Capitulo VI: Conclusiones y Recomendaciones		68
Bibliografía		69

INDICE DE ILUSTRACIONES

Figura 1	Ejemplo de Transmisión de Datos Binarios	4
Figura 2	Señales del Protocolo RS–232	6
Figura 3	Conector DB9	10
Figura 4	Pinaje del Conector DB9	10
Figura 5	Conector RJ45	12
Figura 6	Código de Colores para Conectores RJ45 – Estándar EIA/TIA 568	12
Figura 7	Centralilla Telefónica Avaya	13
Figura 8	Módulos Instalados en una Centralilla Avaya E80 Controller en Funcionamiento	16
Figura 9	Componente SMDR del Procesador ACS	20
Figura 10	Ejemplo del Reporte de Llamadas	21
Figura 11	Modelo Entidad–Relación	57
Figura 12	Armado de Conectores RJ45 y DB9	58
Figura 13	Pantalla de Visualización del Programa Captura	59
Figura 14	Pantalla del Programa Reporte	62
Tabla 1	Descripción del Pinaje del Conector DB9	11
Tabla 2	Capacidad de la Centralilla Telefónica de Acuerdo a los Módulos Instalados	15
Tabla 3	Pinaje de Conectores DB25 en Modo SPP	32
Tabla 4	Métodos para ejecutar sentencias SQL	39
Tabla 5	Prompts de MySQL	45
Tabla 6	Atributos de la Tabla Llamada	56
Tabla 7	Atributos de la Tabla Extensión	56
Tabla 8	Configuración del Pinaje entre Conectores RJ45 y DB9	58

RESUMEN

El desarrollo del proyecto consiste en la recepción de información de todas las llamadas entrantes y salientes que provee el dispositivo SMDR instalado en el Módulo Procesador ACS de la centralilla telefónica AVAYA E80 CONTROLLER, obtener el registro de cada llamada, dividirlo y asignar la información correspondiente a cada campo de acuerdo a la tabla de la base de datos creada para almacenar el proyecto.

Por medio de un cable construido siguiendo las reglas del protocolo RS-232 y los estándares de la norma ANSI/TIA/EIA-568B, instalado entre la centralilla telefónica y el puerto serial de un computador se logrará la comunicación necesaria para la obtención de los datos. Una aplicación realizada en el Lenguaje de Programación Java, configurará la comunicación e interpretará lo que la centralilla telefónica envía, para luego conectarse al Gestor de Base de Datos MySQL y desde la aplicación java, procesar, grabar y finalmente consultar la información de las tablas de la Base de Datos.

ABSTRACT

The project's function consists of receiving information on all incoming and outgoing calls provided by the SMDR device installed in the ACS Processor Module of the AVAYA E80 CONTROLLER, which is a telephone exchange device; obtain the record of each call, divide it and assign to it the information corresponding to each field, according to the data base created to store the project.

By means of a cable constructed to follow the RS-232 protocol rules and the standards of the ANSI/TIA/EIA-568B norm, installed in the telephone exchange device and the serial port of a computer, the necessary communication will be established to obtain the data. An application created in Java Programming Language will configure the communication and interpret what the telephone exchange device sends, then connect to the MySQL Data Base Manager, and from the Java application, process, record and finally consult the information in the Data Base tables.

INTRODUCCIÓN

La conexión entre diferentes tipos de dispositivos es una tarea que se realiza utilizando normas o estándares de comunicación. En la informática estas normas son las que regulan la transmisión y recepción de datos, utilizando medios físicos para trasladar señales que se coordinan desde una aplicación programada con los parámetros sincronizados de cada uno de los dispositivos involucrados. Al tener totalmente establecidos estos parámetros, se puede desarrollar el proyecto, determinando el origen de los datos (centralilla telefónica), su medio físico de transporte (cableado y configuración de conectores), el lugar de almacenamiento de la información (computador) y la aplicación que será la interfase con el usuario final y el medio que combinará todos los elementos.

Al tener un programa que gestione la recepción de los datos desde una centralilla telefónica hacia un computador, tendríamos la capacidad de manejar dicha información de manera que permita al usuario final tomar decisiones rápidas valiéndose de datos confiables mediante consultas y reportes ordenados de acuerdo a su criterio o necesidad.

El diseño de aplicaciones informáticas en la actualidad, tiende al uso de programas open source para su creación, debido a que estas herramientas, a más de ser una opción económica, tienen gran cantidad de documentación de soporte para las personas que decidan utilizarlas. Las características antes expuestas y la capacidad de manejar comunicaciones seriales y almacenamiento de grandes cantidades de información, han sido las razones que han motivado a utilizar dentro del desarrollo de este proyecto, el Lenguaje de Programación Java y el Gestor de Base de Datos MySQL.

CAPITULO I

LA COMUNICACIÓN SERIAL

1.1. Introducción

La información que maneja un computador puede transmitirse de un lugar a otro en dos formas básicas, en forma serial o en forma paralela. En una transmisión serial se forma un flujo de bits, uno tras de otro viajan del lugar de emisión al de recepción, utilizando un conductor eléctrico. La transmisión tiene una sola dirección y si se desea realizar en el sentido contrario, se debe esperar a que la vía esté libre. En la comunicación en paralelo cada bit tiene su vía exclusiva, con la condición de que todos viajen simultáneamente, como en el caso de la comunicación serial para transmitir en el sentido contrario se debe esperar que la vía este libre, a menos que se tenga una exclusiva para el sentido contrario.

1.2. Características

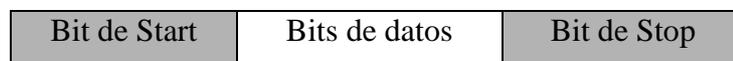
El mundo de las comunicaciones internas del computador se realiza en forma paralela alternada, por fuera del computador predominan las comunicaciones seriales; las redes de computadores se basan en dicha comunicación. La información en una cadena serial de bits está contenida en su forma de onda dependiente del tiempo: los bits se representan por códigos que se transmiten por un periodo de tiempo fijo. El periodo de tiempo usado para transmitir cada código se conoce como periodo *baud*.

En un computador físicamente tenemos el primer puerto serial denominado comúnmente COM1, este tiene asignada la interrupción IRQ4 y sus registros empiezan en la dirección de la memoria %3F8, y de ahí en adelante hasta la %3FE. Para las máquinas que tienen un segundo puerto serial este se denomina COM2, tiene asignada la interrupción IRQ3 y sus registros se alojan en las direcciones %2F8 hasta la %2FE. Los puertos denominados COM3 y COM4 a pesar de que se mapean en un espacio diferente de los puertos anteriores, comparten las interrupciones, COM1 con

COM3 y COM2 con COM4, por esto es muy difícil utilizar los cuatro cuando se trata de hacerlos funcionar mediante interrupciones.

La comunicación serial, como su nombre lo indica, realiza la transferencia de información enviando o recibiendo datos descompuestos en bits, los cuales viajan secuencialmente uno tras otro. Las cadenas seriales de bits generadas por los puertos serie de un computador usan una forma muy simple de codificación. Un bit se transmite durante cada periodo baud, con un bit “1” representado por un voltaje alto TTL y un “0” por un voltaje bajo TTL. Así la velocidad en baudios (baud rate, $1/[\text{periodo baud}]$) de un puerto serie de un computador es igual al número de bits por segundo que se transmiten o reciben.

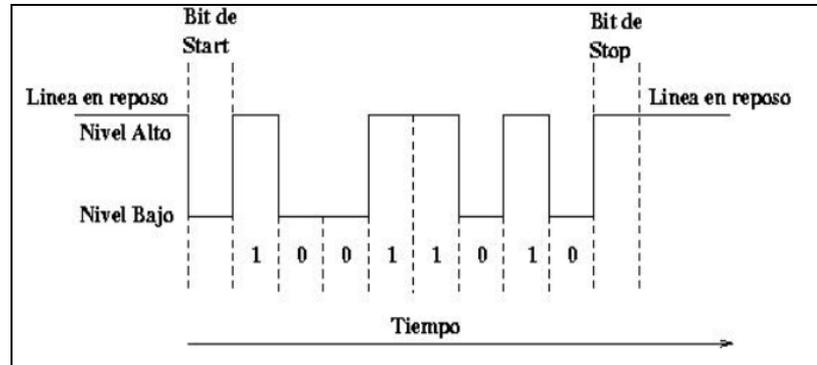
Para enviar información codificada de esta manera, el transmisor y receptor registran el tiempo, el cual define el periodo baud, deben estar a la misma frecuencia y estar sincronizados. Los bits se transmiten como grupos separados, con una longitud típica de 7 u 8 bits, llamados *caracteres*. El nombre carácter se usa porque cada grupo de bits representan una letra del alfabeto cuando el texto está codificado en ASCII. Cada carácter se envía en una armazón (frame) consistiendo de un bit “0” llamado un *bit de inicio*, seguido por el carácter mismo, seguido (opcionalmente) por un bit de paridad, y después un bit “1” llamado *bit de paro*. La lógica del bit bajo de inicio le dice al receptor que está empezando una armazón, y la lógica del bit alto de paro denota el final de la armazón. Los datos serie se encuentran encapsulados en tramas de la siguiente forma:



Primero se envía un bit de start, a continuación los bits de datos (primero el bit de mayor peso) y finalmente los bits de STOP. El número de bits de datos y de bits de stop es uno de los parámetros configurables, así como el criterio de paridad par o impar para la detección de errores. Normalmente, las comunicaciones serie tienen los siguientes parámetros: 1 bit de Start, 8 bits de Datos, 1 bit de Stop y sin Paridad.

En la figura 1, se puede observar un ejemplo de la transmisión del dato binario 10011010. La línea en reposo está a nivel alto:

Figura 1 Ejemplo de Transmisión de Datos Binarios



Se llama comunicación serial asíncrona porque el receptor se resincroniza el mismo con el transmisor usando el bit de inicio de cada armazón. Los caracteres se pueden transmitir en cualquier tiempo, con un retraso de tiempo arbitrario entre caracteres. Existen también protocolos de comunicación serial síncrona donde los caracteres se envían en bloques sin una armazón de bits circundante. En ésta aproximación, el transmisor continuamente transmite señales, con un carácter de sincronización especial que se transmite si no hay datos reales disponibles para transmitir. Los bits dentro de cada carácter transmitido se envían con el bit menos significativo primero, cada bit durando un periodo baud. Los transmisores y receptores seriales se pueden instruir para enviar o recibir de 5 a 8 bits por carácter (ambos deben de estar de acuerdo en cuantos).

Después de que los bits de cada carácter se envían, puede seguir un bit de paridad opcional. El bit de paridad es útil si la línea de datos está muy ruidosa como para proporcionar una transmisión fiel. El bit de paridad, P, se puede elegir para dar ya sea paridad par o impar. Para paridad par, $P = 1$ si el número de 1's en el carácter es impar y $P = 0$ si el número es par. Es decir, en la paridad par P se elige tal que el número de 1's incluyendo P es par. Para paridad impar, P se elige tal que el número de 1's incluyendo P es impar. El receptor local revisa para asegurar que la paridad es aun la misma a pesar de que el cable haya recogido ruido. Si la paridad ha cambiado, algún bit se ha perdido, y el receptor pone una bandera de error de paridad en el registro de estado.

La comunicación serial está compuesta principalmente de dos elementos básicos, el hardware, que hace referencia a la configuración de los conectores y niveles de voltaje y el software con el que se controla la información binaria que se quiere transferir. Todo esto está regido por normas o protocolos donde el utilizado por las computadoras convencionales es el protocolo RS-232.

1.3. El Protocolo RS-232

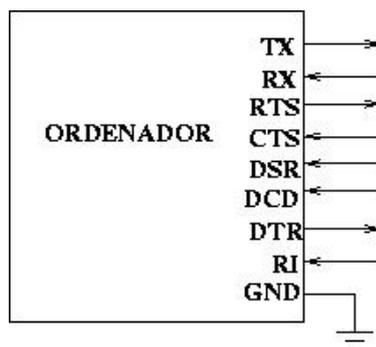
Los sistemas de comunicación serie responden a un conjunto de características fijadas por varios institutos de normalización (ISO, EIA, CCITT, etc.) que describen detalladamente las especificaciones de cada sistema. La más extendida de las normas utilizadas es la EIA RS-232 (Recommended Standard 232). Por medio de este protocolo se estandarizan las velocidades de transferencia de datos, la forma de control que utiliza dicha transferencia, los niveles de voltajes utilizados, el tipo de cable permitido, las distancias entre equipos, los conectores, etc., entre un terminal informático (DTE Data Terminal Equipment) y un equipo de comunicaciones como podría ser, por ejemplo, un módem (DCE Data Communications Equipment).

El protocolo RS-232 fue definido para conectar un ordenador a un módem y es básicamente la selección de la velocidad en baudios (1200, 2400, 4800, etc.), la verificación de datos o paridad (paridad par o paridad impar o sin paridad), los bits de parada luego de cada dato (1 ó 2), y la cantidad de bits por dato (7 ó 8), que se utiliza para cada símbolo o carácter. Además de transmitir (línea Tx) y recibir (línea Rx) los datos en una forma serie asíncrona, son necesarias otras líneas de control de flujo (*Hands-hake*) con tensiones comprendidas entre +15/-15 voltios, donde su uso es opcional dependiendo del dispositivo a conectar.

1.3.1. Señales del Protocolo RS-232

Las señales utilizadas en este protocolo, se describen en la Figura 2:

Figura 2 Señales del Protocolo RS-232



Request To Send (RTS).- Se pone a nivel alto para indicar al corresponsal que se desea transmitir. Esta señal se envía de la computadora (DTE) al módem (DCE) para indicar que se quieren transmitir datos. Si el módem decide que esta OK, asiente por la línea CTS. Una vez que la computadora prende la señal RTS, esperará que el módem asiente la línea CTS. Cuando la señal CTS es afirmado por el módem, la computadora empezará a transmitir datos.

Clear To Send (CTS).- El corresponsal indica que está listo para recibir datos poniendo esta señal a nivel alto. Afirmado por el módem después de recibir la señal de RTS indica que la computadora puede transmitir.

Data Terminal Ready (DTR).- Si el computador está conectado a un módem detecta la presencia de portadora en la línea. Esta línea de señal es afirmada por la computadora, e informa al módem que la computadora está lista para recibir datos.

Data Set Ready (DSR).- El corresponsal indica que está conectado. Esta línea de señal es afirmada por el módem en respuesta a una señal DTR de la computadora. La computadora supervisa el estado de ésta línea después de afirmar DTR para descubrir si el módem está encendido.

Receive Signal Line Detect (RSLD).- Indica que hay portadora en la línea (en un módem indicaría que la comunicación telefónica está establecida, aunque no se transmitan datos). Esta línea de control es afirmada por el módem e informa al computador que se ha establecido una conexión física con otro módem. A veces se conoce como detector de portadora (Carrier Detect CD). Sería un error que un

computador transmita información a un módem si ésta línea no está encendida, es decir si la conexión física no funciona.

Transmit Data (TD).- Es la línea a través de la cual los datos salen del computador. Se transmite un bit a la vez.

Receive Data (RD).- Es la línea a través de la cual los datos entran al computador. Se recibe un bit a la vez.

Ring Indicator (RI).- Detección de llamada. Cuando el computador está conectado a un módem y suena el teléfono, el módem activa esta señal.

Ground (GND).- Línea de masa de referencia (0 V).

1.3.2. Componentes Físicos

1.3.2.1. Tipos de Cables

En la actualidad existen básicamente tres tipos de cables factibles de ser utilizados para el cableado en el interior de edificios o entre edificios:

- Coaxial
- Par Trenzado (2 pares y 4 pares)
- Fibra Óptica

El Par Trenzado (2 y 4 pares) y la Fibra Óptica son reconocidos por la norma ANSI/TIA/EIA-568 y el Coaxial se acepta pero no se recomienda en instalaciones nuevas. Para el desarrollo del proyecto se usará el Par Trenzado de 4 pares, estándar TIA/EIA-568B.

Par Trenzado.- Es el tipo de cable más común y se originó como solución para conectar teléfonos, terminales y ordenadores sobre el mismo cableado, ya que está habilitado para comunicación de datos permitiendo frecuencias más altas de

transmisión. Cada cable de este tipo está compuesto por una serie de pares de cables trenzados. Los pares se trenzan para reducir la interferencia entre pares adyacentes. Normalmente una serie de pares se agrupan en una única funda de color codificado para reducir el número de cables físicos que se introducen en un conducto. El número de pares por cable son 4, 25, 50, 100, 200 y 300. Cuando el número de pares es superior a 4 se habla de cables multipar.

Los tipos de cable par trenzado, son los siguientes:

- **No Blindado**.- Es el cable de par trenzado normal y se le referencia por sus siglas en inglés UTP (Unshield Twisted Pair; Par Trenzado no Blindado). Las mayores ventajas de este tipo de cable son su bajo costo y su facilidad de manejo. Sus mayores desventajas son su mayor tasa de error respecto a otros tipos de cable, así como sus limitaciones para trabajar a distancias elevadas sin regeneración.

Para las distintas tecnologías de red local, el cable de pares de cobre no blindado se ha convertido en el sistema de cableado más ampliamente utilizado. El estándar EIA-568 en el adendum TSB-36 diferencia tres categorías distintas para este tipo de cables.

- Categoría 3: Admiten frecuencias de hasta 16 Mhz.
- Categoría 4: Admiten frecuencias de hasta 20 Mhz.
- Categoría 5: Admiten frecuencias de hasta 100 Mhz.

Las características generales del cable no blindado son:

- **Tamaño**: El menor diámetro de los cables de par trenzado no blindado permite aprovechar más eficientemente las canalizaciones y los armarios de distribución. El diámetro típico de estos cables es de 0,52 cm.
- **Peso**: El poco peso de este tipo de cable con respecto a los otros tipos de cable facilita el tendido.

- **Flexibilidad:** La facilidad para curvar y doblar este tipo de cables permite un tendido más rápido así como el conexionado de las rosetas y las regletas.
- **Instalación:** Debido a la amplia difusión de este tipo de cables, existen una gran variedad de suministradores, instaladores y herramientas que abaratan la instalación y puesta en marcha.
- **Integración:** Los servicios soportados por este tipo de cable incluyen:
 - Red de Area Local ISO 8802.3 (Ethernet) e ISO 8802.5 (Token Ring)
 - Telefonía analógica
 - Telefonía digital
 - Terminales síncronos
 - Terminales asíncronos
 - Líneas de control y alarmas
- **Blindado.-** Cada par se cubre con una malla metálica, de la misma forma que los cables coaxiales, y el conjunto de pares se recubre con una lámina blindada. Se referencia frecuentemente con sus siglas en inglés STP (Shield Twisted Pair, Par Trenzado blindado).

La malla del cable (tejido de cobre que envuelve a los cablecillos metálicos finos) debe conectarse a la carcasa del conector. El empleo de una malla blindada reduce la tasa de error, pero incrementa el coste al requerirse un proceso de fabricación más costoso. La conexión de la malla establece el mismo nivel de potencial (voltaje) entre los 2 computadores y evita que grandes cargas de electricidad estática fluyan a través de las líneas y fundan el chip del puerto serie. Además, se produce un efecto de apantallamiento, que evita que señales electromagnéticas interfieran en la información que circula por la línea.

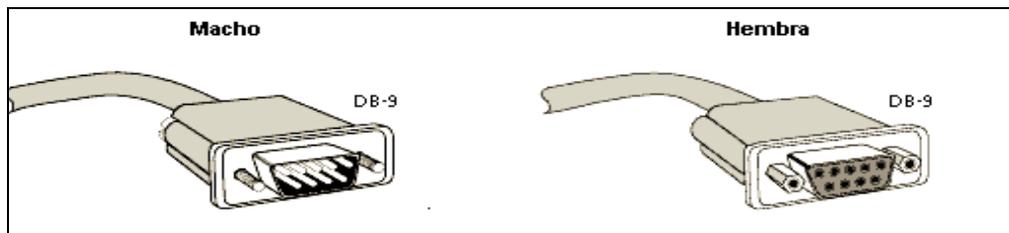
- **Uniforme.-** Cada uno de los pares es trenzado uniformemente durante su creación. Esto elimina la mayoría de las interferencias entre cables y además protege al conjunto de los cables de interferencias exteriores. Se realiza un blindaje global de todos los pares mediante una lámina externa blindada. Esta

técnica permite tener características similares al cable blindado con unos costes por metro ligeramente inferior.

1.3.2.2. Conectores DB9

El puerto físico de conexión de un computador, puede tener el zócalo DB9 (conector de 9 pines), o el DB25 (conector de 25 pines), por el que se conectan los dispositivos al puerto serie. En estos conectores se utilizan tres pines para alcanzar la comunicación de los equipos. El pin TXD, se usa para transmitir bits, el pin RXD, para recibir y el pin GND de masa o referencia. Cuando no hay transmisión de datos el pin TXD, se indica con el estado *mark*, con un uno lógico continuo o $-10v$ continuos. Por razones de desarrollo de nuestro proyecto, profundizaremos en el estudio del conector DB9.

Figura 3 Conector DB9



Los conectores hembra se enchufan al conector macho en el computador, estos tienen una colocación de pines diferente, de manera que al conectarse coinciden el pin 1 del macho con el pin 1 del hembra, el pin2 con el 2, etc... Cada uno de estos pines tiene asignado una señal de control de la norma RS-232 como se visualiza en la Figura 4 y en la Tabla 1:

Figura 4 Pinaje del Conector DB9

5 4 3 2 1	1 NC	1 NC	DCD
○ ○ ○ ○ ○	2	2	RXD
○ ○ ○ ○ ○	3	3	TXD
9 8 7 6	4 NC	4 NC	DTR
Conector DB-9	5	5	SIG GND
Hembra	6 NC	6 NC	DSR
Vista Frontal	7 NC	7 NC	RTS
	8 NC	8 NC	CTS
	9 NC	9 NC	RI
	Shield	Shield	

Tabla 1 Descripción del Pinaje del Conector DB9

Pin	Señal	E/S	Definición
1	DCD	E	Detección de portadora de datos
2	RX	E	Entrada serie
3	TX	S	Salida serie
4	DTR	S	Terminal de datos lista
5	GND	N/D	Tierra de señal
6	DSR	E	Grupo de datos listo
7	RTS	S	Petición para enviar
8	CTS	E	Listo para enviar
9	RI	E	Indicador de llamada
Casquete	N/D	N/D	Conexión a tierra del chasis

Las comprobaciones de las señales que se realizan durante la transmisión de un carácter se describen a continuación:

1. Ponemos el DTR, indicando que estamos preparados.
2. Ponemos RTS, solicitando permiso para emitir.
3. Comprueba que está DSR, verificamos si el destinatario está listo.
4. Esperamos a CTS, a que se autorice a enviar datos.
5. Enviamos datos.

1.3.2.3. Conectores RJ45

Son conectores de 8 hilos, también llamados Giant Modular Jack, están diseñados para conectar un cable UTP (Unshielded Twisted Pair [par Trenzado sin Blindaje]) para red Ethernet equipado con enchufes convencionales compatibles con el estándar RJ45. Se coloca, presionando un extremo del cable UTP dentro del conector NIC hasta que el enchufe se asiente en su lugar. Luego se conecta el otro extremo del cable a una placa de pared con enchufe RJ45 o a un puerto RJ45 en un concentrador o central UTP, dependiendo de la configuración de su red. También son usados para conectar dispositivos tipo DB9, por ejemplo: Cable Categoría 5, dispositivos seriales, equipo de telecomunicaciones, etc.

Figura 5 Conector RJ45

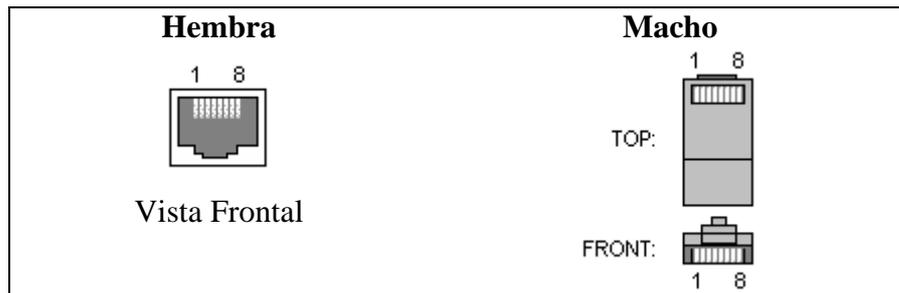
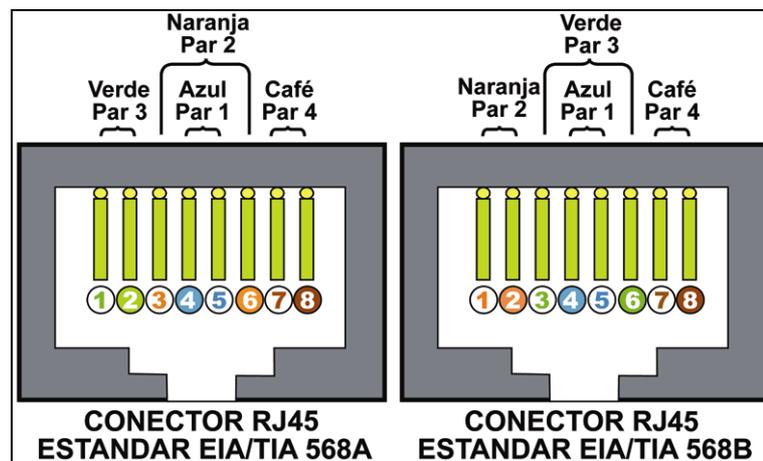


Figura 6 Código de Colores para Conectores RJ45 – Estándar EIA/TIA 568



1.4 Conclusiones

Las normas o especificaciones del protocolo RS-232 creadas para manejar la comunicación serial permiten controlar el envío o la recepción de la información de una manera segura. Analizando los dispositivos involucrados, podemos determinar los tipos de conectores que estos utilizan, para desde y hacia ellos configurar las señales de control de flujo a ser transportadas por cada uno de los hilos del cable de tipo par trenzado. Respetando las reglas descritas por este protocolo podemos lograr la recepción o transmisión de datos desde la centralilla telefónica y un computador, que es uno de los objetivos del proyecto.

CAPITULO II

LA CENTRALILLA TELEFÓNICA AVAYA E80 CONTROLLER

Figura 7 Centralilla Telefónica Avaya



2.1. Introducción

Actualmente en el mundo de la negocios, es una necesidad imprescindible el contar con un equipo adecuado para controlar el flujo de las comunicaciones personales, tanto internas como externas. El facilitar a los empleados de una empresa la comunicación entre ellos sin tener que pagar un costo adicional por este servicio, y a la vez permitirles acceder a cada uno de ellos desde un solo teléfono (extensión) a varias líneas de salida, son las ventajas principales que ofrece el utilizar una centralilla telefónica.

2.2. Generalidades

La centralilla telefónica AVAYA E80 Controller es un Sistema de Comunicaciones Avanzado (ACS por sus siglas en inglés). Este sistema de comunicaciones dinámico, tiene un manejo de llamadas intuitivo combinado con una variedad de características que hacen al sistema eficiente y flexible. Puede operar en dos modos: Principal e Híbrido, y también como parte de un sistema Centrex.

Este sistema soporta teléfonos conmutadores, con pantallas que permiten visualizar las operaciones de programación o reprogramación. Incluye soporte para teléfonos de línea simple, y la posibilidad de conectar al sistema una variedad de dispositivos auxiliares como: faxes, contestador de llamadas, módems, lectores de tarjetas de crédito, etc.

2.3. Módulos del Sistema

Los módulos descritos a continuación pueden ser instalados en la centralilla telefónica AVAYA E80 Controller:

Módulo Procesador PARTNER ACS.- Proporciona mediante un software inteligente el control de las características de la centralilla. Tiene 3 líneas de salida y 8 extensiones, música de espera, conexión a tierra y un dispositivo de reportes de llamadas (SMDR).

Módulo 200.- Tiene 2 líneas de salida y ninguna extensión. Es una opción económica para agregar líneas cuando no se necesitan más extensiones.

Módulo 206E.- Tiene un máximo de 2 líneas de salida y 6 extensiones. Se pueden conectar teléfonos y otros dispositivos de telecomunicaciones como faxes y módems.

Módulo 406E.- Es similar al 206E, pero no tiene extensiones, tiene 4 líneas de salida. Como el módulo 200, es una opción económica para agregar líneas cuando no se necesita más extensiones.

Módulos 206EC/400EC.- Proporcionan las mismas capacidades que los Módulos 206E y 400E, pero agregan la capacidad de identificación de llamadas.

Módulo de Expansión 308EC.- Proporciona la expansión de líneas y extensiones. Tiene 3 líneas de salida y 8 extensiones.

De acuerdo a los módulos que se instalen, la centralilla telefónica puede tener una de las tres siguientes configuraciones básicas:

- Autosuficiente con un solo Módulo. Permite instalar únicamente el Procesador PARTNER ACS.
- Una portadora con dos slots. Permite instalar hasta dos módulos, el procesador se instala a la izquierda.
- Una portadora con cinco slots. Permite instalar hasta cinco módulos, el procesador se instala en el centro.

2.4. Capacidad del Sistema

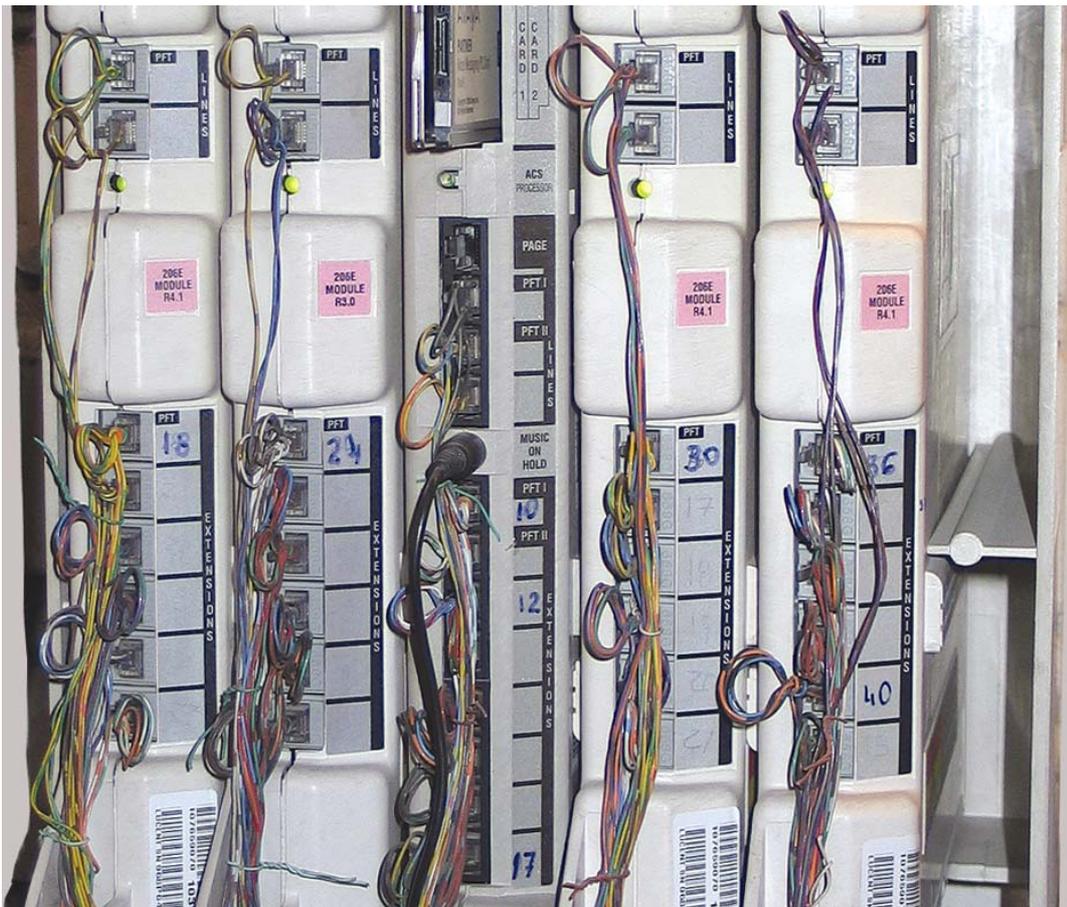
De acuerdo a la portadora que se use, y el número de módulos que se instalen, la combinación de éstos determinarán el número de líneas y extensiones disponibles:

Tabla 2 Capacidad de la Centralilla Telefónica de Acuerdo a los Módulos Instalados

Configuración	Máximo de Líneas	Máximo de Extensiones
Autosuficiente Con un Módulo	1 Procesador PARTNER ACS Total: 3 líneas y 8 extensiones	1 Procesador PARTNER ACS Total: 8 extensiones y 3 líneas
Una portadora Con dos slots	1 Procesador PARTNER ACS 1 Módulo 400 Total: 7 líneas y 8 extensiones	1 Procesador PARTNER ACS 1 Módulo 308EC Total: 16 extensiones y 6 líneas
Una portadora Con cinco slots	1 Procesador PARTNER ACS 4 Módulos 400 Total: 19 líneas y 8 extensiones	1 Procesador PARTNER ACS 4 Módulos 308EC Total: 40 extensiones y 15 líneas

En la Figura 7 se pueden ver los módulos instalados en una centralilla telefónica Avaya Controller E80 en funcionamiento. En esta figura observamos al Procesador ACS instalado en el slot número 3 de la centralilla. En los slots 1, 2, 4 y 5 están instalados cuatro módulos 206E. El resultado de esta configuración, nos permite tener 9 líneas de salida (3 por el Procesador ACS y 2 por cada módulo 206E) y 32 extensiones (8 por el Procesador ACS y 6 por cada módulo 206E).

Figura 8 Módulos Instalados en una Centralilla Avaya E80 Controller en Funcionamiento



2.5. Modos de Funcionamiento

El modo de funcionamiento determina cómo los usuarios acceden a las líneas de salida desde sus teléfonos o extensiones. El modo de funcionamiento es determinado por la configuración del Módulo Procesador. La centralilla soporta dos modos de funcionamiento: Principal e Híbrido. Por defecto la centralilla está configurada en Modo Principal.

2.5.1. Modo Principal

El Modo Principal permite un acceso individual de los usuarios a las líneas de salida para realizar y recibir llamadas. En extensiones con teléfonos conmutadores, cada línea individual (Línea 1, Línea 2, etc) está representada por un botón. Los usuarios podrán presionar cualquier botón que tenga una línea disponible en su conmutador para realizar una llamada hacia fuera. Los usuarios con teléfonos estándares deben presionar 9 al obtener el tono de marcado para poder realizar una llamada hacia fuera.

El modo principal ayuda a los usuarios de teléfonos conmutadores a determinar fácilmente las llamadas de cada línea, etiquetando cada botón con un único número de línea. Además permite monitorear la actividad de las llamadas, usando las luces indicadoras al lado de cada botón. Las que tienen una línea específica asignada a una extensión, indicarán cuando tengan una llamada entrante, en espera, o cuando la línea esté ocupada.

Al instalar, la centralilla asigna las líneas de salida a los botones de todos los teléfonos conmutadores desde izquierda a derecha, empezando por la fila de abajo, pudiendo posteriormente cambiar este orden, si así se lo deseara. Todas las extensiones de la centralilla configuradas en Modo Principal son conocidas como Extensiones Principales.

2.5.2. Modo Híbrido

El Modo Híbrido ofrece flexibilidad a los usuarios en el acceso a las líneas de salida desde sus teléfonos. Como en el Modo Principal, se pueden asignar líneas individuales a los conmutadores, y además múltiples líneas de salida pueden ser agrupadas en lo que llamaremos pools. La centralilla telefónica configurada en modo híbrido puede tener hasta cuatro pools, incluyendo una Pool Principal y tres Pools Auxiliares. Cada pool es identificada por un Código de Acceso, como por ejemplo: 880, 881, 882, y 883 respectivamente.

Los pools son representados en los teléfonos conmutadores por botones. A diferencia de los botones de línea, los botones de pool dan a los usuarios acceso a múltiples líneas desde un solo botón. Cada pool auxiliar es asociado con un sólo botón de una pool. La pool principal contiene generalmente la mayoría de las líneas de salida, y están asociadas a dos botones. Esta configuración permite al usuario poner una llamada entrante en uno de los botones de la pool principal, o ponerla en espera y tomar otra llamada usando el segundo botón, o el usuario puede establecer una conferencia usando las líneas del pool principal. El pool principal y cada pool auxiliar pueden ser asignadas a una extensión, con un máximo de cinco botones pool.

Los usuarios de teléfonos conmutadores pueden presionar cualquier botón pool disponible en sus teléfonos o ingresar el código de acceso del pool al tener tono de marcado para realizar una llamada externa. Los usuarios de teléfonos estándares deben marcar 9 o ingresar el código de acceso del pool al tono de marcado para acceder a un pool desde sus teléfonos. Después que el usuario presiona un botón pool, o ingresa el código y la centralilla selecciona una línea libre para que el usuario pueda realizar su llamada. Un usuario puede acceder a un pool siempre y cuando haya por lo menos una línea disponible.

El mayor beneficio del Modo Híbrido es que permite a los usuarios con teléfonos conmutadores tener acceso a múltiples líneas de salida y a varios tipos de pools con menos botones. Se pueden usar eficazmente las líneas de salida agrupándolas a las que tienen un tipo o funcionamiento similar. En este modo la extensión 10 opera siempre como una extensión en modo principal. Esto significa que cada línea de salida está asociada con un botón de línea específica de la extensión 10. Todas las otras extensiones pueden configurarse con acceso a sólo líneas, sólo pools, o una combinación de líneas y pools:

- Las extensiones que están en un botón pool, aún si tienen botón de línea individuales, son llamadas extensiones pool.
- Las extensiones que están solo en un botón de línea (incluyendo la extensión 10), son llamadas Extensiones Principales. Las extensiones principales no pueden acceder a los pools.

Si el sistema es configurado en Modo Híbrido, se deberán tener las siguientes consideraciones:

- Una línea puede asignarse sólo a una pool.
- Puede restringirse el acceso de extensiones individuales a pools específicas.
- Pueden asignarse las líneas individuales a una extensión con botones pool con tal de que las líneas no sean parte de cualquier pool.

Al instalar, la centralilla asigna todas las líneas de salida a la Pool Principal y además asigna las dos Pools Principales de izquierda a derecha, empezando por la fila de debajo, en todos los teléfonos conmutadores, excepto en la extensión 10. Si se desea, se pueden quitar algunas de las líneas de la pool principal y crearlas en pools auxiliares. Entonces, se pueden asignar pools y/o líneas individuales sobre una base de extensiones.

2.6. Reporte de Llamadas

La Estación de Almacenamiento del Detalle de Mensajes (SMDR por sus siglas en inglés), es un componente del Procesador ACS, que genera reportes detallados de la actividad de las llamadas, es decir, es un módulo principalmente destinado a la comunicación serial con una impresora, que registra en papel el detalle de las características de todas las llamadas entrantes y salientes que va procesando la centralilla telefónica. La información que se obtiene de este reporte nos permitiría:

- Reducir costos telefónicos, al identificar la necesidad de cambiar los servicios de telecomunicaciones, o al limitar el uso telefónico en una extensión en particular, después de determinar excesos.
- Detectar cualquier llamada no autorizada.

Figura 9 Componente SMDR del Procesador ACS



La centralilla telefónica envía la información para ser impresa después de completada la llamada. Para las llamadas salientes, el sistema registra la información por cada llamada que dure más de 10 segundos desde que se accede a una línea de salida. Para las llamadas entrantes el inicio se considera desde que hayan sido contestadas.

La desventaja de conectar directamente una impresora a este módulo, radica en que la revisión de la información impresa se hace muy complicada, ya que su único criterio de ordenamiento es la hora de terminada la llamada, lo que permitiría únicamente una revisión manual del uso telefónico de un determinado usuario o una extensión.

2.6.1. Registro por Llamada

Los registros por cada llamada es una línea de información que incluyen los siguientes campos:

Figura 10 Ejemplo del Reporte de Llamadas

T	FECHA	HORA	NUMERO	DURACIÓN	LIN.	EXT.	CUENTA
C	04/26/97	11:34	1023319085556036	00:04:28	02	32	1725
I	04/26/97	13:35	IN	00:02:12	01	10	
I	04/26/97	13:38	9085559111	00:01:22	12	15	

Tipo de Llamada.- Este campo tiene únicamente dos valores posibles: la “C” indica llamada saliente y la “I” llamada entrante.

Fecha.- La fecha en que la llamada se realizó. Su formato es mm/dd/aa, que representa mes, día y año.

Hora.- La hora en que se realizó la llamada. Su formato es hh:mm, que representa horas y minutos.

Número.- En las llamadas salientes es el número marcado por la extensión, para las llamadas entrantes se registra “IN”, a menos que la centralilla telefónica tenga instalada un identificador de llamadas. En este campo se pueden almacenar un máximo de 24 dígitos.

Duración.- La duración es el tiempo total de la llamada. Su formato es hh:mm:ss, que representa horas, minutos y segundos.

Número de Línea.- Es la línea de salida que la extensión utilizó para hacer o recibir una llamada.

Extensión.- Esta normalmente es la extensión que contestó o realizó la llamada. Al redireccionar llamadas entrantes, la última extensión en la llamada es la que se registra.

Código de Cuenta.- Es el código asignado a cada uno de los registros y es típicamente usado para totalizar el valor de las llamadas realizadas por un usuario, proyecto o departamento.

2.6.2. Parámetros de Comunicación

Para conseguir la conexión con la centralilla telefónica, utilizando el componente SMDR y otro dispositivo externo (en nuestro caso el puerto serial de un computador) se deben configurar los siguientes parámetros de transmisión usados en el protocolo RS-232:

- Data Bits = 8
- Stop Bits = 2
- Parity = None
- Flow Control = None
- Speed = 1200 bps

2.7. Conclusiones

El estudio de la centralilla telefónica AVAYA E80 Controller, nos permite conocer las características físicas y técnicas de este equipo. Las especificaciones físicas nos dieron como resultado que es posible la comunicación entre la centralilla telefónica con un computador mediante el protocolo RS-232, usando el SMDR que es uno de los componentes del Procesador ACS; y las especificaciones técnicas nos permiten obtener la estructura de los registros con toda la información que proporciona la centralilla, que será almacenada en la computadora.

CAPITULO III

LENGUAJE DE PROGRAMACIÓN JAVA

3.1. Introducción

Java se creó como parte de un proyecto de investigación para el desarrollo de software avanzado para una amplia variedad de dispositivos de red y sistemas embebidos. La meta era diseñar una plataforma operativa sencilla, fiable, portable, distribuida y de tiempo real. Cuando se inició el proyecto, C++ era el lenguaje más utilizado, pero a lo largo del tiempo, las dificultades encontradas con C++ crecieron hasta el punto en que se pensó que los problemas podrían resolverse mejor creando una plataforma de lenguaje completamente nueva. Se extrajeron decisiones de diseño y arquitectura de una amplia variedad de lenguajes como Eiffel, SmallTalk, Objective C y Cedar/Mesa. El resultado es un lenguaje que se ha mostrado ideal para desarrollar aplicaciones de usuario final seguras, distribuidas y basadas en red en un amplio rango de entornos desde los dispositivos de red embebidos hasta los sistemas de sobremesa e Internet.

3.2. Objetivos del Diseño de Java

Java fue diseñado para ser:

Sencillo, Orientado a Objetos y Familiar.- Sencillo, para que no requiera grandes esfuerzos de entrenamiento para los desarrolladores. Orientado a objetos, porque la tecnología de objetos se considera madura y es el enfoque más adecuado para las necesidades de los sistemas distribuidos y/o cliente/servidor. Familiar, porque aunque se rechazó C++, se mantuvo Java lo más parecido posible a C++, eliminando sus complejidades innecesarias, para facilitar la migración al nuevo lenguaje.

Robusto y Seguro.- Robusto, simplificando la gestión de memoria y eliminando las complejidades de la gestión explícita de punteros y aritmética de punteros del C. Seguro para que pueda operar en un entorno de red.

Independiente de la Arquitectura y Portable.- Java está diseñado para soportar aplicaciones que serán instaladas en un entorno de red heterogéneo, con hardware y sistemas operativos diversos. Para hacer esto posible el compilador Java genera “bytecodes”, un formato de código independiente de la plataforma diseñado para transportar código eficientemente a través de múltiples plataformas de hardware y software. Es además portable en el sentido de que es rigurosamente el mismo lenguaje en todas las plataformas. El “bytecode” es traducido a código máquina y ejecutado por la Java Virtual Machine, que es la implementación Java para cada plataforma hardware–software concreta.

Alto Rendimiento.- A pesar de ser interpretado, Java tiene en cuenta el rendimiento, y particularmente en las últimas versiones dispone de diversas herramientas para su optimización. Cuando se necesitan capacidades de proceso intensivas, pueden usarse llamadas a código nativo.

Interpretado, Multi–Hilo y Dinámico.- El intérprete Java puede ejecutar bytecodes en cualquier máquina que disponga de una Máquina Virtual Java (JVM). Además Java incorpora capacidades avanzadas de ejecución multi–hilo (ejecución simultánea de más de un flujo de programa) y proporciona mecanismos de carga dinámica de clases en tiempo de ejecución.

3.3. Características de Java

- Lenguaje de propósito general.
- Lenguaje Orientado a Objetos.
- Sintaxis inspirada en la de C/C++.
- **Lenguaje Multiplataforma.**- Los programas Java se ejecutan sin variación (sin recompilar) en cualquier plataforma soportada (Windows, UNIX, Mac, etc.)

- **Lenguaje Interpretado.**- El intérprete a código máquina (dependiente de la plataforma) se llama Java Virtual Machine (JVM). El compilador produce un código intermedio independiente del sistema denominado *bytecode*.
- **Lenguaje Gratuito.**- Creado por SUN Microsystems, que distribuye gratuitamente el producto base, denominado JDK (Java Development Toolkit) o actualmente J2SE (Java 2 Standard Edition).
- API distribuido con el J2SE muy amplia. Código fuente de las APIs, disponibles libremente.

3.4. Bibliotecas de Clases Java

Los programas java constan de varios elementos llamados clases. Estas clases incluyen elementos llamados métodos, los cuales realizan tareas y devuelven información cuando completan esas tareas. Se pueden crear cada uno de los elementos necesarios para crear un programa java. Sin embargo, la mayoría de los programadores en java aprovechan las ricas colecciones de clases existentes en las bibliotecas de clases java, que también se conocen como APIs (Interfaces de Programación de Aplicaciones). Por lo tanto, en realidad existen dos fundamentos para conocer el “mundo” de java: El primero es el lenguaje java en sí, de manera que se programen todas las clases necesarias; el segundo son las clases incluidas en las extensas bibliotecas de las clases de java. Las bibliotecas de clases las proporcionan principalmente los vendedores de compiladores o los vendedores de software. En el desarrollo de esta monografía se utilizarán los APIs de comunicaciones y JDBC, que se describen a continuación:

3.4.1. API de Comunicaciones

El API de comunicaciones es una extensión estándar que permite realizar comunicaciones con los puertos serie RS-232 y el paralelo IEEE-1284, esto permitirá realizar aplicaciones que utilizan los puertos de comunicaciones (tarjetas

inteligentes, módems, faxes, etc.) independientes de la plataforma. Este API no se encuentra incluido en el JDK, pero es una extensión de éste, así que antes de empezar a trabajar se debe instalar este nuevo API en las maquinas donde se realizará el desarrollo y que vayan a ejecutar programas realizados con éste.

3.4.1.1. Instalación

Los pasos para la instalación del API de comunicaciones son los siguientes:

1. Se copia el fichero Win32com.dll a <JDK>\jre\bin. En el caso UNIX las librerías se deberían instalar en donde indique la variable de entorno. Por ejemplo en una maquina Sun será donde indique la variable LD_LIBRARY_PATH.
2. Se copia el archivo comm.jar a <JDK>\jre\lib\ext.
3. Se copia javax.comm.properties a <JDK>\jre\lib este fichero contendrá los controladores que soportará el API, esto es así ya que se pueden crear nuestros propios controladores de puertos.

La nomenclatura <JDK> indica el directorio donde se ha instalado el paquete de desarrollo de Java.

3.4.1.2. Características

En el paquete de comunicaciones javax.comm se tiene una serie de clases que permiten tratar varios niveles de programación, estos niveles son los siguientes:

Nivel Alto.- En este nivel están las clases CommPortIdentifier y CommPort que permiten el acceso a los puertos de comunicación. Nos encontramos ante la clase principal del paquete ya que lo primero que se debe hacer antes de empezar a utilizar un puerto es descubrir si está libre para su utilización.

Nivel Medio.- Se trabaja con las clases SerialPort y ParallelPort que cubren las interfaces física RS-232 para el puerto serie y IEEE-1284 para el puerto paralelo.

Nivel Bajo.- Este nivel trabaja directamente en el sistema operativo y en él se encuentra el desarrollo de controladores.

Los servicios que proporciona este paquete son:

- Poder obtener los puertos disponibles así como sus características.
- Abrir y mantener una comunicación en los puertos.
- Resolver colisiones entre aplicaciones. Gracias a este servicio se pueden tener varias aplicaciones Java funcionando y utilizando los mismos puertos, y no solo se obtiene el puerto ocupado sino también qué aplicación lo está utilizando.
- Se dispone de métodos para el control de los puertos de entrada/salida a bajo nivel, de esta forma no solo se limita a enviar y recibir datos sino que se conoce en qué estado está el puerto. Así en un puerto serial se puede no solo cambiar los estados sino que se puede programar un evento que notifique el cambio de cualquier estado.

3.4.1.3. Clases y Métodos Java para Control de Puertos de Comunicación

Lo primero antes de intentar abrir un puerto será ver si ya tiene o no un propietario. Para tener esta información se debe obtener el objeto de CommPortIdentifier correspondiente al puerto que se realizará con alguno de los siguientes métodos:

- getPortIdentifiers() método que entrega un enumerado con tantos objetos CommPortIdentifier como puertos dispongamos.

- `getPortIdentifier(String)` este método proporciona el objeto correspondiente al puerto que se le pase como parámetro. Este método deberá saber manejar la excepción `NoSuchPortException` que saltará en el caso de que se solicite un puerto inexistente.

Una vez obtenido el objeto del puerto, se pueden utilizar una serie de métodos que proporcionan información del puerto y abrirlo para poder iniciar con una comunicación. Estos métodos son los siguientes:

- `getName()` da el nombre del puerto. En el caso de haber utilizado el método `getPortIdentifier(String)` será el mismo valor pasado como parámetro.
- `getPortType()` devuelve un entero que informa el tipo de puerto (serie o paralelo). Para no tener que recordar el valor de cada tipo de puerto se dispone de las constantes `PORT_PARALLEL` y `PORT_SERIAL`.
- `isCurrentlyOwned()` informa si está libre o no el puerto. En caso de que esté ocupado se puede conocer quien lo está utilizando mediante el método `getCurrentOwner()`.
- `open(String, int)` abre y por lo tanto reserva un puerto, en el caso de que se intente abrir un puerto que ya se encuentre en uso saltará la excepción `PortInUseException`. Los parámetros que se deben pasar son un `String` con el nombre de la aplicación que reserva el puerto y un `int` que indica el tiempo de espera para abrir el puerto.

Este método da un objeto `CommPort`, realmente esto es una clase abstracta de la que heredan las clases `SerialPort` y `ParallelPort`. Una vez que se tiene este objeto se deberán encaminar sus salidas a `OutputStream` y las entradas a `InputStream` una vez realizado esto la escritura y lectura de los puertos serán tan fácil como utilizar los métodos de estas clases que pertenecen al estándar del JDK.

- `addPortOwnershipListener(CommPortOwnershipListener)` permite añadir una clase que escuche los cambios de propietarios en los puertos.

- Si una vez que esté registrado un oyente de eventos, se desea eliminarlo, se debe utilizar el siguiente método:

`removePortOwnershipListener(CommPortOwnershipListener)`.

La clase `CommPort` es una clase abstracta que describe los métodos comunes de comunicación y serán las clases que hereden de ellas (`SerialPort` y `ParallelPort`) las que añadan métodos y variables propias del tipo del puerto. Los métodos más importantes de esta clase se listan a continuación:

- `close()` libera el puerto que se reservó con `open`, este método notificará el cambio de dueño a las clases que se hubiesen registrado con el método `addPortOwnershipListener`.
- `getInputStream()` enlaza la entrada del puerto al `InputStream` que devuelve para leer del puerto.
- `getOutputStream()` enlaza la salida del puerto al `OutputStream` que devuelve para poder escribir en el puerto de la misma forma que si se escribiera en un fichero.
- `getInputBufferSize()` informa el tamaño del buffer de entrada del puerto. Este tamaño se puede modificar con el método `setInputBufferSize(int)`. Estos métodos no siempre dan el resultado esperado ya que será la memoria disponible y el sistema operativo quien al final decida si realizar o no la operación.
- `getOutputBufferSize()` informa el tamaño del buffer de salida, como en el caso anterior se tiene un método para modificar el tamaño que es `setOutputBufferSize(int)`. Al igual que pasaba con los métodos anteriores su correcto funcionamiento depende del sistema operativo en si y del desarrollo del controlador.

La clase `SerialPort` se encuentran interfaces de bajo nivel del puerto paralelo que cumplen el estándar RS-232. La clase `SerialPort` hereda de la clase abstracta `CommPort` y por lo tanto cuenta con sus métodos pero además de estos se dispone de otros métodos y variables específicas para el tratamiento de los puertos serie.

- `setSerialPortParams(int, int, int, int)` configura los parámetros del puerto serie, este método deberá tratar la excepción `UnsupportedCommOperationException` que saltará en el caso de que se le introduzcan valores no soportados.

Los parámetros del método son:

1. La velocidad. Si se pasara 1210 (valor no válido) se produciría la excepción y no se modificarían los datos.
 2. El segundo parámetro son los bits de datos. Para indicar el valor se pueden utilizar las constantes de la clase que se tiene para lo mismo (`DATABITS_5`, `DATABITS_6`, `DATABITS_7` o `DATABITS_8`).
 3. El bit o bits de stop que se utiliza, y que puede ser 1, 2 o 1½. Las constantes que definen estas configuraciones son: `STOPBITS_1`, `STOPBITS_2` y `STOPBITS_1_5` respectivamente.
 4. Paridad que se utiliza y que puede ser `PARITY_NONE` en el caso de no utilizar paridad, `PARITY_ODD` para la paridad impar, `PARITY_EVEN` paridad par, `PARITY_MARK` paridad por marca y `PARITY_SPACE` paridad por espacio.
- `getBaudrate()` da la velocidad a la que está preparada para transmitir y que se puede cambiar con el método `setSerialPortParams(int, int, int, int)`.
 - `getDataBits()` da la configuración de número de datos y al igual que el método anterior se puede cambiar con el mismo método, los valores posibles son los mismos que utiliza el método de configuración de parámetros.
 - `getStopBits()` indica la configuración de bits de parada y al igual que en los métodos anteriores se puede configurar con el mismo método.
 - `getParity()` indica la paridad que utiliza.

- `getFlowControlMode()` proporciona la configuración del control de flujo que se puede cambiar con el método `setFlowControlMode(int)` y sus valores son: `FLOWCONTROL_NONE` no existe control de flujo, (`FLOWCONTROL_RTSCCTS_IN` o `FLOWCONTROL_RTSCCTS_OUT`) para el control de flujo por hardware y (`FLOWCONTROL_XONXOFF_IN` o `FLOWCONTROL_XONXOFF_OUT`) para control de flujo por software. El método `setFlowControlMode(int)` al igual que pasaba con `setSerialPortParams(int, int, int, int)` deberá de saber capturar la excepción `UnsupportedCommOperationException`.

- `addEventListener(SerialPortEventListener)` permite escuchar los cambios de estado del puerto, si se quiere quitar este oyente se utilizaría el método `removeEventListener()`. Mediante los métodos `notifyOnXXX(boolean)`, se habilita o deshabilita la notificación de los diferentes cambios que pueden ser:
 - `DataAvailable` existen datos en la entrada.
 - `OutputEmpty` el buffer de salida esta vacío.
 - `CTS` cambio de Clear To Send.
 - `DSR` cambio de Data Set Ready.
 - `RingIndicator` cambio en RI.
 - `CarrierDetect` cambio en CD.
 - `ParityError` se ha producido un error de paridad
 - `BreakInterrupt` se detecta una rotura en la línea.

La clase oyente deberá de tener el método `serialEvent(SerialPortEvent)` que recibirá un objeto que trata los eventos del puerto serie con tres métodos importantes:

- `getEventType()` informa del evento que se ha producido.
- `getNewValue()` devuelve el nuevo estado.
- `getoldValue()` da el estado anterior al cambio.
- `isDTR()` informa el estado terminal que se podrá cambiar con `setDTR(boolean)`.
- `isRTS()` indica si ha solicitado permiso para transmitir o no y para cambiar su estado se tiene el método `setRTS(boolean)`.
- `isCTS()` informa si se puede transmitir, `isDSR()` dará el estado en el que se encuentra el pin DSR. `IsRI()` informa de si está el indicador de timbre, `isCD()` indicará si hay portadora.

La clase `ParallelPort` contiene el interfase de bajo nivel del puerto paralelo que cumple la norma IEEE-1284. El estándar IEEE-1284 cuenta con 8 líneas de datos, 5 entrada de estado y 4 salidas de control. La tabla 3 describe las conexiones DB25 de interfaces IEEE-1284 en modo SPP:

Tabla 3 Pinaje de Conectores DB25 en Modo SPP (nombres subrayados indican señal negada)

PIN	NOMBRE	DESCRIPCIÓN
1	STROBE	Indica que hay datos validos en la línea de datos D0-7
2	D0	Dato bit 0
3	D1	Dato bit 1
4	D2	Dato bit 2
5	D3	Dato bit 3
6	D4	Dato bit 4
7	D5	Dato bit 5
8	D6	Dato bit 6
9	D7	Dato bit 7
10	<u>ACK</u>	Indica que el ultimo carácter se recibió
11	<u>BUSY</u>	Indica que la impresora está ocupada y no puede recoger datos

12	<u>PE</u>	Sin papel
13	<u>SLCT</u>	Indica que la impresora esta en línea
14	AUTO FD	Indica a la impresora que realice una alimentación de línea
15	<u>ERROR</u>	Existe algún error en la impresora
16	<u>INIT</u>	Pide a la impresora que se inicie
17	<u>SLCT IN</u>	Indica a la impresora que esta seleccionada

En este estándar se puede trabajar con 5 posibles modos de funcionamiento:

1. Compatible o SPP.
2. Modo Nibble, reserva el canal de datos y se combina con el modo SPP.
3. Modo Byte, este modo es bidireccional utilizado por IBM y tiene la capacidad de deshabilitar los controladores usando la línea de datos.
4. EPP (Extended Parallel Port) es el modo extendido y se utiliza para periféricos que principalmente no son impresoras.
5. ECP (Enhanced Capabilities Port) al igual que el modo EPP es bidireccional y se utiliza para comunicar diversos tipos de periféricos, este modo es una propuesta de Hewlett Packard y Microsoft.

La clase ParallelPort es una clase que hereda de CommPort. Esta clase cuenta con métodos que facilitan el uso del puerto paralelo y son descritos a continuación:

- addEventListener(ParallelPortEventListener ev) asigna un oyente para cuando aparezca un error en el puerto no lo notifique. Informará cuando el buffer de salida esté lleno o cuando la impresora indique error. Los métodos que habilitan o deshabilitan estos avisos son: notifyOnError(boolean) y notifyOnBuffer(boolean). Para quitar el oyente se utiliza el método removeEventListener().

- `getOutputBufferFree()` informa de los bytes disponibles en el buffer de salida.
- `isPaperOut()` devuelve un valor true en caso de que la impresora se quede sin papel, esto es lo mismo que preguntar por el estado de la señal PE.
- `isPrinterBusy()` devuelve un valor true en el caso de que la impresora esté ocupada y se corresponde con el estado BUSY.
- `isPrinterSelected()` informa si la impresora está seleccionada.
- `isPrinterError()` chequea si ocurre un error en la impresora y esto se refleja con el estado de ERROR que está en el pin 15.
- `restart()` realiza un reset en la impresora.
- `suspend()` suspende la salida.
- `getMode()` indica el modo configurado del puerto paralelo y sus valores son:
 - `LPT_MODE_SPP` modo compatible unidireccional.
 - `LPT_MODE_PS2` modo Byte.
 - `LPT_MODE_EPP` modo extendido.
 - `LPT_MODE_ECP` modo mejorado.
 - `LPT_MODE_NIBBLE` modo nibble.
- `setMode(int)` configura el modo del puerto paralelo, los valores que pueden tomar son los anteriores y además `LPT_MODE_ANY` que será el mejor modo disponible. Este método deberá ser capaz de tratar la excepción `UnsupportedCommOperationException` que saltará en el caso de introducir un modo no soportado.

3.4.2. El API JDBC

Es un API de Java para acceder a sistemas de bases de datos, y prácticamente a cualquier tipo de dato tabular. El API JDBC consiste de un conjunto de clases e interfaces que permiten a cualquier programa Java acceder a sistemas de bases de datos de forma homogénea. En otras palabras, con el API JDBC no es necesario escribir un programa para acceder a Sybase, otro para Oracle, y otro para MySQL; con esta API, se puede crear un sólo programa en Java capaz de enviar sentencias SQL a la base de datos apropiada. Al igual que ODBC, la aplicación Java debe tener acceso a un controlador (*driver*) JDBC adecuado. Este controlador implementa la funcionalidad de todas las clases de acceso a datos y proporciona la comunicación entre el API JDBC y la base de datos. Al usar JDBC se pueden hacer tres cosas:

- Establecer una conexión a una fuente de datos (ejemplo: una base de datos).
- Mandar consultas y sentencias a la fuente de datos.
- Procesar los resultados.

Los distribuidores de bases de datos suministran los controladores que implementan el API JDBC y que permiten acceder a sus propias implementaciones de bases de datos. De esta forma JDBC proporciona a los programadores de Java una interfaz de alto nivel y les evita el tener que tratar con detalles de bajo nivel para acceder a bases de datos. En el caso del manejador de bases de datos MySQL, Connector/J es el controlador JDBC oficial. Cabe señalar que actualmente JDBC es el nombre de una marca registrada, y ya no más un acrónimo; es decir, JDBC ya no debe entenderse como "*Java Database Connectivity*".

3.4.3. Cargando el controlador JDBC

En el API JDBC se definen los objetos que proporcionan toda la funcionalidad que se requiere para el acceso a bases de datos. Para trabajar con este paquete se tiene que importar `java.sql`, añadiendo la sentencia `“import java.sql.*;”`.

Luego de importar el paquete `java.sql` se debe cargar el controlador JDBC, es decir un objeto `Driver` específico para una base de datos que define cómo se ejecutan las instrucciones para esa base de datos en particular. Hay varias formas de hacerlo, pero la más sencilla es utilizar el método `forName()` de la clase `Class`:

Sintaxis:

```
Class.forName("Controlador JDBC");
```

Ejemplo:

```
Class.forName("com.mysql.jdbc.Driver");
```

Debe tenerse en cuenta que el método estático `forName()` definido por la clase `Class` genera un objeto de la clase especificada. Cualquier controlador JDBC tiene que incluir una parte de iniciación estática que se ejecuta cuando se carga la clase. En cuanto el cargador de clases carga dicha clase, se ejecuta la iniciación estática, que pasa a registrarse como un controlador JDBC en el `DriverManager`:

Ejemplo:

```
Class.forName("Controlador JDBC"); es equivalente a:  
Class c = Class.forName("Controlador JDBC");  
Driver driver = (Driver)c.newInstance();  
DriverManager.registerDriver(driver);
```

Algunos controladores no crean automáticamente una instancia cuando se carga la clase. Si `forName()` no crea por sí solo una instancia del controlador, se tiene que hacer lo siguiente de manera explícita:

Sintaxis:

```
Class.forName("Controlador JDBC").newInstance();
```

Ejemplo:

```
Class.forName("com.mysql.jdbc.Driver").newInstance();
```

3.4.4. Establecer la Conexión

Una vez registrado el controlador con el DriverManager, se debe especificar la fuente de datos a la que se desea acceder. En JDBC, una fuente de datos se especifica por medio de un URL con el prefijo de protocolo jdbc:

Sintaxis:

```
jdbc:(subprotocolo):(subnombre)
```

El (subprotocolo) expresa el tipo de controlador, normalmente es el nombre del sistema de base de datos, como db2, oracle, mysql, etc. El contenido y la sintaxis de (subnombre) dependen del (subprotocolo), pero en general indican el nombre y la ubicación de la fuente de datos. Por ejemplo, para acceder a la base de datos *centralilla* en tres tipos de sistemas, el URL sería de la siguiente manera:

```
String url = "jdbc:oracle:centralilla";
```

```
String url = "jdbc:db2:dbserver.ibm.com/centralilla";
```

```
jdbc:mysql://[servidor][:puerto]/[base_de_datos][?param1=valor1][param2=valor2]..
```

Una vez que se ha determinado el URL, se puede establecer una conexión a una base de datos. El objeto Connection es el principal objeto utilizado para proporcionar un vínculo entre las bases de datos y una aplicación en Java. Connection proporciona métodos para manejar el procesamiento de transacciones, para crear objetos y ejecutar instrucciones SQL, y para crear objetos para la ejecución de procedimientos almacenados. Se puede emplear el objeto Driver o el objeto DriverManager para crear un objeto Connection. Se utiliza el método connect() para el objeto Driver, y el método getConnection() para el objeto DriverManager. El objeto Connection proporciona una conexión estática a la base de datos. Esto significa que hasta que se llame en forma explícita a su método close() para cerrar la conexión o se destruya el objeto Connection, la conexión a la base de datos permanecerá activa. La manera más usual de establecer una conexión a una base de datos es invocando el método getConnection() de la clase DriverManager. A menudo, las bases de datos están protegidas con nombres de usuario (login) y

contraseñas (password) para restringir el acceso. El método `getConnection()` permite que el nombre de usuario y la contraseña se pasen como parámetros.

Ejemplo:

```
String login = "root";  
String password = "central";  
Connection conn = DriverManager.getConnection(url,login,password);
```

3.4.5. Creación de Sentencias

Para ejecutar instrucciones SQL y procesar los resultados de las mismas, se debe hacer uso de un objeto `Statement`. Los objetos `Statement` envían comandos SQL a la base de datos, y pueden ser de cualquiera de los tipos siguientes:

- Un comando de definición de datos como `CREATE TABLE` o `CREATE INDEX`.
- Un comando de manipulación de datos como `INSERT`, `DELETE` o `UPDATE`.
- Un sentencia `SELECT` para consulta de datos.

Un comando de manipulación de datos devuelve un contador con el número de filas (registros) afectados, o modificados, mientras una instrucción `SELECT` devuelve un conjunto de registros denominado conjunto de resultados (*result set*). La interfaz `Statement` no tiene un constructor, sin embargo, podemos obtener un objeto `Statement` al invocar el método `createStatement()` de un objeto `Connection`.

Ejemplo:

```
conn = DriverManager.getConnection(url,login,password);  
Statement stmt = conn.createStatement();
```

Al crear el objeto `Statement`, se puede emplear para enviar consultas a la base de datos usando los métodos `execute()`, `executeUpdate()` o `executeQuery()`. El método depende del tipo de consulta que se va a enviar al servidor de base de datos:

Tabla 4 Métodos para ejecutar sentencias SQL

Método	Descripción
execute()	Se usa principalmente cuando una sentencia SQL regresa varios conjuntos de resultados. Esto ocurre principalmente cuando se está haciendo uso de procedimientos almacenados.
executeUpdate()	Este método se utiliza con instrucciones SQL de manipulación de datos tales como INSERT, DELETE o UPDATE.
executeQuery()	Se usa en las instrucciones del tipo SELECT.

Es recomendable cerrar los objetos Connection y Statement que se hayan creado cuando ya no se necesiten. Cuando en una aplicación Java se están usando recursos externos, como es el caso del acceso a bases de datos con el API JDBC, el recolector de basura de Java (*garbage collector*) no tiene manera de conocer cuál es el estado de esos recursos, y por lo tanto, no es capaz de liberarlos en el caso de que ya no sean útiles. En estos casos se pueden quedar almacenados en memoria grandes cantidades de recursos relacionados con la aplicación de base de datos que se está ejecutando. Es por esto que se recomienda que se cierren de manera explícita los objetos Connection y Statement. De manera similar a Connection, la interfaz Statement tiene un método close() que permite cerrarlo de manera explícita. Al cerrar un objeto Statement se liberan los recursos que están en uso tanto en la aplicación Java como en el servidor de base de datos.

3.4.6. Ejecución de Consultas

Al ejecutar sentencias SELECT usando el método executeQuery(), se obtiene como respuesta un conjunto de resultados, que en Java es representado por un objeto ResultSet. El objeto ResultSetMetaData proporciona varios métodos para obtener información sobre los datos que están dentro de un objeto ResultSet. Estos métodos permiten entre otras cosas obtener de manera dinámica el número de columnas en el conjunto de resultados, así como el nombre y el tipo de cada columna.

Ejemplo:

```
Statement stmt = conn.createStatement();
```

```
ResultSet res = stmt.executeQuery("SELECT * FROM centralilla");  
ResultSetMetaData metadata = res.getMetaData();
```

La información del conjunto de resultados se puede obtener usando el método `next()` y los diversos métodos `getXXX()` del objeto `ResultSet`. El método `next()` permite moverse fila por fila a través del `ResultSet`, mientras que los diversos métodos `getXXX()` permiten acceder a los datos de una fila en particular. Los métodos `getXXX()` toman como argumento el índice o nombre de una columna, y regresan un valor con el tipo de datos especificado en el método. Así por ejemplo, `getString()` regresará una cadena, `getBoolean()` regresará un booleano y `getInt()` regresará un entero. Cabe mencionar que estos métodos deben tener una correspondencia con los tipos de datos que se tienen en el `ResultSet`, y que son a las vez los tipos de datos provenientes de la consulta `SELECT` en la base de datos, sin embargo, si únicamente se desean mostrar los datos se puede usar `getString()` sin importar el tipo de dato de la columna.

3.5. Conclusiones

Java es un lenguaje de programación que tiene la capacidad de manejar comunicaciones entre dispositivos. Esta característica nos ha permitido obtener bibliotecas de clases que realizan tareas útiles para el desarrollo de nuestro proyecto. El API de comunicaciones permite configurar vía software los parámetros dictados por el protocolo RS-232 para que la centralilla telefónica se comunique con el puerto serial de un computador. El segundo API utilizado, es el JDBC, que permite la comunicación de java con el gestor de base de datos MySQL, para almacenar la información de las llamadas registradas en la centralilla telefónica.

CAPITULO IV

EL GESTOR DE BASE DE DATOS MYSQL

4.1. Introducción

El gestor de bases de datos MySQL es, probablemente, el gestor más usado en el mundo del software libre (open source), debido a su ingeniosa arquitectura, que lo hace extremadamente rápido y fácil de personalizar. La extensiva reutilización del código dentro del software y una aproximación minimalística para producir características funcionalmente ricas, ha dado lugar a un sistema de administración de la base de datos incomparable en velocidad, compactación, estabilidad y facilidad de despliegue. Existen infinidad de librerías y otras herramientas que permiten su uso a través de gran cantidad de lenguajes de programación, y además, su fácil instalación y configuración lo hacen acreedor de gran aceptación en el mundo informático.

4.2. Características

MySQL es un sistema de gestión de bases de datos relacional, licenciado bajo la GPL de la GNU. Su diseño multi-hilo le permite soportar una gran carga de forma muy eficiente. MySQL fue creada por la empresa sueca MySQL AB, que mantiene el copyright del código fuente del servidor SQL, así como también de la marca. Aunque MySQL es software libre, MySQL AB distribuye una versión comercial de MySQL, que no se diferencia de la versión libre más que en el soporte técnico que se ofrece, y la posibilidad de integrar este gestor en un software propietario, ya que de no ser así, se vulneraría la licencia GPL.

Las principales características de MySQL son enumeradas a continuación:

- 1.** Aprovecha la potencia de sistemas multiprocesador, gracias a su implementación multi-hilo.
- 2.** Soporta gran cantidad de tipos de datos para las columnas.

3. Dispone de API's en gran cantidad de lenguajes (C, C++, Java, PHP, etc).
4. Gran portabilidad entre sistemas.
5. Soporta hasta 32 índices por tabla.
6. Gestión de usuarios y claves, con un muy buen nivel de seguridad en los datos.

4.3. Conectarse al Servidor MySQL

Para conectarse al servidor, usualmente necesitamos un nombre de usuario (login) y una contraseña (password), y si el servidor al que deseamos conectar está en una máquina diferente de la nuestra, también necesitamos indicar el nombre o la dirección IP de dicho servidor. Una vez que conocemos estos tres valores, podemos conectarnos de la siguiente manera:

Sintaxis:

```
shell> mysql -h NombreServidor -u NombreUsuario -p
```

Al ejecutar este comando, se nos pedirá que proporcionemos también la contraseña para el nombre de usuario que estamos utilizando. Si la conexión se pudo establecer de manera satisfactoria, recibiremos el mensaje de bienvenida y estaremos en el prompt de MySQL.

Ejemplo

```
mysql> mysql -h central -u root -p
Enter password: *****
mysql>
```

El prompt nos indica que MySQL está listo para recibir comandos. Algunas instalaciones permiten que los usuarios se conecten de manera anónima al servidor corriendo en la máquina local. Al conectarnos de manera satisfactoria, podemos desconectarnos en cualquier momento al escribir en la línea de comandos cualquiera de estas sentencias "quit", "exit", o presionar CONTROL+D.

4.4. Comandos Básicos de MySQL

Cada comando ilustra distintos procesos dentro de MySQL y al usarlos se debería tener en cuenta las siguientes normas:

- Un comando normalmente consiste de un sentencia SQL seguida por un punto y coma.
- Cuando emitimos un comando, MySQL lo manda al servidor para que lo ejecute, nos muestra los resultados y regresa el prompt indicando que está listo para recibir más consultas.
- MySQL muestra los resultados de la consulta como una tabla (filas y columnas). La primera fila contiene etiquetas para las columnas. Las filas siguientes muestran los resultados de la consulta. Normalmente las etiquetas de las columnas son los nombres de los campos de las tablas que se usan en la consulta. Si lo recuperado es el valor de una expresión las etiquetas en las columnas son la expresión en sí.

Ejemplo: Consulta que demuestra como MySQL nombra a las etiquetas de las columnas al escribir expresiones matemáticas y trigonométricas:

Ejemplo:

```
mysql> SELECT SIN(PI()/4), (4+1)*5;
```

```
+-----+-----+
| SIN(PI()/4) | (4+1)*5 |
+-----+-----+
|    0.707107 |      25 |
+-----+-----+
```

- MySQL muestra cuántas filas fueron regresadas y cuanto tiempo tardó en ejecutarse la consulta, lo cual indica la eficiencia del servidor, aunque estos valores pueden ser un tanto imprecisos ya que no muestran la hora del computador, y pueden verse afectados por otros factores, como la carga del servidor y la velocidad de comunicación en una red.

- Las palabras reservadas pueden ser escritas en mayúsculas y en minúsculas.

Ejemplo: Las siguientes consultas son equivalentes:

```
mysql> SELECT VERSION(), CURRENT_DATE;
mysql> select version(), current_date;
mysql> SeLeCt vErSiOn(), current_DATE;
```

Al conectarnos al servidor MySQL, aún cuando no hemos seleccionado una base de datos para trabajar, tenemos la posibilidad de escribir algunos comandos que nos ayudarán a familiarizarnos con el funcionamiento de MySQL (se pueden escribir más de una sentencia por línea, siempre y cuando estén separadas por un punto y coma):

Ejemplo:

```
mysql> SELECT VERSION(); SELECT NOW();
```

```
+-----+
| VERSION() |
+-----+
| 3.23.41   |
+-----+
1 row in set (0.01 sec)
```

```
+-----+
| NOW()          |
+-----+
| 2002-10-28 14:26:04 |
+-----+
1 row in set (0.01 sec)
```

```
mysql> SELECT
-> USER(),
-> CURRENT_DATE;
```

```
+-----+-----+
| USER()          | CURRENT_DATE |
+-----+-----+
| root@localhost | 2002-09-14   |
+-----+-----+
1 row in set (0.00 sec)
```

En este ejemplo debe notarse como cambia el prompt de “mysql>” a “-a>”, cuando se escribe una sola consulta en varias líneas. Esta es la manera como MySQL indica que está esperando a que se finalice con la sentencia para poder ejecutar la consulta. En la tabla 4, se muestran los diferentes prompts que podemos obtener en una consulta multi-línea y una breve explicación del significado para MySQL de cada uno de ellos:

Tabla 5 Prompts de MySQL

Prompt	Significado
mysql>	Listo para una nueva consulta
->	Esperando la línea siguiente de una consulta multi-línea
'>	Esperando la siguiente línea para completar una cadena que comienza con una comilla sencilla (')
">	Esperando la siguiente línea para completar una cadena que comienza con una comilla doble (")

Los comandos multi-línea comúnmente ocurren por accidente cuando tecleamos ENTER antes de terminar la sentencia u olvidamos escribir el punto y coma. En estos casos MySQL se queda esperando para que finalicemos la consulta de manera que si escribimos toda la sentencia o el punto y coma MySQL podrá ejecutarla:

Ejemplo:

```
mysql> SELECT USER()
      ->
      -> ;

+-----+
| USER() |
+-----+
| root@localhost |
+-----+
1 row in set (0.00 sec)
```

4.5. Crear y Usar Una Base de Datos

Primeramente usamos la sentencia `SHOW DATABASES` para ver cuáles son las bases de datos existentes en el servidor al que estamos conectados:

```
mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| mysql    |
| test     |
+-----+
2 rows in set (0.00 sec)
```

Las bases de datos "mysql" y "test" son creadas por defecto en la instalación de MySQL. La base de datos "mysql" es requerida, ya que ésta tiene la información de los privilegios de los usuarios de MySQL. La base de datos "test" es creada con el propósito de servir como área de trabajo para los usuarios que inician en el aprendizaje de MySQL. Es posible que no se visualicen todas las bases de datos si el usuario no tiene el privilegio `SHOW DATABASES`. Para crear una base de datos ejecutamos la siguiente línea de comando:

Sintaxis:

```
shell> CREATE DATABASE <NombreBaseDeDatos>;
```

Ejemplo:

```
mysql> CREATE DATABASE centralilla;
Query OK, 1 row affected (0.00 sec)
```

Al crear una base de datos no se selecciona ésta de manera automática; debemos hacerlo de manera explícita, para ello usamos el comando `USE`.

Sintaxis:

```
shell> USE <BaseDeDatos>;
```

Ejemplo:

```
mysql> USE centralilla;  
Database changed
```

La sentencia USE, al igual que QUIT, no requieren el uso del punto y coma, aunque si se usa éste, no hay ningún problema. El comando USE es especial también de otra manera: éste debe ser usado en una sola línea. La base de datos se crea sólo una vez, pero nosotros debemos seleccionarla cada vez que iniciamos una sesión con MySQL. Por ello es recomendable que se indique la base de datos sobre la que vamos a trabajar al momento de invocar al monitor de MySQL.

Ejemplo:

```
shell>mysql -h root -u <contraseña> -p centralilla  
Enter password: *****
```

Observar que "centralilla" no es la contraseña que se está proporcionando desde la línea de comandos, sino el nombre de la base de datos a la que deseamos acceder. Si deseamos proporcionar la contraseña en la línea de comandos después de la opción "-p", debemos de hacerlo sin dejar espacios (por ejemplo, -p<clave>, no como -p <clave>. Sin embargo, escribir nuestra contraseña desde la línea de comandos no es recomendado, porque es inseguro.

Bajo el sistema operativo Unix, los nombres de las bases de datos son sensibles al uso de mayúsculas y minúsculas (no como las palabras reservadas de SQL), por lo tanto debemos de tener cuidado de escribir correctamente el nombre de la base de datos y los nombres de las tablas.

4.6. Crear Tablas

Al momento de crearla, la base de datos está vacía. Esto podremos verificarlo usando el comando SHOW TABLES, que nos lista las tablas creadas dentro de la base de datos:

Ejemplo:

```
mysql> SHOW TABLES;  
Empty set (0.00 sec)
```

La sentencia CREATE TABLE nos permite crear tablas con las atributos particulares que deseemos otorgarle a cada campo que la conforme.

Sintaxis:

```
shell> CREATE TABLE <NombreTabla>  
    (<NombreCampo1> <TipoDeCampo1> (Tamaño1) <opciones>,  
    (<NombreCampo2> <TipoDeCampo2> (Tamaño2) <opciones>, ...  
    (<NombreCampoN> <TipoDeCampoN> (TamañoN) <opciones>);
```

Ejemplo:

```
mysql> CREATE TABLE extension(  
    -> Ext_Numero INT(02) NOT NULL PRIMARY KEY,  
    -> Ext_Responsable VARCHAR(20),  
    -> Ext_Empresa VARCHAR(15),  
    -> Ext_Departamento VARCHAR(20));  
Query OK, 0 rows affected (0.02 sec)
```

Ahora que hemos creado la tabla, la sentencia SHOW TABLES debe producir lo siguiente:

```
mysql> SHOW TABLES;  
+-----+  
| Tables_in_centralilla |  
+-----+  
| extension              |  
+-----+  
1 row in set (0.00 sec)
```

Para verificar que la tabla fue creada como nosotros esperábamos, usaremos la sentencia DESCRIBE:

Sintaxis:

```
shell> DESCRIBE <NombreTabla>;
```

Ejemplo:

```
mysql> DESCRIBE extension;
```

Field	Type	Null	Key	Default	Extra
Ext_Numero	int(2)		PRI	0	
Ext_Responsable	varchar(20)	YES		NULL	
Ext_Empresa	varchar(15)	YES		NULL	
Ext_Departamento	varchar(20)	YES		NULL	

```
4 rows in set (0.08 sec)
```

4.7. Agregar, Modificar y Eliminar Registros de Una Tabla

Después de haber creado una tabla, podemos incorporar algunos datos en ella, para lo cual haremos uso de las sentencias INSERT y LOAD DATA. Para cargar el contenido de un archivo de texto en una tabla, usaremos el siguiente comando:

Sintaxis:

```
shell> LOAD DATA LOCAL INFILE <"Archivo.txt"> INTO TABLE <NombreTabla>;
```

Ejemplo:

```
mysql> LOAD DATA LOCAL INFILE "extension.txt" INTO TABLE extension;
```

LOAD DATA permite especificar el separador de columnas, y el separador de registros, por defecto el tabulador es el separador de columnas (campos), y el salto de línea es el separador de registros, en el ejemplo son suficientes para que la sentencia LOAD DATA lea correctamente el archivo "extension.txt". Si lo deseado es añadir un registro a la vez, entonces se utiliza la sentencia INSERT. En la manera más simple, debemos proporcionar un valor para cada columna en el orden en el cual fueron listados en la sentencia CREATE TABLE.

Sintaxis:

```
shell> INSERT INTO <NombreTabla>  
VALUES (<ValorCampo1>, <ValorCampo2>, ...<ValorCampoN>);
```

Ejemplo:

```
mysql> INSERT INTO extension
-> VALUES(23, 'FELIPE ARÍZAGA', 'ORTIZ & JACOME', 'COMPUTO');
```

Los valores de cadenas y fechas deben estar encerrados entre comillas. También, con la sentencia INSERT podemos insertar el valor NULL directamente para representar un valor nulo, un valor que no conocemos. Para modificar un registro de una tabla se usa la sentencia UPDATE:

Sintaxis:

```
shell> UPDATE <NombreTabla> SET <Campo1>=<ValorCampo1>,
      <Campo2>=<ValorCampo2>, ...<CampoN>=<ValorCampoN>
      WHERE <Condición>;
```

Ejemplo:

```
mysql> UPDATE extension SET Ext_Departamento="SISTEMAS"
WHERE Ext_Numero=23;
```

Para eliminar uno o varios registros de una tabla se usa la sentencia DELETE:

Sintaxis:

```
shell> DELETE FROM <NombreTabla> WHERE <Condición>;
```

Ejemplo:

```
mysql> DELETE FROM extension WHERE Ext_Numero=23;
```

4.8. Recuperar y Ordenar Información de Una Tabla

La sentencia SELECT es usada para obtener la información guardada en una tabla. La forma general de esta sentencia es:

Sintaxis:

```
shell> SELECT <Campo1>, <Campo2>, ...<CampoN>
      FROM <NombreTabla> WHERE <Condición>;
```

Aquí, <Campo#> es la información de la tabla que queremos ver. Puede ser una lista de columnas, o un * para indicar "todas las columnas". <NombreTabla>, es el nombre de la tabla de la cual vamos a obtener los datos. La cláusula WHERE es opcional, y va precedida de la <Condición>, que especifica las condiciones que los registros deben satisfacer para que puedan ser mostrados. La manera más simple de la sentencia SELECT es cuando se recuperan todos los datos de una tabla:

Ejemplo:

```
mysql> SELECT * FROM extension;
```

Ext_Numero	Ext_Responsable	Ext_Empresa	Ext_Departamento
10	RECEPCIÓN	ORTIZ Y JACOME	CONMUTADOR
11	SUPER STOCK	SUPER STOCK	BODEGA
12	ORTIZ Y JACOME	ORTIZ Y JACOME	VENTAS
13	SUPER STOCK	SUPER STOCK	SEGUNDA PLANTA
18	SEGURIDAD	ORTIZ Y JACOME	GARITA
19	ING. JACINTO JACOME	ORTIZ Y JACOME	GERENCIA
20	P.O.S. N.3	SUPER STOCK	SUPER STOCK
21	P.O.S. N.2	SUPER STOCK	SUPER STOCK
23	FELIPE ARIZAGA	ORTIZ Y JACOME	COMPUTO
24	SUPER STOCK	SUPER STOCK	JUEGOS ELECTRONICOS
28	SUPER STOCK	SUPER STOCK	DELICATESSEN
30	SRA. BETTY ORTIZ	SUPER STOCK	GERENCIA
31	P.O.S. N.6	SUPER STOCK	SUPER STOCK
32	P.O.S. N.5	SUPER STOCK	SUPER STOCK
33	P.O.S. N.4	SUPER STOCK	SUPER STOCK
34	INDETERMINADA	INDETERMINADA	INDETERMINADA
36	INDETERMINADA	INDETERMINADA	INDETERMINADA
39	INDETERMINADA	INDETERMINADA	INDETERMINADA
41	SUPER STOCK	SUPER STOCK	TERCERA PLANTA
78	P.O.S.	ORTIZ Y JACOME	ORTIZ Y JACOME
79	P.O.S.	ORTIZ Y JACOME	ORTIZ Y JACOME
99	VARIOS	ORTIZ Y JACOME	VARIOS

22 rows in set (0.01 sec)

Esta forma del SELECT es útil si deseamos ver los datos completos de la tabla, por ejemplo, para asegurarnos de que están todos los registros después de la carga de un archivo. Además se puede seleccionar sólo registros particulares de una

tabla utilizando la opción WHERE, y especificando condiciones sobre cualquier columna, de la siguiente manera:

Ejemplo:

```
mysql> SELECT * FROM extension WHERE Ext_Responsable="FELIPE ARIZAGA";
+-----+-----+-----+-----+
| Ext_Numero | Ext_Responsable | Ext_Empresa      | Ext_Departamento |
+-----+-----+-----+-----+
|          23 | FELIPE ARIZAGA  | ORTIZ Y JACOME  | SISTEMAS         |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

La comparación de cadenas es no sensitiva, así que podemos especificar, por ejemplo: en el campo Ext_Responsable: "FELIPE ARIZAGA", o, "Felipe Arizaga", el resultado de la consulta será el mismo. Para no visualizar todos los campos de una tabla, se tienen que usar los nombres de las columnas deseadas separándolas por una coma.

Ejemplo:

```
mysql> SELECT Ext_Número, Ext_Responsable FROM extension;
+-----+-----+
| Ext_Numero | Ext_Responsable |
+-----+-----+
|          10 | RECEPCIÓN       |
|          11 | SUPER STOCK     |
|          12 | ORTIZ Y JACOME  |
|          13 | SUPER STOCK     |
|          18 | SEGURIDAD       |
|          19 | ING. JACINTO JACOME |
|          20 | P.O.S. N.3     |
|          21 | P.O.S. N.2     |
|          23 | FELIPE ARIZAGA  |
|          24 | SUPER STOCK     |
|          28 | SUPER STOCK     |
|          30 | SRA. BETTY ORTIZ |
|          31 | P.O.S. N.6     |
|          32 | P.O.S. N.5     |
|          33 | P.O.S. N.4     |
|          34 | INDETERMINADA  |
|          36 | INDETERMINADA  |
|          39 | INDETERMINADA  |
```

```

|          41 | SUPER STOCK          |
|          78 | P.O.S.               |
|          79 | P.O.S.               |
|          99 | VARIOS               |
+-----+-----+
22 rows in set (0.01 sec)

```

Se debe notar en los ejemplos anteriores que las filas regresadas son mostradas sin ningún orden en particular. Sin embargo, frecuentemente es más fácil examinar la salida de una consulta cuando las filas son ordenadas en alguna forma útil. Para ordenar los resultados por una columna o múltiples columnas, agregamos la cláusula ORDER BY y las columnas o criterios de ordenamiento.

Ejemplo:

```
mysql> SELECT Ext_Número, Ext_Responsable FROM extensión ORDER BY
Ext_Responsable;
```

```

+-----+-----+
| Ext_Numero | Ext_Responsable      |
+-----+-----+
|          23 | FELIPE ARIZAGA      |
|          34 | INDETERMINADA       |
|          36 | INDETERMINADA       |
|          39 | INDETERMINADA       |
|          19 | ING. JACINTO JACOME |
|          12 | ORTIZ Y JACOME      |
|          78 | P.O.S.              |
|          79 | P.O.S.              |
|          21 | P.O.S. N.2          |
|          20 | P.O.S. N.3          |
|          33 | P.O.S. N.4          |
|          32 | P.O.S. N.5          |
|          31 | P.O.S. N.6          |
|          10 | RECEPCIÓN           |
|          18 | SEGURIDAD           |
|          30 | SRA. BETTY ORTIZ    |
|          11 | SUPER STOCK         |
|          13 | SUPER STOCK         |
|          24 | SUPER STOCK         |
|          28 | SUPER STOCK         |
|          41 | SUPER STOCK         |
|          99 | VARIOS              |
+-----+-----+
22 rows in set (0.01 sec)

```

En las columnas de tipo carácter, el ordenamiento es ejecutado normalmente de forma no sensitiva, es decir, no hay diferencia entre mayúsculas y minúsculas. Sin embargo, se puede forzar un ordenamiento sensitivo al usar el operador BINARY. Para ordenar de forma inversa, debemos agregar la palabra clave DESC al nombre de la columna que estamos usando en el ordenamiento:

Ejemplo:

```
mysql> SELECT Ext_Número, Ext_Responsable FROM extensión ORDER BY  
Ext_Responsable DESC;
```

4.9 Conclusiones

MySQL es un poderoso Gestor de Base de Datos de distribución libre, que puede comunicarse con el lenguaje de programación Java mediante el API JDBC. Esto nos permite usar MySQL en este proyecto para almacenar la información que se receipta desde la centralilla telefónica en tablas relacionadas; para luego, mediante el uso de sentencias SQL, obtener rápidamente consultas y reportes ordenados, filtrados o clasificados de acuerdo al criterio que el usuario establezca. Con esto alcanzamos otro de los objetivos principales de este proyecto.

CAPITULO V

ANÁLISIS Y DISEÑO DEL PROYECTO

5.1. Introducción

La obtención de la información proporcionada por la centralilla telefónica, nos permitirá analizar cada registro para descomponerlo en campos individuales que conformarán las tablas de la Base de Datos del proyecto. La correcta normalización de la información nos dará como resultado tablas relacionadas para generar consultas y reportes rápidos y confiables. El almacenamiento de la información se consigue con el diseño y la construcción del cable necesario, la configuración de sus señales siguiendo las normas citadas en el estándar RS-232 y mediante una aplicación programada en java, todo esto, servirá para conseguir la comunicación entre la centralilla telefónica y el puerto serial de un computador.

5.2. Ambiente de Desarrollo del Proyecto

Las herramientas informáticas que se utilizarán en el desarrollo de este proyecto son:

- El Gestor de Base de Datos MySQL. La Base de Datos creada para almacenar todos los datos de las tablas del proyecto será nombrada como “centralilla”.
- El Lenguaje de Programación Java, con el API de comunicaciones, y el API JDBC (Java DataBase Connectivity) para MySQL (Connector/J).

5.3. Definición de Entidades y sus Atributos

De acuerdo a los registros proporcionados por la centralilla telefónica, determinamos todos los campos que están involucrados en el proyecto. Estos atributos al ser analizados nos dan como resultado las siguientes tablas:

Llamada.- Esta tabla contiene toda la información que proporciona el registro de cada una de las llamadas entrantes y salientes que se realizan a través de la centralilla telefónica.

Tabla 6 Atributos de la Tabla Llamada

ATRIBUTO	TIPO	NULO	LLAVE	DEFAULT
Call_Fecha_Sys	Timestamp	SI	PRI	0000-00-00 00:00:00
Ext_Numero	int(2)		PRI	0
Call_Fecha	Date	SI		NULO
Call_Hora	int(6) unsigned zerofill	SI		NULO
Call_Numero	varchar(18)	SI		NULO
Call_Duración	int(6) unsigned zerofill	SI		NULO
Call_Linea	int(2) unsigned zerofill	SI		NULO
Call_Entrada_Salida	char(1)	SI		NULO

La sentencia SQL para crear la tabla “llamada” y sus atributos es la siguiente:

```
mysql> CREATE TABLE llamada(
-> Call_Fecha_Sys TIMESTAMP,
-> Ext_Numero INT(02),
-> Call_Fecha DATE,
-> Call_Hora INT(06) UNSIGNED ZEROFILL,
-> Call_Numero VARCHAR(18),
-> Call_Duración INT(06) UNSIGNED ZEROFILL,
-> Call_Linea INT(02) UNSIGNED ZEROFILL,
-> Call_Entrada_Salida CHAR(01),
-> PRIMARY KEY (Call_Fecha_Sys, Ext_Numero));
```

Extensión.- Esta tabla se relaciona con el campo Número de Extensión almacenado en la tabla Llamada, además se registra toda la información que se puede obtener de cada uno de los usuarios responsables de las extensiones instaladas a la centralilla telefónica.

Tabla 7 Atributos de la Tabla Extension

ATRIBUTO	TIPO	NULO	LLAVE	DEFAULT
Ext_Numero	Int(2)		PRI	0
Ext_Responsable	varchar(20)	SI		NULO
Ext_Empresa	varchar(15)	SI		NULO
Ext_Departamento	varchar(20)	SI		NULO

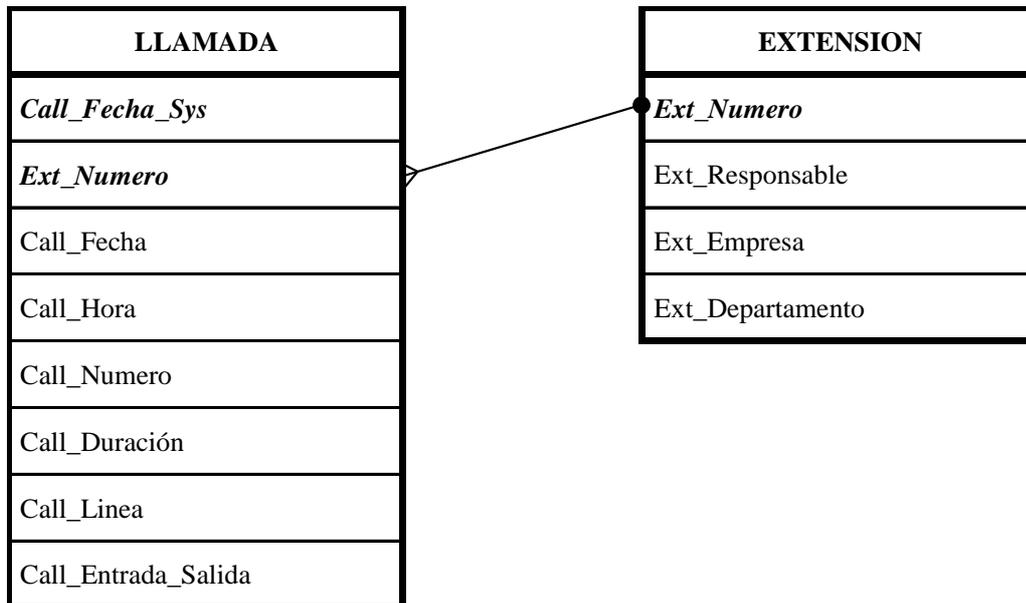
La sentencia SQL para crear la tabla “extension” y sus atributos es la siguiente:

```
mysql> CREATE TABLE extension(  
-> Ext_Numero INT(02) NOT NULL PRIMARY KEY,  
-> Ext_Responsable VARCHAR(20),  
-> Ext_Empresa VARCHAR(15),  
-> Ext_Departamento VARCHAR(20));
```

5.4. Modelo Entidad–Relación

El modelo Entidad–Relación de las tablas “llamada” y “extension”, creadas a partir de la información proporcionada por la centralilla telefónica, se muestra a continuación:

Figura 11 Modelo Entidad–Relación



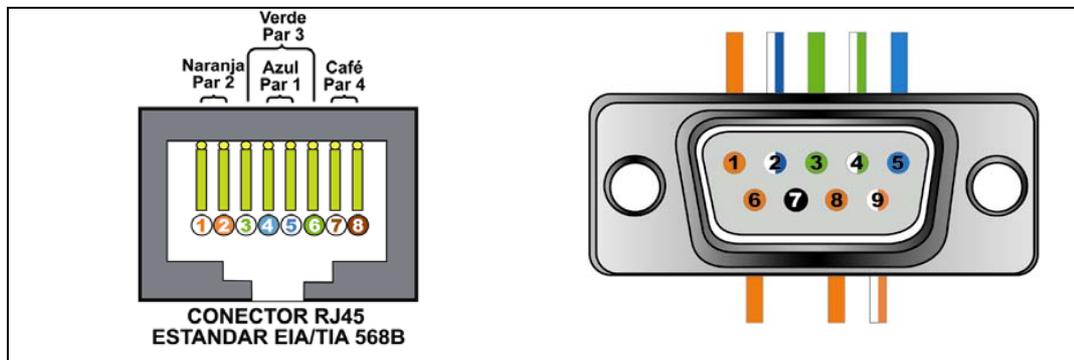
La sentencia SQL usada para crear la relación entre las tablas “llamada” y “extension”, utilizando el campo “Ext_Numero” como llave alterna, es la siguiente:

```
mysql> ALTER TABLE Llamada ADD FOREIGN KEY (Ext_Numero)  
-> REFERENCES Extension (Ext_Numero);
```

5.5. Construcción del Cableado

En la construcción del cable adecuado para la comunicación serial entre el componente SMDR, ubicado en el Procesador ACS de la centralilla telefónica, y el puerto serial de cualquier computador, usaremos las reglas del Protocolo RS-232 y la norma ANSI/TIA/EIA-568B. Siguiendo las reglas de estos estándares utilizaremos cables Blindado y No Blindado, correspondiendo al tipo Par Trenzado de 4 pares. Los conectores que deben instalarse al cable en cada uno de sus extremos son: Un RJ45 de 8 hilos que se conectará al Jack del dispositivo SMDR de la centralilla telefónica y un DB9 hembra que se conectará al puerto serial del computador.

Figura 12 Armado de Conectores RJ45 y DB9



El resultado de la combinación del protocolo RS-232 y la norma ANSI/TIA/EIA-568B nos da como resultado la siguiente tabla de pinaje entre los conectores RJ45 y DB9:

Tabla 8 Configuración del Pinaje entre Conectores RJ45 y DB9

RJ45 (SMDR)	DB9 (PC)	SEÑAL
1	9	RI
2	1-6-8	DCD – DSR – CTS
3	4	DTR
4	5	GND
5	2	RDX
6	3	TDX
7	–	–
8	–	–

5.6. Programación de la Interfase

La aplicación programada en java que permite configurar la comunicación serial entre la centralilla telefónica y el puerto serial de un computador, y la posterior recepción y almacenaje de los datos de las llamadas entrantes y salientes, fue denominada “captura.java”, y es descrita a continuación:

Figura 13 Pantalla de Visualización del Programa Captura

```

C:\WINNT\system32\cmd.exe
<SI>I 03/21/06 14:35          IN 00:00:14 01 26
<SI>I 03/21/06 14:42          IN 00:00:22 05 26
<SI>I 03/21/06 14:46          IN 00:01:30 04 26
<SI>I 03/21/06 14:49          IN 00:00:25 05 26
<SI>C 03/21/06 15:00          098087084 00:00:36 03 15
<SI>C 03/21/06 15:01          28301122830112 00:00:25 07 33
<SI>C 03/21/06 15:01          2943711 00:00:54 02 23
<SI>C 03/21/06 15:00          943711 00:01:18 01 23
<SI>C 03/21/06 14:58          092906690 00:04:01 04 26
<SI>I 03/21/06 15:00          IN 00:01:49 06 15
<SI>C 03/21/06 15:02          098087084 00:03:10 04 15
<BLANCO><0>
<SI>C 03/21/06 15:09          095339775 00:00:58 06 37
<BLANCO>C 03/21/06 15:11      <24>
<SI>C 03/21/06 15:11          2882209 00:01:11 03 37
<SI>C 03/21/06 15:10          2862111 00:02:14 01 17
<SI>C 03/21/06 15:13          098663113 00:00:42 06 17
<SI>C 03/21/06 15:16          2853884 00:00:47 01 10
<SI>C 03/21/06 15:17          2853884 00:00:43 01 10
<SI>C 03/21/06 15:19          022869408 00:00:10 06 26
<SI>C 03/21/06 15:19          022869408 00:00:10 06 26
<BLANCO><0>
FECHA  HORA  NUMERO  DURACION  LINEA  EXTIN  CUEN
<SI>C 03/21/06 15:18          2853884 00:00:41 01 10
<SI>I 03/21/06 15:18          IN 00:00:42 05 10

```

```

import java.sql.*;
import java.io.*;
import java.util.*;
import javax.comm.*;

public class Captura implements Runnable, SerialPortEventListener {
    static final String CONTROLADOR_JDBC = "com.mysql.jdbc.Driver";
    static final String URL_BASEDEDATOS = "jdbc:mysql://localhost/centralilla";
    static final String USUARIO = "root";
    static final String CONTRASENIA = "central";
    static SerialPort serialPort;
    static CommPortIdentifier portId;
    static Enumeration portList;
    static InputStream inputStream;
    static Thread readThread;
    String registro="";
    public static void main(String[] args)
    {
        portList = CommPortIdentifier.getPortIdentifiers();
        while (portList.hasMoreElements())
        {
            portId = (CommPortIdentifier) portList.nextElement();
            if (portId.getPortType() == CommPortIdentifier.PORT_SERIAL)
            {
                if (portId.getName().equals("COM2"))
                {
                    Captura reader = new Captura();
                }
            }
        }
    }
} // fin de main

public void ejecutarSQL(String cadena) {
    String nro_ext="", fecha="", hora="", nro_lla="", duracion="", linea="", ent_sal="";

```

```

String sentencia="", aceptar;
aceptar="NO";
for (int cont1=0; cont1<cadena.length(); cont1++ )
{
    if(cadena.charAt(cont1)=='\n')
        aceptar="SI";
}
if(aceptar=="SI")
{
    if(registro.length()>=55)
    {
        if(registro.substring(0,1).equals("C") || registro.substring(0,1).equals("I"))
        {
            nro_ext    = registro.substring(53,55);
            fecha      = registro.substring( 8,10)+"/"+registro.substring(02,07);
            hora       = registro.substring(11,13)+registro.substring(14,16)+"00";
            nro_lla    = registro.substring(18,35).trim();
            duración   = registro.substring(37,39)+
                registro.substring(40,42)+
                registro.substring(43,45);
            linea      = registro.substring(48,50);
            ent_sal    = registro.substring( 0,01);
            sentencia = "INSERT INTO Llamada VALUES(current_timestamp, "+ nro_ext+", "+
                fecha+"', "+hora+"', "+nro_lla+"', "+duracion+"', "+ linea+"', '"+
                ent_sal+"')";
            if(nro_ext.equals("") || fecha.equals("") || hora.equals("") ||
                duracion.equals("") || linea.equals("") || ent_sal.equals(""))
            {
                System.out.println("<FALTA>"+sentencia);
            }
            else
            {
                try
                {
                    Class.forName(CONTROLADOR_JDBC);
                    Connection conn = DriverManager.getConnection(URL_BASEDEDATOS,
                                                                    USUARIO, CONTRASENIA);

                    Statement instruccion = conn.createStatement();
                    instruccion.executeUpdate(sentencia);
                    System.out.println("<SI>"+registro);
                    conn.close();
                }
                catch(SQLException ex)
                {
                    System.out.println(ex);
                    System.out.println(sentencia);
                }
                catch(ClassNotFoundException ex)
                {
                    System.out.println(ex);
                }
            }
        }
        else
        {
            System.out.println("<NO>"+registro);
        }
        else
        {
            System.out.println("<BLANCO>"+registro+"<"+registro.length()+">");
        }
        registro="";
    }
    else
    {
        registro+=cadena;
    }
} // fin de ejecutarSQL

public Captura() {
    try {
        serialPort = (SerialPort) portId.open("BasculaApp", 1500);
    }
    catch (PortInUseException e) {}
    try {
        inputStream = serialPort.getInputStream();
    }
}

```

```

    }
    catch (IOException e) {}
    try {
        serialPort.addEventListener(this);
    }
    catch (TooManyListenersException e) {}
        serialPort.notifyOnDataAvailable(true);
        try {
            serialPort.setSerialPortParams(1200,
                SerialPort.DATABITS_8,
                SerialPort.STOPBITS_2,
                SerialPort.PARITY_NONE);
        }
        catch (UnsupportedCommOperationException e) {}
        readThread = new Thread(this);
        readThread.start();
    } // fin de Captura

public void run() {
    try {
        Thread.sleep(20000);
    } catch (InterruptedException e) {}
} // fin de run

public void serialEvent(SerialPortEvent event) {
    switch(event.getEventType())
    {
        case SerialPortEvent.BI:
        case SerialPortEvent.OE:
        case SerialPortEvent.FE:
        case SerialPortEvent.PE:
        case SerialPortEvent.CD:
        case SerialPortEvent.CTS:
        case SerialPortEvent.DSR:
        case SerialPortEvent.RI:
        case SerialPortEvent.OUTPUT_BUFFER_EMPTY:
            break;
        case SerialPortEvent.DATA_AVAILABLE:
            byte[] readBuffer = new byte[8];
            try {
                while (inputStream.available() > 0)
                {
                    int numBytes = inputStream.read(readBuffer);
                }
                ejecutarSQL(new String(readBuffer));
            }
            catch (IOException e) {}
            break;
    }
}
} // fin de serialEvent

```

5.7. Programación de Consultas

La aplicación programada en java que permite visualizar los datos almacenados en las tablas del proyecto, filtrarlos por cualquiera de sus columnas y realizar un mantenimiento completo (Inserción, Modificado y Borrado) de los registros de la tabla “llamada” fue denominada “Reporte.java”, y es descrita a continuación:

Figura 14 Pantalla del Programa Reporte

Ext	Fecha	Hora	Numero	Duracion	Linea	E/S	Responsable
19	15/12/2005	13:02:00	IN	00:00:11	3	I	ING. JACINTO JACOME
19	15/12/2005	15:27:00	04256023825	00:01:28	3	C	ING. JACINTO JACOME
19	15/12/2005	15:30:00	IN	00:01:28	3	I	ING. JACINTO JACOME
19	15/12/2005	15:34:00	049895	00:00:13	3	C	ING. JACINTO JACOME
19	15/12/2005	15:34:00	098957845	00:01:39	6	C	ING. JACINTO JACOME
19	15/12/2005	15:43:00	042355295	00:02:18	1	C	ING. JACINTO JACOME
19	16/12/2005	08:35:00	IN	00:02:31	1	I	ING. JACINTO JACOME
19	16/12/2005	08:58:00	2847950	00:00:24	3	C	ING. JACINTO JACOME
19	16/12/2005	08:55:00	IN	00:06:46	5	I	ING. JACINTO JACOME
19	16/12/2005	09:23:00	IN	00:01:41	4	I	ING. JACINTO JACOME
19	16/12/2005	09:25:00	2847950	00:02:58	4	C	ING. JACINTO JACOME
19	16/12/2005	09:45:00	IN	00:02:55	6	I	ING. JACINTO JACOME
19	16/12/2005	10:06:00	IN	00:06:21	6	I	ING. JACINTO JACOME
19	16/12/2005	10:15:00	IN	00:10:31	4	I	ING. JACINTO JACOME
19	16/12/2005	10:31:00	IN	00:01:17	3	I	ING. JACINTO JACOME
19	16/12/2005	10:35:00	2847950	00:00:31	3	C	ING. JACINTO JACOME
19	16/12/2005	10:36:00	IN	00:05:39	2	I	ING. JACINTO JACOME
19	16/12/2005	11:09:00	IN	00:02:41	1	I	ING. JACINTO JACOME
19	16/12/2005	11:10:00	IN	00:04:01	4	I	ING. JACINTO JACOME
19	16/12/2005	12:06:00	IN	00:01:37	4	I	ING. JACINTO JACOME
19	16/12/2005	12:46:00	IN	00:00:19	4	I	ING. JACINTO JACOME
19	16/12/2005	12:56:00	IN	00:01:55	4	I	ING. JACINTO JACOME
19	16/12/2005	13:02:00	IN	00:00:36	4	I	ING. JACINTO JACOME
19	16/12/2005	13:05:00	2882594	00:01:56	3	C	ING. JACINTO JACOME
19	16/12/2005	13:10:00	IN	00:03:50	4	I	ING. JACINTO JACOME

```
// Reporte.java
// Formulario para mantenimiento y consulta de Llamadas
import java.sql.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import javax.swing.*;
import javax.swing.table.*;
import java.text.DecimalFormat;
import lib.awtextra.*;

public class Reporte extends JFrame {
    static final String CONTROLADOR_JDBC = "com.mysql.jdbc.Driver";
    static final String URL_BASEDEDATOS = "jdbc:mysql://localhost/centralilla";
    static final String USUARIO = "root";
    static final String CONTRASENIA = "central";
    static final String CONSULTA_PREDETERMINADA =
        "SELECT Llamada.Ext_Numero AS 'Ext', Call_Fecha AS 'Fecha'," +
        "concat((substring(Call_Hora,1,2)),':'," +
        "substring(Call_Hora,3,2)),':'," +
        "substring(Call_Hora,5,2)) AS 'Hora', " +
        "Call_Numero AS 'Numero', " +
        "concat((substring(Call_Duracion,1,2)),':'," +
        "substring(Call_Duracion,3,2)),':'," +
        "substring(Call_Duracion,5,2)) AS 'Duracion', " +
        "Call_Linea AS 'Linea', " +
        "Call_Entrada_Salida AS 'E/S', " +
        "Ext_Responsable AS 'Responsable' " +
        "FROM Llamada, Extension " +
        "WHERE Llamada.Ext_Numero=Extension.Ext_Numero";

    private Container contenedor;
    private JLabel eExt_Numero, eCall_Fecha, eCall_Hora, eCall_Numero;
    private JTextField ctExt_Numero, ctCall_Fecha, ctCall_Hora, ctCall_Numero;
    private JLabel eCall_Duracion, eCall_Linea, eCall_Entrada_Salida;
    private JTextField ctCall_Duracion, ctCall_Linea, ctCall_Entrada_Salida;
    private JTextArea taConsulta;
    private JScrollPane csConsulta;
    private Box csLlamada = Box.createHorizontalBox();
    private JButton bAnadir, bEditar, bBorrar, bFiltrar, bCerrar;
    private JButton bAceptar, bCancelar;
}
```

```

private ResultSetTableModel modeloTabla;
private JTable tablaResultados;
private String cadena;
String nro_ext="", fecha="", hora="", nro_lla="", duracion="", linea="", ent_sal="";
String sentencia="";
private int accion = 0;
private static DecimalFormat dosDigitos = new DecimalFormat( "00" );
// configurar GUI
public Reporte() {
    super( "Ficha de Llamadas" );
    contenedor = getContentPane();
    contenedor.setBackground( Color.WHITE );
    contenedor.setLayout( new AbsoluteLayout() );
    // crear componentes de GUI
    ctExt_Numero = new JTextField();
    ctCall_Fecha = new JTextField();
    ctCall_Hora = new JTextField();
    ctCall_Numero = new JTextField();
    ctCall_Duracion = new JTextField();
    ctCall_Linea = new JTextField();
    ctCall_Entrada_Salida = new JTextField();
    eExt_Numero = new JLabel( "Ext" );
    eCall_Fecha = new JLabel( "Fecha" );
    eCall_Hora = new JLabel( "Hora" );
    eCall_Numero = new JLabel( "Numero" );
    eCall_Duración = new JLabel( "Duracion" );
    eCall_Linea = new JLabel( "Linea" );
    eCall_Entrada_Salida = new JLabel( "E/S" );
    bAnadir = new JButton("Nuevo ", new ImageIcon("imagenes/iconos/anadir.gif" ));
    bEditar = new JButton("Editar", new ImageIcon("imagenes/iconos/editar.gif" ));
    bBorrar = new JButton("Eliminar", new ImageIcon("imagenes/iconos/borrar.gif" ));
    bFiltrar = new JButton("Filtrar ", new ImageIcon("imagenes/iconos/buscar.gif" ));
    bCerrar = new JButton("Salir ", new ImageIcon("imagenes/iconos/cerrar.gif" ));
    bAnadir.setToolTipText( "Agregar un nuevo registro");
    bEditar.setToolTipText( "Editar el registro actual");
    bBorrar.setToolTipText( "Eliminar el registro actual");
    bFiltrar.setToolTipText( "Filtrar registros");
    bCerrar.setToolTipText( "Cerrar al ventana actual");
    bAnadir.setMargin( new Insets( 0,0,0,0 ) );
    bEditar.setMargin( new Insets( 0,0,0,0 ) );
    bBorrar.setMargin( new Insets( 0,0,0,0 ) );
    bFiltrar.setMargin( new Insets( 0,0,0,0 ) );
    bCerrar.setMargin( new Insets( 0,0,0,0 ) );
    bAceptar = new JButton("Aceptar ",
        new ImageIcon("imagenes/iconos/aceptar.gif" ));
    bCancelar = new JButton("Cancelar",
        new ImageIcon("imagenes/iconos/cancelar.gif" ));
    taConsulta = new JTextArea ( "", 3, 100 );
    taConsulta.setWrapStyleWord( true );
    taConsulta.setLineWrap ( true );
    csConsulta = new JScrollPane( taConsulta,
        ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED,
        ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER );
    tablaResultados = new JTable ( modeloTabla );
    tablaResultados = extraerConsulta( CONSULTA_PREDETERMINADA );
    tablaResultados.setSelectionMode ( ListSelectionModel.SINGLE_SELECTION );
    csLlamada.add ( new JScrollPane ( tablaResultados ));
    contenedor.add( new JLabel( new ImageIcon( "imagenes/logo.jpg" ) ),
        new AbsoluteConstraints( 0, 0, -1, -1 ) );
    contenedor.add( new JLabel( new ImageIcon( "imagenes/java.jpg" ) ),
        new AbsoluteConstraints( 760, 0, -1, -1 ) );
    contenedor.add(csLlamada, new AbsoluteConstraints( 20, 90, 500, 420 ) );
    contenedor.add(bAnadir, new AbsoluteConstraints( 20, 530, 80, 30 ) );
    contenedor.add(bEditar, new AbsoluteConstraints( 100, 530, 80, 30 ) );
    contenedor.add(bBorrar, new AbsoluteConstraints( 260, 530, 80, 30 ) );
    contenedor.add(bFiltrar, new AbsoluteConstraints( 340, 530, 80, 30 ) );
    contenedor.add(bCerrar, new AbsoluteConstraints( 420, 530, 80, 30 ) );
    contenedor.add(bAceptar, new AbsoluteConstraints( 545, 480, 105, 30 ) );
    contenedor.add(bCancelar, new AbsoluteConstraints( 650, 480, 105, 30 ) );
    contenedor.add(new JSeparator(), new AbsoluteConstraints(540, 90, 210, -1 ) );
    contenedor.add(new JSeparator(), new AbsoluteConstraints(540, 250, 210, -1 ) );
    contenedor.add(eExt_Numero, new AbsoluteConstraints( 540, 100, -1, -1 ) );
    contenedor.add(eCall_Fecha, new AbsoluteConstraints( 540, 120, -1, -1 ) );
    contenedor.add(eCall_Hora, new AbsoluteConstraints( 540, 140, -1, -1 ) );
    contenedor.add(eCall_Numero, new AbsoluteConstraints( 540, 160, 140, -1 ) );
    contenedor.add(eCall_Duracion, new AbsoluteConstraints( 540, 180, 140, -1 ) );
    contenedor.add(eCall_Linea, new AbsoluteConstraints( 540, 200, 140, -1 ) );

```

```

contenedor.add(eCall_Entrada_Salida, new AbsoluteConstraints(540, 220, 140,-1));
contenedor.add(ctExt_Numero, new AbsoluteConstraints( 600, 100, 140, -1 ) );
contenedor.add(ctCall_Fecha, new AbsoluteConstraints( 600, 120, 140, -1 ) );
contenedor.add(ctCall_Hora, new AbsoluteConstraints( 600, 140, 140, -1 ) );
contenedor.add(ctCall_Numero, new AbsoluteConstraints( 600, 160, 140, -1 ) );
contenedor.add(ctCall_Duracion, new AbsoluteConstraints( 600, 180, 140, -1 ) );
contenedor.add(ctCall_Linea, new AbsoluteConstraints( 600, 200, 140, -1 ) );
contenedor.add(ctCall_Entrada_Salida, new AbsoluteConstraints( 600,220,140,-1) );
contenedor.add( csConsulta, new AbsoluteConstraints( 545, 250, 200, 200 ) );
activarPorDefecto();
Font fuente = new Font("Arial", Font.PLAIN,10);
tablaResultados.setFont(fuente);
establecerAnchoColumnas();
bAceptar.addActionListener( new ActionListener() {
    public void actionPerformed ( ActionEvent evento ) {
        String sentencia = "";
        switch (accion)
        {
            case 1: sentencia = "INSERT INTO Llamada VALUES (current_timestamp, " +
                ctExt_Numero.getText () + "," +
                ctCall_Fecha.getText () + "," +
                ctCall_Hora.getText () + "," +
                ctCall_Numero.getText () + "," +
                ctCall_Duracion.getText() + "," +
                ctCall_Linea.getText () + "," +
                ctCall_Entrada_Salida.getText() + "' )";
            break;
            case 2: sentencia = "UPDATE Llamada SET " +
                "Ext_Numero=" + ctExt_Numero.getText () + "," +
                "Call_Fecha=" + ctCall_Fecha.getText () + "," +
                "Call_Hora=" + ctCall_Hora.getText () + "," +
                "Call_Numero=" + ctCall_Numero.getText () + "," +
                "Call_Duracion=" + ctCall_Duracion.getText() + "," +
                "Call_Linea=" + ctCall_Linea.getText () + "," +
                "Call_Entrada_Salida=" + ctCall_Entrada_Salida.getText() +
                "' WHERE Ext_Numero=" + nro_ext + " AND " +
                "Call_Fecha=" + fecha + "' AND " +
                "Call_Hora=" + hora + " AND " +
                "Call_Numero=" + nro_lla + "' AND " +
                "Call_Duracion=" + duracion + " AND " +
                "Call_Linea=" + linea + " AND " +
                "Call_Entrada_Salida=" + ent_sal + "' ";
            break;
            case 3: sentencia = "DELETE FROM Llamada WHERE " +
                "Ext_Numero=" + nro_ext + " AND " +
                "Call_Fecha=" + fecha + "' AND " +
                "Call_Hora=" + hora + " AND " +
                "Call_Numero=" + nro_lla + "' AND " +
                "Call_Duracion=" + duracion + " AND " +
                "Call_Linea=" + linea + " AND " +
                "Call_Entrada_Salida=" + ent_sal + "' ";
            break;
            case 4: sentencia = construirConsulta();
            break;
        }
        if ( accion < 4 ){
            ejecutarSQL( sentencia );
        }
        else {
            cargarTabla( sentencia );
            establecerAnchoColumnas();
        }
        activarPorDefecto();
    }
}); // fin del listener de bAceptar

bCancelar.addActionListener( new ActionListener() {
    public void actionPerformed ( ActionEvent evento ) {
        activarPorDefecto();
    }
}); // fin del listener de bCancelar

bAnadir.addActionListener( new ActionListener() {
    public void actionPerformed ( ActionEvent evento ) {
        accion = 1;
        ctExt_Numero.setText ( "" );
        ctCall_Fecha.setText ( "" );
    }
});

```

```

        ctCall_Hora.setText          ( " " );
        ctCall_Numero.setText        ( " " );
        ctCall_Duracion.setText      ( " " );
        ctCall_Linea.setText         ( " " );
        ctCall_Entrada_Salida.setText( " " );
        activarPantallaDatos();
    }
} ); // fin del listener de bAnadir

bEditar.addActionListener( new ActionListener() {
    public void actionPerformed ( ActionEvent evento ) {
        accion = 2;
        activarPantallaDatos();
    }
} ); // fin del listener de bEditar

bBorrar.addActionListener( new ActionListener() {
    public void actionPerformed ( ActionEvent evento ) {
        accion = 3;
        activarPantallaDatos();
    }
} ); // fin del listener de bBorrar

bFiltrar.addActionListener( new ActionListener() {
    public void actionPerformed ( ActionEvent evento ) {
        accion = 4;
        ctExt_Numero.setText      ( " " );
        ctCall_Fecha.setText      ( " " );
        ctCall_Hora.setText        ( " " );
        ctCall_Numero.setText      ( " " );
        ctCall_Duracion.setText    ( " " );
        ctCall_Linea.setText       ( " " );
        ctCall_Entrada_Salida.setText( " " );
        activarPantallaDatos();
    }
} ); // fin del listener de bFiltrar

bCerrar.addActionListener( new ActionListener() {
    public void actionPerformed ( ActionEvent evento ) {
        dispose();
    }
} ); // fin del listener de bCerrar

tablaResultados.addKeyListener(new KeyAdapter() {
    public void keyReleased(KeyEvent evento) {
        leerRegistro();
    }
} );

tablaResultados.addMouseListener(new MouseAdapter() {
    public void mouseClicked(MouseEvent evento) {
        leerRegistro();
    }
} );

setSize( 800, 600 );
setVisible( true );
} // fin del constructor

private String construirConsulta() {
    String sentencia = CONSULTA_PREDETERMINADA;
    String condicion = "";
    if (ctExt_Numero.getText().length() != 0)
        condicion+=" AND Llamada.Ext_Numero LIKE " + ctExt_Numero.getText();
    if (ctCall_Fecha.getText().length() != 0)
        condicion+=" AND Llamada.Call_Fecha LIKE ' " + ctCall_Fecha.getText() + "'";
    if (ctCall_Hora.getText().length() != 0)
        condicion += " AND Llamada.Call_Hora LIKE " + ctCall_Hora.getText();
    if (ctCall_Numero.getText().length() != 0)
        condicion+=" AND Llamada.Call_Numero LIKE ' " + ctCall_Numero.getText() + "'";
    if (ctCall_Duracion.getText().length() != 0)
        condicion += " AND Llamada.Call_Duracion " + ctCall_Duracion.getText();
    if (ctCall_Linea.getText().length() != 0)
        condicion += " AND Llamada.Call_Linea LIKE " + ctCall_Linea.getText();
    if (ctCall_Entrada_Salida.getText().length() != 0)
        condicion += " AND Llamada.Call_Entrada_Salida LIKE ' " +
            ctCall_Entrada_Salida.getText() + "'";
}

```

```

        sentencia += condicion;
    return sentencia;
} // fin de construirConsulta

private void leerRegistro() {
    int i = tablaResultados.getSelectedRow();
    cadena = tablaResultados.getValueAt( i, 0 ).toString();
    ctExt_Numero.setText(cadena);
    cadena = tablaResultados.getValueAt( i, 1 ).toString();
    ctCall_Fecha.setText(cadena);
    cadena = tablaResultados.getValueAt( i, 2 ).toString();
    cadena = cadena.substring(0,2)+cadena.substring(3,5)+cadena.substring(6,8);
    ctCall_Hora.setText(cadena);
    cadena = tablaResultados.getValueAt( i, 3 ).toString();
    ctCall_Numero.setText(cadena);
    cadena = tablaResultados.getValueAt( i, 4 ).toString();
    cadena = cadena.substring(0,2)+cadena.substring(3,5)+cadena.substring(6,8);
    ctCall_Duracion.setText(cadena);
    cadena = dosDigitos.format(tablaResultados.getValueAt( i, 5 ));
    ctCall_Linea.setText(cadena);
    cadena = tablaResultados.getValueAt( i, 6 ).toString();
    ctCall_Entrada_Salida.setText(cadena);
    nro_ext = ctExt_Numero.getText();
    fecha = ctCall_Fecha.getText();
    hora = ctCall_Hora.getText();
    nro_lla = ctCall_Numero.getText();
    duracion = ctCall_Duracion.getText();
    linea = ctCall_Linea.getText();
    ent_sal = ctCall_Entrada_Salida.getText();
} // fin de leerRegistro

public void cargarTabla( String sentencia ) {
    try {
        modeloTabla.establecerConsulta( sentencia );
    }
    catch ( SQLException excepcionSQL ) {
        JOptionPane.showMessageDialog( null,
            excepcionSQL.getMessage(), "Error en la base de datos",
            JOptionPane.ERROR_MESSAGE );
    }
} // fin de cargarTabla

private JTable extraerConsulta( String sentencia ) {
    try {
        modeloTabla = new ResultSetTableModel(
            CONTROLADOR_JDBC, URL_BASEDEDATOS, USUARIO, CONTRASENIA, sentencia );
    }
    catch(SQLException ex) {
        System.out.println(ex);
    }
    catch(ClassNotFoundException ex) {
        System.out.println(ex);
    }
}
return new JTable( modeloTabla );
} // fin de extraerConsulta
public void ejecutarSQL( String sentencia ) {
    try {
        Connection conn=DriverManager.getConnection(URL_BASEDEDATOS,USUARIO,CONTRASENIA);
        Statement instruccion = conn.createStatement();
        instruccion.executeUpdate( sentencia );
        conn.close();
        cargarTabla( CONSULTA_PREDETERMINADA );
    }
    catch(SQLException ex) {
        System.out.println(ex);
    }
} // fin de ejecutarSQL

public void activarPorDefecto() {
    bAnadir.setEnabled( true );
    bEditar.setEnabled( true );
    bBorrar.setEnabled( true );
    bFiltrar.setEnabled( true );
    bCerrar.setEnabled( true );
    ctExt_Numero.setEnabled ( false );
    ctCall_Fecha.setEnabled ( false );
    ctCall_Hora.setEnabled ( false );
}

```

```

ctCall_Numero.setEnabled ( false );
ctCall_Duracion.setEnabled( false );
ctCall_Linea.setEnabled ( false );
ctCall_Entrada_Salida.setEnabled( false );
bAceptar.setVisible ( false );
bCancelar.setVisible ( false );
csConsulta.setVisible( false );
eExt_Numero.setVisible ( true );
eCall_Fecha.setVisible ( true );
eCall_Hora.setVisible ( true );
ctExt_Numero.setVisible ( true );
ctCall_Fecha.setVisible ( true );
ctCall_Hora.setVisible ( true );
ctCall_Numero.setVisible ( true );
ctCall_Duracion.setVisible( true );
ctCall_Linea.setVisible ( true );
ctCall_Entrada_Salida.setVisible( true );
} // fin de activarPorDefecto

public void activarPantallaDatos() {
bAnadir.setEnabled( false );
bEditar.setEnabled( false );
bBorrar.setEnabled( false );
bFiltrar.setEnabled( false );
bCerrar.setEnabled( false );
ctExt_Numero.setEnabled ( true );
ctCall_Fecha.setEnabled ( true );
ctCall_Hora.setEnabled ( true );
ctCall_Numero.setEnabled ( true );
ctCall_Duracion.setEnabled( true );
ctCall_Linea.setEnabled ( true );
ctCall_Entrada_Salida.setEnabled( true );
bAceptar.setVisible ( true );
bCancelar.setVisible ( true );
ctExt_Numero.requestFocus ( true );
} // fin activarPantallaDatos

public void establecerAnchoColumnas() {
tablaResultados.getColumn(tablaResultados.getColumnIndex(0)).setPreferredWidth( 20);
tablaResultados.getColumn(tablaResultados.getColumnIndex(1)).setPreferredWidth( 50);
tablaResultados.getColumn(tablaResultados.getColumnIndex(2)).setPreferredWidth( 40);
tablaResultados.getColumn(tablaResultados.getColumnIndex(3)).setPreferredWidth( 60);
tablaResultados.getColumn(tablaResultados.getColumnIndex(4)).setPreferredWidth( 40);
tablaResultados.getColumn(tablaResultados.getColumnIndex(5)).setPreferredWidth( 20);
tablaResultados.getColumn(tablaResultados.getColumnIndex(6)).setPreferredWidth( 15);
tablaResultados.getColumn(tablaResultados.getColumnIndex(7)).setPreferredWidth(100);
} // fin establecerAnchoColumnas

public static void main( String args[] ) {
Reporte aplicacion = new Reporte();
aplicacion.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
} // fin de main
} // fin de la clase

```

5.8. Conclusiones

Después de realizar el análisis de toda la información disponible, es cómodo desarrollar las aplicaciones necesarias, haciendo uso de herramientas de distribución libre como Java y MySQL. Estas herramientas combinadas han hecho posible la favorable culminación del proyecto, por sus funcionalidades que cumplen con los requerimientos de las aplicaciones y por su extensa documentación que fue necesaria para configurar la comunicación serial y lograr el almacenamiento de los datos.

CAPITULO VI

CONCLUSIONES Y RECOMENDACIONES

Las normas o especificaciones del protocolo RS-232 sirvieron para controlar físicamente la comunicación serial para el envío y la recepción de información de manera segura. Se logró que el componente SMDR del Procesador ACS de la centralilla telefónica y el puerto serial de un computador se configuraran para enviar y recibir respectivamente las señales de control de flujo por cada uno de los hilos del cable de tipo par trenzado. La creación de las aplicaciones resultó muy amigable, mediante el uso del lenguaje Java y sus bibliotecas de clases, que se usaron para programar la configuración de la comunicación serial, y el almacenamiento de la información en el Gestor de Base de Datos MySQL. Es decir, los objetivos del proyecto se cumplieron completamente.

Al termino de este proyecto, podemos sugerir la necesidad de incentivar de mejor manera el uso de programas de distribución libre en la Escuela de Ingeniería de Sistemas. Poniendo como ejemplo la aplicación desarrollada en esta monografía, podemos afirmar que una de las ventajas de usar estas herramientas, es que en nuestro medio cubren en la mayoría de los casos, los requerimientos o necesidades informáticas por parte de pequeñas, medianas o grandes empresas, convirtiéndose en soluciones interesantes, económica y tecnológicamente hablando

BIBLIOGRAFÍA

- DEITEL Harvey, DEITEL Paul, Como Programar en Java, México, Pearson Educación, 2004, 1326 páginas, Quinta Edición.
- <http://java.sun.com/products/javacomm/>
- <http://support.avaya.com/edoc/docs/partacs/praprgu1.pdf>
- <http://www.mysql-hispano.org/page.php?id=24>
- http://www.programacion.com/php/tutorial/mysql_basico/
- <http://www.programatium.com/tutoriales/cursos/mysql/mysql3.htm>
- http://www.unalmed.edu.co/~daristiz/guias/ing_software/semana_9/api_com_2.doc
- <http://www.arrakis.es/~abelp/ApuntesJava/Introduccion.htm>
- <http://neo.lcc.uma.es/evirtual/cdd/tutorial/fisico/inter232.html>
- <http://eq3.uab.es/personal/baeza/comunicaciones/comunica.htm>
- <http://www.euskalnet.net/shizuka/rs232.htm>
- <http://juandeg.tripod.com/cableutp.htm>
- <http://redesej.tripod.com/cableadoestructurado.html>

DISEÑO DE TESIS

1.- TITULO DEL PROYECTO:

Interfase de Comunicaciones para Centralillas Telefónicas AVAYA E80 CONTROLLER con un computador, desarrollado en JAVA.

2.- RESUMEN DEL PROYECTO:

En el proyecto se desarrollará un software que permita la comunicación con centralillas telefónicas “AVAYA E80 CONTROLLER”. El proceso principal consiste en recibir la información que proporciona la centralilla y almacenarla en una Base de Datos.

Los procesos que comprenden la ejecución de este proyecto son:

Construcción, instalación y configuración del cableado necesario para la comunicación serial de la centralilla telefónica con un computador.-

Para la construcción del cableado se determinará el tipo de material más apto para realizar la comunicación entre la centralilla telefónica y el otro dispositivo, este paso es de mucha importancia, ya que cada uno de los pares o hilos del cable que se utilizan tienen una tarea específica, como por ejemplo el envío, verificación, etc., de los datos que viajan desde la centralilla, esto nos ayudará a determinar la configuración correcta del conector que se instalará en un puerto del otro dispositivo (en nuestro caso el puerto serial de un computador personal).

Además, el determinar un tipo adecuado de cable nos ayuda a evitar pérdidas de información; ya que primero, los dos equipos que se van a comunicar podrían estar ubicados a una distancia considerable; y segundo, no contar con instalaciones adecuadas que permitan una separación entre las redes informáticas y las eléctricas, obligándonos a instalarlas juntas, práctica que no es recomendable, pero que con la utilización de un cable adecuado podría contrarrestarse cualquier inconveniente por interferencias indeseadas.

Programación de la interfase (Java) para lograr el almacenamiento de la información en una base de datos (MySQL).-

Una vez que la comunicación física sea determinada, se procederá a realizar la interfase que posibilitará la interpretación de los datos que la centralilla telefónica envía, para esto se deberá configurar vía software la velocidad de transmisión, número de bits enviados por cada trama, bits de parada y la paridad que utiliza la centralilla al momento de efectuar la conexión con otro dispositivo. Al tener establecidos estos parámetros programaremos en lenguaje Java la aplicación que receptorá trama por trama lo que la centralilla nos envíe hacia el puerto serial, se construirá una cadena con todos los datos concernientes a cada una de las llamadas entrantes y salientes efectuadas por cada extensión. Esta cadena será dividida en campos que conformarán un registro a almacenarse en una tabla de la Base de Datos.

Programación de Reportes y Consultas (Java – MySQL).-

Con la información almacenada, se programaran las consultas y reportes necesarios para tener una estadística sobre el uso de cada una de la extensiones telefónicas que administra la centralilla telefónica.

Pruebas e Implementación del Software en una Empresa para su uso final.-

El último paso para la culminación de este proyecto es realizar las pruebas necesarias para la puesta en marcha del software en una empresa con la necesidad de esta aplicación.

3.- IMPACTO TECNOLÓGICO:

Dentro de lo que engloba el impacto tecnológico podríamos hablar sobre el uso de herramientas informáticas de última generación como el lenguaje de programación Java (para diseñar todos los programas del sistema) y la Base de Datos MySQL (donde almacenaremos toda la información enviada por la centralilla telefónica).

Con las herramientas antes mencionadas podremos implementar un sistema para automatizar las estadísticas sobre el uso telefónico que hasta el momento no ha sido registrado en la empresa.

4.- OBJETIVOS:

Entre los objetivos a realizar dentro de este proyecto están:

- Crear e implementar un sistema para automatizar el control sobre el uso telefónico en una determinada empresa.
- Proveer de un Sistema Estadístico de Llamadas.

5.- IMPACTO SOCIAL:

Este proyecto tiene mucha importancia ya que permitiría conocer el costo telefónico y las horas utilizadas por cada extensión al momento de realizar y recibir llamadas telefónicas. Con esta sistema seríamos capaces de ordenar, filtrar, analizar, y tomar decisiones sobre los resultados en forma inmediata, pudiendo realizar correctivos sobre la mala utilización del personal de los recursos telefónicos de la empresa, permitiéndonos un ahorro en el gasto por este servicio y una mejor administración del tiempo de trabajo de los empleados de la empresa donde se desarrollará el proyecto.

6.- TEORIA REFERENCIAL:

Este proyecto se desarrollará utilizando las siguientes herramientas:

JAVA:

Es un Lenguaje de propósito general Orientado a Objetos, su sintaxis está inspirada en la de C/C++. Es multiplataforma, es decir, que los programas Java se ejecutan sin variación (sin recompilar) en cualquier plataforma soportada (Windows, UNIX, LINUX, Macintosh, etc.)

El Java es un lenguaje gratuito, creado por SUN Microsystems, que distribuye gratuitamente el producto base, denominado JDK (Java Development Toolkit) o actualmente J2SE (Java 2 Standard Edition).

MySQL:

MySQL básicamente es una herramienta cliente/servidor para la gestión de Sistemas Manejadores de Bases de Datos Relacionales (RDBMS) y además es el conjunto de datos que puede proporcionar la capacidad de almacenar y recurrir a estos de forma consecuente con un modelo definido como relacional.

CONNECTOR/J:

El manejador de las clases que da acceso a la base de datos desde JAVA es el controlador JDBC para MySQL conocido como Connector/J.

7.- UNIVERSO DE ESTUDIO:

El universo de estudio para este proyecto será la Empresa Ortiz & Jácome de Comercio Cia. Ltda., en donde está siendo utilizada una centralilla telefónica AVAYA E80 CONTROLLER y no tiene implementado un sistema de control de llamadas.

8.- TÉCNICAS DE INVESTIGACIÓN:

- Investigación Bibliográfica.- Se recopilará toda la información necesaria para un adecuado discernimiento del tema.
- Observación.- A través de la Observación, analizaremos la estructura del proyecto.
- Documentación.- Se llevará una adecuada documentación detallada de todos los aspectos que se vayan desarrollando.

9.- PLAN OPERATIVO:

El Plan de trabajo que se desarrollará para alcanzar los objetivos planteados es el siguiente:

FASE 1: INVESTIGACIÓN DE CAMPO.

- 1.1 Funcionamiento de Centralilla Telefónica: Análisis de la manera en que se comunica la centralilla telefónica con un computador.

- 1.2 Consultas con las personas involucradas en el mantenimiento y configuración de la Centralilla Telefónica.
- 1.3 Construcción e Instalación del Cableado para la comunicación serial.

FASE 2: ANÁLISIS DEL PROYECTO.

- 2.1 Definición de Entidades.
 - 2.1.1 Definición de los Atributos de las Entidades.
 - 2.1.2 Definición de las Relaciones entre Entidades.
- 2.2 Normalización.- Técnica para asegurar la consistencia interna de la base de datos en los sistemas.
 - 2.2.1 Primera Forma Normal.- Identificación de los campos llave y los campos atómicos.
 - 2.2.2 Segunda Forma Normal.- Determinar los campos que dependen funcionalmente de la llave.
 - 2.2.3 Tercera Forma Normal.- Identificación de los campos que no dependen directamente de la llave.
- 2.3 Modelo Entidad Relación.- La relación que se da entre las diferentes entidades.

FASE 3: DISEÑO DEL PROYECTO.

- 3.1 Diseño de Interfaces.
- 3.2 Diseño de Consultas.

FASE 4: PROGRAMACIÓN Y PRUEBAS.

FASE 5: IMPLEMENTACIÓN DEL SOFTWARE.

FASE 6: DOCUMENTACIÓN.

- 6.1 Documentación de la Investigación de Campo.
- 6.2 Documentación de la Normalización, Entidad Relación y de sus entidades.
- 6.3 Documentación sobre el Software utilizado (Java y MySQL).

10.- RECURSOS HUMANOS:

Las personas que participarán en la elaboración de ésta tesis son las siguientes:

- Asesoría en el desarrollo de la tesis:

Ing. Oswaldo Merchán

Ing. Pablo Esquivel

- Alumnos aspirantes al título de Ingenieros de Sistemas:

Felipe Arízaga N.

Jorge Arreaga O.

11.- RECURSOS MATERIALES

Para desarrollar ésta tesis utilizaremos los siguientes recursos materiales:

Hardware:

1 centralilla Telefónica AVAYA E80 CONTROLLER

1 computador Pentium III de 1 Ghz

Disco duro 80 Gb

512 Mb. RAM

Full multimedia

1 computador Pentium 4 de 1.5 Ghz

Disco Duro de 40 Gb

256 Mb. RAM

Full multimedia

1 Unidad de CD–Writer 12x8x32x

1 Impresora Hewlett Packard LaserJet 4600

Software:

Java 1.5.0_03

MySQL Server 4.1

Conector/J

Microsoft Windows 98

Microsoft Windows XP

Microsoft Office 2000

12.- BIBLIOGRAFÍA:

Harvey Deitel, Paul Deitel, Cómo Programar en Java, Editorial Pearson Educación, México, 2004, Quinta Edición.

www.programacion.com

www.mysql-hispano.org

www.javahispano.org

13.- CRONOGRAMA:

Tiempo Actividad	1 ^{era} Semana	2 ^{da} Semana	3 ^{era} Semana	4 ^{ta} Semana	5 ^{ta} Semana	6 ^{ta} Semana	7 ^{ma} Semana	8 ^{va} Semana
Análisis del envío de datos desde la centralilla								
Construcción e instalación del cableado serial								
Análisis y Diseño de la interfase								
Reportes y Consultas								
Documentación e Implementación								