UNIVERSIDAD NACIONAL DE INGENIERÍA

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

SISTEMA DE VIGILANCIA DOMICILIARIO
DESARROLLADO CON J2ME

INFORME DE SUFICIENCIA

PARA OPTAR EL TÍTULO PROFESIONAL DE:

INGENIERO ELECTRÓNICO

PRESENTADO POR:

JOSÉ ANTONIO MELÉNDEZ QUISPE

PROMOCIÓN
2006-II

LIMA-PERÚ
2011

SISTEMA DE VIGILANCIA DOMICILIARIO DESARROLLADO CON J2ME

Agradezco a Dios por cada día de vida,
a mi madre por ser tan comprensiva y sincera,
a mi esposa por los bellos momentos que pasamos
y por los que están por venir,
a mis profesores y amigos que día a día me
enseñan lo bueno que es aprender.

SUMARIO

El presente trabajo tiene como finalidad desarrollar un sistema de seguridad domiciliaria de bajo costo, y ofrecer una alternativa a las personas que mantienen una constante preocupación por mantener sus hogares controlados y monitoreados ante cualquier intrusión no deseada.

Con la apertura del mercado de sistemas de seguridad han surgido diferentes sistemas, sensores y diseños para proveer de las seguridades necesarias a bienes y áreas dentro de los cuales interactúan. Todos estos sistemas en su mayoría, permiten al usuario tener acceso visual a sus hogares desde cualquier lugar alrededor del mundo mediante el empleo del Internet como medio de acceso a una imagen en tiempo real para de esta forma poder tener pleno conocimiento del estado del hogar.

Teniendo en cuenta la amplia gama de sistemas existentes y las facilidades que éstos brindan, se ha realizado un análisis de los posibles elementos a emplear sin que estos generen un mayor gasto. Lo que el presente proyecto propone es utilizar la red de telefonía móvil y el envío de MMS para tener controlado el hogar.

Algo importante de considerar es que el sistema no implica una inversión elevada en cuanto a recursos de diseño ya que en su mayoría el sistema está desarrollado en software libre permitiéndole así una mayor adaptación y funcionalidad.

ÍNDICE

INTRODUCCIÓN

El presente trabajo describe el estudio y análisis para la implementación de un Sistema de Seguridad Domiciliaria que integre sistemas embebidos y dispositivos móviles operados remotamente con un sistema de mensajería basado en J2ME. Dicho sistema esta programado para enviar una alerta cuando ocurra una intrusión en el hogar.

Todo el sistema de vigilancia se puede dividir en dos subsistemas, un subsistema para el envío de mensajes de texto o multimedia y un subsistema para el control y monitoreo de los diferentes dispositivos que pueden se añadidos al sistema.

En el capítulo 1 se muestra una perspectiva general del sistema de seguridad domiciliaria enfocado al envío de MMS. Además se exponen los distintos sistemas que se ofrecen en el mercado actual y la evolución de los mismos seguida de los cambios tecnológicos en esta área. Aquí se presentan los distintos problemas y posibles soluciones existentes en el mercado que se debe considerar a la hora de seleccionar un buen sistema de seguridad que sea capaz de actuar ante una posible amenaza al entorno en que operan.

En el capítulo 2 se explica en detalle las diferentes herramientas de administración y desarrollo necesarias para la implementación del subsistema de envío de SMS, así como también su posible utilidad en el desarrollo de otro tipo de sistemas que manejan JAVA como una herramienta de control.

En el capítulo 3 se explican las características de la programación usada en J2ME. Además se explica en detalle los paquetes de programación MMAPI y WMA que permiten el correcto funcionamiento del sistema.

En el capítulo 4 se muestran las simulaciones realizadas para el subsistema de envío de MMS, permitiendo demostrar que la aplicación puede ser implementada a futuro en ambientes reales que satisfagan las necesidades de seguridad de los usuarios.

En el capítulo 5 se indican los aspectos importantes que tienen que ser considerados al momento de implementar el subsistema de control, se explica cada una de las partes que componen este subsistema.

En general, este trabajo tiene como objetivo principal el desarrollo de un sistema de seguridad domiciliaria de bajo costo y escalable que pueda ser usado en cualquier ambiente que se requiera brindar seguridad y tranquilidad a los usuarios.

CAPÍTULO I
ASPECTOS GENERALES DEL SISTEMA

Este capítulo muestra una perspectiva general del sistema de seguridad a desarrollar. Se presentan las consideraciones generales que permitirán el desarrollo de un sistema domiciliario y escalable con cualquier sistema ya implementado, así como los puntos a tratar en el presente trabajo. Además se realiza un breve estudio de la realidad actual en cuanto a seguridad domiciliaria.

1.1 Alcance y limitaciones del proyecto

El proyecto busca implementar un sistema de seguridad domiciliaria de bajo costo que cumpla con las necesidades básicas de los usuarios. Este sistema es operado con una herramienta de monitoreo, utilizando un dispositivo móvil conectado a un sistema de sensores los cuales pueden ser instalados en lugares comunes como el hogar.

El subsistema de envío mensaje, es desarrollado bajo software libre; y es básicamente el punto en el cual se centra los primeros capítulos de este informe, el capítulo 5 brinda los puntos a considerar en el subsistema de control.

1.1.1 Control y monitoreo domiciliario

Uno de los puntos importantes al momento de diseñar un sistema de seguridad domiciliaria es ofrecer un control y monitoreo de su domicilio las 24 horas del día, esto se logra a través de cualquier equipo remoto que pueda enviar una alerta.

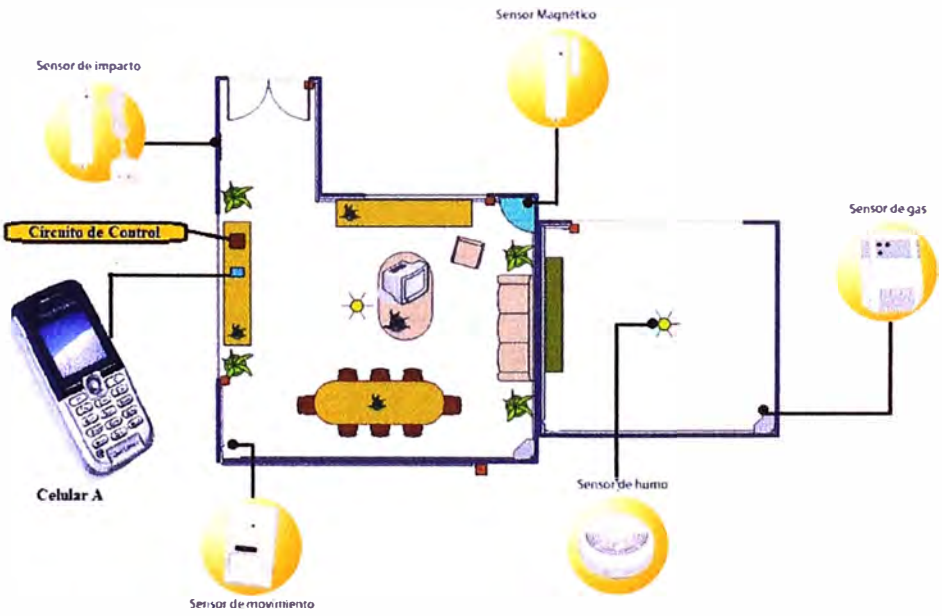

Figura 1.1 Ubicación de sensores en una casa modelo

1.1.2 Estrategia a implementar

En esta primera etapa el informe está centrado en el envío de SMS o MMS con el dispositivo remoto, como ejemplo es aplicado al control de la cámara del celular, llegando a la conclusión de que efectivamente existe comunicación entre ambos dispositivos, el subsistema es dividido en 6 fases, los cuales son:

Fase 1:

El sensor de la apertura de la caja fuerte es activado, el controlador envía una señal al celular (bluetooth ó cable de conexión),

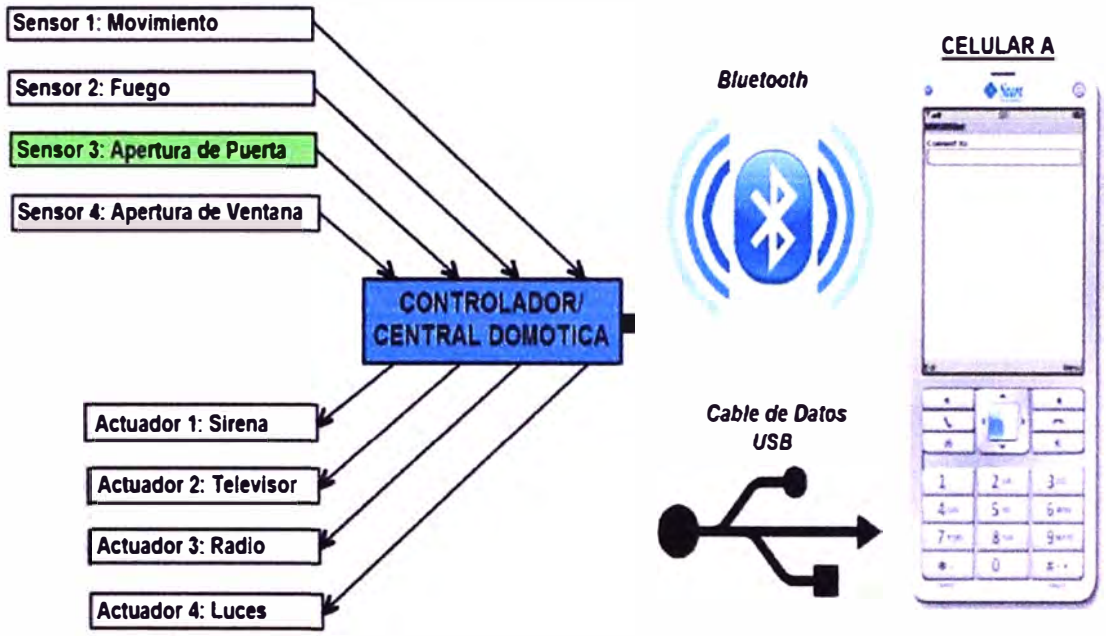

Figura 1.2 Fase 1

Fase 2:

El celular A al recibir la señal del controlador, activa la cámara y realiza la toma de foto del escenario.

Figura 1.3 Fase 2

Fase 3:

El celular A genera un mensaje multimedia, con el asunto relacionado al sensor activado, añadiendo la imagen capturada por la cámara interna del celular, luego procederá a enviar el MMS al Celular B, usando la red móvil existente.

Figura 1.4 Fase 3

Fase 4:

En el celular B se recibirá el con MMS con el asunto y la imagen de la captura realizada por la cámara, el usuario al ver el mensaje decidirá que acción realizar, al ver la imagen de un sujeto desconocido podría llamar a la policía o solicitar ayuda a un vecino para que alerte del robo, o activar vía mensaje una sirena que ahuyente al ladrón, tal como se puede ver en la siguientes Fases.

Figura 1.5 Fase 4

Fase 5:

Posteriormente el celular B puede enviar un mensaje con el asunto ALTAVOZ al celular A.

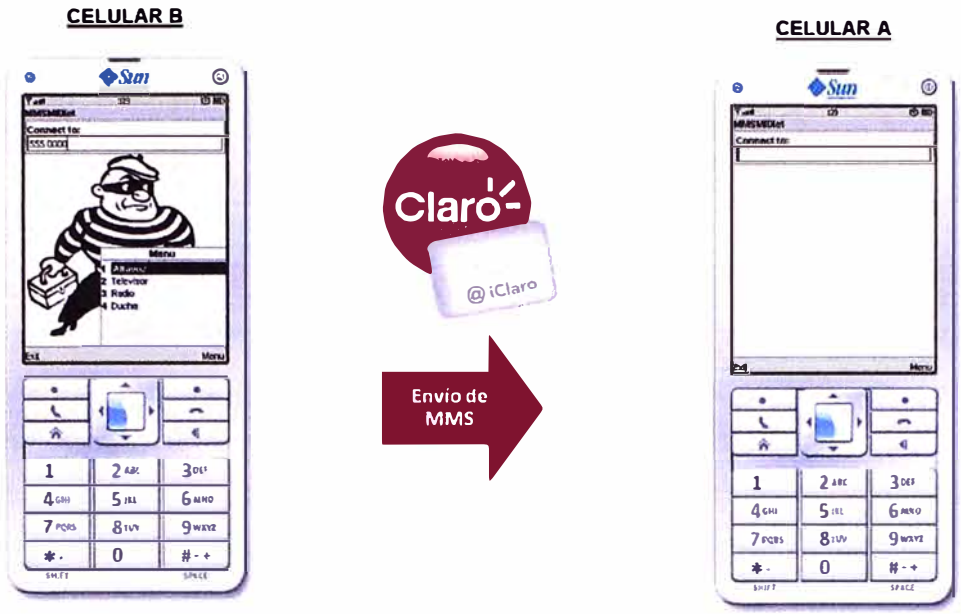

Figura 1.6 Fase 5

Fase 6:

El celular A recibirá el mensaje y relacionará el asunto del mensaje con una orden que debe realizar, el cual podría ser que comunique al controlador que active los altavoces.

Figura 1.7 Fase 6

Como ya se mencionó este proyecto está enfocado en dos puntos importes del sistema, como son la captura de la foto con la cámara interna del dispositivo móvil y el envío y recepción del mensaje multimedia.

1.1.3 Limitaciones del proyecto

El presente proyecto tiene las siguientes limitaciones:

- Los primeros capítulos del informe estará centrado en el análisis y desarrollo del programa para enviar MMS, como aplicación se muestra la captura de fotos con un dispositivo móvil y su envío por mensaje.
- El programa no puede enviar un video en tiempo real lo visualizado por el celular que se encuentra en el domicilio, puesto que para poder enviar por MMS se requiere adjuntar el archivo del video.
- El celular del domicilio debe contar con crédito para el envío de los MMS.
- El envió de los MMS depende de la Red del operador, si en el preciso instante que se intenta enviar el MMS la red del operador está sobrecargada el mensaje no será enviado.
- El sistema manda una alerta cuando uno de los sensores es activado, el usuario responde con un mensaje, el celular que recibe retransmite al microcontrolador la orden, el cual lee el nuevo mensaje recibido y ejecuta la orden correspondiente.
- Se explica la conexión entre el celular y el centro de control, así como la conexión en el microcontrolador y el sensor de movimiento.
- Como ejemplo el sistema de alarma se desarrolla con un sensor PIR de fácil uso.
- Se explica el funcionamiento lógico del sistema de control, para poder realizar la programación en el microcontrolador.

1.2 Estudio de la realidad actual en sistemas de seguridad domiciliaria

El sector de la seguridad es razonablemente amplio, esto significa que la tecnología normalmente guarda una estrecha relación de calidad vs. Precio provocando así la presencia de un sinnúmero de sistemas y diseños que permitan mantener segura un área determinada.

En el mercado, existe una amplia oferta de productos relacionados con seguridad domiciliaria dentro de los cuales se pueden identificar cuatro áreas específicas de funciones y servicios que realizan todo sistema de seguridad, y estas son:

- Alarmas de Intrusión.
- Alarmas Técnicas (detectores de humo, sistemas de respaldo por falta de suministro eléctrico, etc.).
- Alarmas Personal (SOS y Asistencia Inmediata).
- Video Vigilancia.

Los sistemas de seguridad actuales pueden ser conectados a una Central Receptora de Alarmas (CRA) o ser monitoreados personalmente por el usuario desde un equipo portátil (celular, laptop, PDA, etc.). Este tipo de sistemas permiten realizar las siguientes acciones:

- Monitoreo de Alarmas.- El servicio consiste en un enlace vía red telefónica o vía Inalámbrica entre el sistema de alarma instalado en la propiedad y la central de monitoreo de Alarmas.
- Cerco Eléctrico.- Es un sistema de seguridad perimetral (cierre eléctrico o cierre electrificado) compuesto por una red de postes con un alambrado conectado a un energizador que convierte corriente alterna en pulsos eléctricos no letales de alto voltaje pero bajo amperaje.
- Cámaras.- Pueden ser visualizadas por el usuario desde cualquier lugar del mundo vía conexión Internet más conocido como CCTV.
- Simulación de Presencia.- Se programan para que enciendan la luz y/o la radio o televisión a una hora determinada, y con una intensidad previamente seleccionada. Asimismo, disponen de un temporizador para que se apaguen las luces y se desconecten los aparatos eléctricos asociados a una hora preestablecida por el usuario.

Sistema conectado a CRA

Si el usuario desea que su sistema sea mantenido y en casos de alarmas avise a una Central Receptora de Alarmas (CRA), el sistema debe cumplir con los siguientes requerimientos:

- El equipo y los dispositivos deben estar homologados para tal fin.
- El sistema de seguridad debe ser instalado por una empresa
- capacitada y avalada en este tipo de servicios.
- Este tipo de sistemas, en caso que se produzca un evento de intrusión, alarmas técnicas o pánico, siempre se conectan a la CRA para avisar del evento. Según el procedimiento acordado, el personal de la CRA confirma la alarma y avisan a la policía y/o al usuario, e inmediatamente acuden al sitio, según el tipo de contrato y evento.

Sistema de Monitorización Personal

Cualquier sistema de seguridad, esté o no homologado, puede ser instalado en una vivienda y configurado para que avise directamente al usuario o propietario de la misma. En este caso es el propio usuario quien gestiona quehacer en caso de que se produzca un evento en la vivienda. Cada vez más están saliendo al mercado productos o sistemas que pueden ser configurados para avisar directamente al usuario final. Además suelen incorporar la opción de darse alta en una CRA y según el tipo de evento, avisar a uno o a ambos.

Centrales cableadas o inalámbricas

Se pueden clasificar las centrales en dos tipos a nivel tecnológico:

- Centrales Cableadas: Todos los sensores y actuadores (sirenas, etc.), están cableados a la central, la cual es el controlador principal de todo el sistema. Esta tiene normalmente

una batería de respaldo, para en caso de fallo del suministro eléctrico, poder alimentar a todos sus sensores y actuadores y así seguir funcionando normalmente durante unas horas.

- Centrales Inalámbricas: En este caso usan sensores inalámbricos alimentados por pilas o baterías y transmiten vía radio la información de los eventos a la central, la cual está alimentada por redes eléctricas y tiene sus baterías de respaldo.

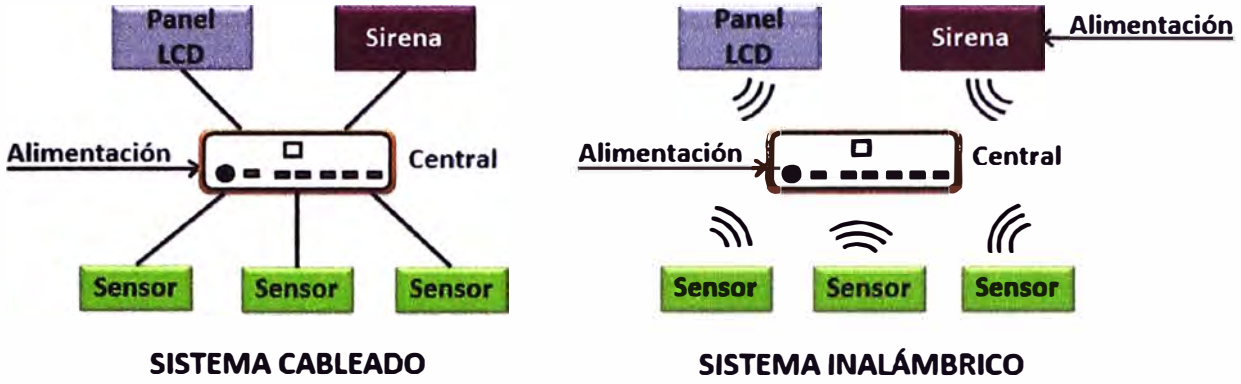

Figura 1.2 Topología de Sistemas de Seguridad [13]

En ambos casos, si desaparece el suministro eléctrico, el sistema seguirá funcionando unas horas, pero además suelen informar a la Central Receptora de Alarmas (CRA) o al usuario de dicho evento.

Por otro lado los sistemas de seguridad han empezado a incluir más funcionalidades, principalmente son de domótica para el control de la iluminación, climatización, accesos, etc., así como también de videovigilancia. Estas funcionalidades de actuar según eventos, o realizar eventos de domótica, convierte cada vez más los sistemas de seguridad en una parte central e imprescindible en el hogar digital.

CAPÍTULO II
FUNDAMENTOS PARA EL DESARROLLO DEL SISTEMA

En este capítulo se exponen las bases teóricas conceptuales relacionadas con el Lenguaje Java.

2.1 Generalidades del Java

El lenguaje de programación orientado a objetos desarrollado por Sun Microsystems a principio de los años 90 es llamado **Java**. Este lenguaje toma mucha de sus sintaxis de C y C++, con la ventaja de que el Java tiene un modelo de objetos más simple eliminando así herramientas de bajo nivel, que suelen tener muchos errores, como la manipulación directa de punteros o memoria.

Sun Microsystems en los primeros años de la década de los noventa decidió introducirse en el mercado de la electrónica de consumo y desarrollar programas para dispositivos electrónicos pequeños. Tras unos comienzos débiles, Sun decidió crear una sucursal, denominada FirstPerson Inc., para dar margen de maniobra al equipo responsable del proyecto.

Java se llamó Oak (roble en inglés) inicialmente, aunque tuvo que cambiar de denominación, debido a que dicho nombre ya estaba registrado por otra empresa en 1994. Se dicen muchas cosas sobre el origen de este nombre como: que este nombre se le puso debido a la existencia del árbol de java en los alrededores del lugar de trabajo de los promotores del lenguaje, otras suposiciones dicen que le pusieron ese nombre mientras tomaban café (Java es nombre de un tipo de café, originario de Asia), otros también afirman que el nombre deriva de las siglas de James Gosling, Arthur Van Hoff, y Andy Bechtolsheim, también se especula que el equipo de marketing organizó una reunión, un asesor de nomenclatura la dirigió y varios integrantes del grupo de Sun Microsystems , tuvieron un debate dando palabras al azar en la cual el primero en decir primero el nombre JAVA fue Mark Opperman. Sin embargo ninguna de ellas está confirmanda por algún documento que certifique este hecho.

El soporte para java fue incluido por Netscape en sus navegadores en 1995, produciendo una verdadera revolución en internet, dando con eso un buen apoyo a Java. Java es un lenguaje de programación, un entorno de desarrollo, un entorno de ejecución de aplicaciones, un entorno de despliegue de aplicaciones.

2.2 Versiones de Java

Existen versiones de Java de acuerdo a las necesidades de cada uno. Según esto se encuentra con que el paquete JAVA 2 se divide en 3 ediciones distintas, J2SE orientada al desarrollo de aplicaciones independientes de la plataforma, J2EE orientada al entorno empresarial y J2ME orientada a dispositivos con capacidades restringidas. A continuación se ven cuáles son las características de cada una de las versiones:

2.2.1 Java 2 Platform, Standard Edition (J2SE)

Es la edición original de JAVA, dirigida principalmente a computadores de uso personal, con herramientas básicas para desarrollar Applets, y con los aplicativos para realizar interfaces gráficas para los usuarios, redes, multimedia, etc.

Entre sus principales características se tienen:

- Inspirado inicialmente en C++, pero con componentes de alto nivel, como soporte nativo de strings y recolector de basura.
- Código independiente de la plataforma, precompilado a bytecodes intermedio y ejecutado en el cliente por una JVM (Java Virtual Machine).
- Modelo de seguridad tipo *sandbox* proporcionado por la JVM.
- Abstracción del sistema operativo subyacente mediante un juego completo de APIs de programación.

2.2.2 Java 2 Platform, Enterprise Edition (J2EE)

Software dirigido al entorno empresarial. Se ejecuta no sobre un computador particular, sino sobre una red de computadores, por lo que debe integrar datos que provienen de entornos no compatibles.

Debido a que las necesidades empresariales son diferentes a las de un usuario casero, se ha creado una extensión de JAVA para solucionar las necesidades de los empresarios.

Esta edición está orientada especialmente al desarrollo de servicios web, servicios de nombres, persistencia de objetos, XML, autenticación, APIs para la gestión de transacciones, etc. Tiene como objetivo ampliar la J2SE para dar soporte a los requisitos de las aplicaciones de empresa.

2.2.3 Java 2 Platform, Micro Edition (J2ME)

Esta versión de Java está enfocada a la aplicación de la tecnología Java en dispositivos electrónicos con capacidades computacionales y gráficas muy reducidas, tales como teléfonos móviles, PDAs o electrodomésticos inteligentes.

Esta edición tiene unos componentes básicos que la diferencian de las otras versiones, como el uso de una máquina virtual denominada KVM (Kilo Virtual Machine, debido a que requiere sólo unos pocos Kilobytes de memoria para funcionar) en vez del

uso de la JVM clásica.

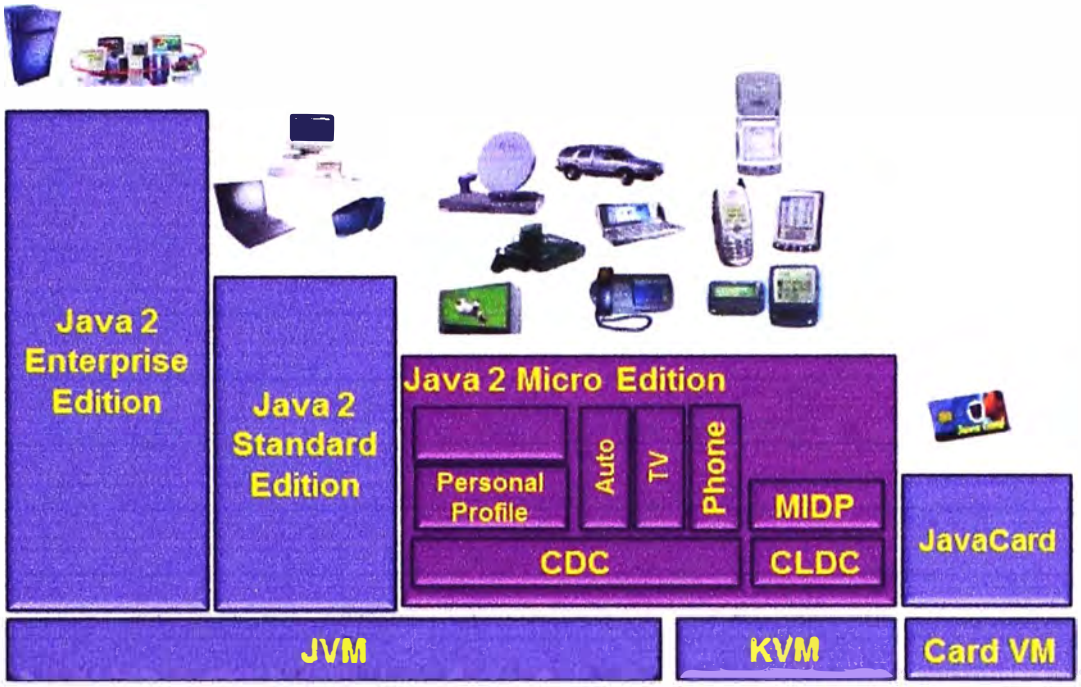

Figura 2.1 Arquitectura de la plataforma Java 2 de Sun. [2]

2.2.4 Aspectos comparativos

Dos características importantes de Java:

- El código fuente en lenguaje Java es compilado a código intermedio interpretado por una Java Virtual Machine (JVM), por lo que el código ya compilado es independiente de la plataforma.
- Todas las tecnologías comparten un conjunto más o menos amplio de APIs básicas del lenguaje, agrupadas principalmente en los paquetes java.lang y java.io.

J2EE es un superconjunto de J2SE pues contiene toda la funcionalidad de éste y más características, así como J2ME es un subconjunto de J2SE (excepto por el paquete javax.microedition) ya que contiene varias limitaciones con respecto a J2SE.

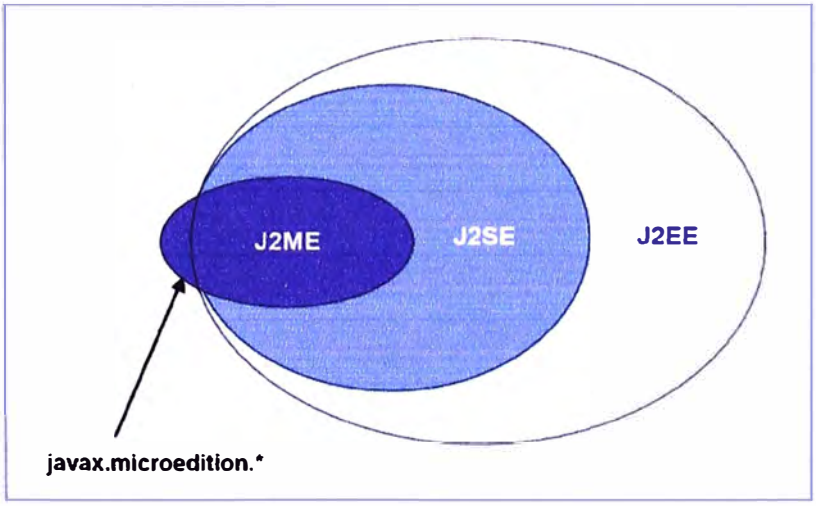

Figura 2.2 Relación entre las APIs de la plataforma Java. [2]

De manera sencilla se puede considerar a J2ME como una versión reducida de J2SE y J2EE como la versión ampliada de J2SE (ver Figura 2.2); en realidad cada una de las ediciones está enfocada a ámbitos de aplicación muy distintos. Las necesidades computacionales y APIs de programación requeridas para un juego ejecutándose en un móvil difieren bastante con las de un servidor distribuido de aplicaciones basadas en la arquitectura EJB (Enterprise JavaBeans) los cuales están destinados a ejecutarse en la parte servidora de una aplicación.

2.3 Nociones básicas de J2ME

Para poder empezar a crear un código que se pueda ejecutar en un dispositivo móvil, se debe entender el proceso de compilación y las herramientas usadas para tal fin. Es por esto que se agrupan todas esas cosas en un "entorno de ejecución". El entorno de ejecución debe tener entonces los siguientes elementos:

- Una máquina virtual
- Una configuración
- Un perfil
- Unos paquetes adicionales

El desarrollo de una aplicación utilizando el programa J2ME está basado en clases. Para asegurar la portabilidad del programa J2ME se utiliza una arquitectura en jerarquía de clases. Así un programa J2ME está basado en configuración, perfil y APIs opcionales. Cada uno de estos bloques de esta jerarquía contiene un determinado número de paquetes (conjunto de clases), los cuales se complementan en el desarrollo de una aplicación. En la Figura 2.3 se detalla de manera gráfica el concepto de la arquitectura de un programa J2ME.

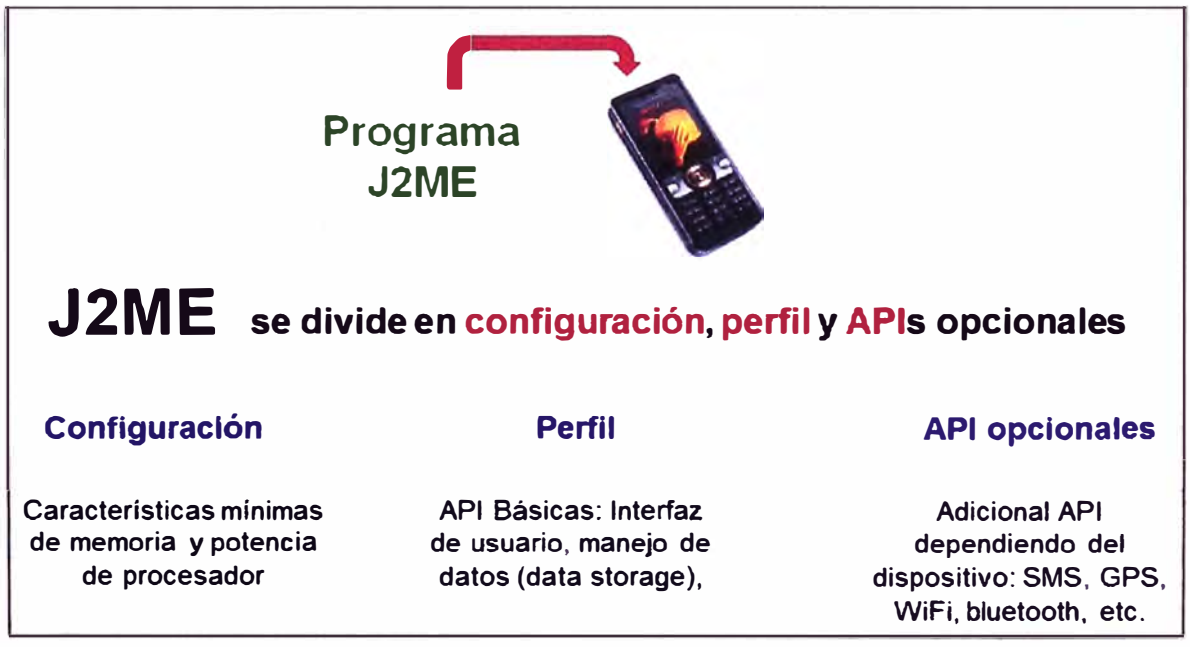

Figura 2.3 Arquitectura de un programa J2ME. [5]

2.3.1 Máquinas Virtuales J2ME

Existen 2 configuraciones definidas en J2ME: Connected Limited Device Configuration (CLDC) enfocada a dispositivos con restricciones de procesamiento y memoria, y Connected Device Configuration (CDC) enfocada a dispositivos con más recursos. Como consecuencia, cada una requiere su propia máquina virtual. La VM (Virtual Machine) de la configuración CLDC se denomina KVM y la de la configuración CDC se denomina CVM.

- KVM para CLDC.
- CVM para CDC

A continuación se describen las características principales de cada una de ellas:

a. KVM

Es la Máquina Virtual más pequeña desarrollada por Sun, su nombre KVM proviene de Kilobyte (haciendo referencia a la baja ocupación de memoria, entre 40Kb y 80Kb). Esta especialmente orientada a dispositivos con bajas capacidades computacionales y de memoria. La KVM está escrita en lenguaje C, aproximadamente unas 24000 líneas de código, y fue diseñada para ser:

- Pequeña, con una carga de memoria entre los 40Kb y los 80 Kb, dependiendo
- de la plataforma y las opciones de compilación.
- Alta portabilidad.
- Modulable.
- Lo más completa y rápida posible y sin sacrificar características para las que fue diseñada.

Sin embargo, esta baja ocupación de memoria hace que posea algunas limitaciones con respecto a la clásica *Java Virtual Machine* (JVM):

- No hay soporte para tipos en coma flotante. No existen por tanto los tipos double ni float. Esta limitación está presente porque los dispositivos carecen del hardware necesario para estas operaciones.
- No existe soporte para JNI (*Java Native Interface*) debido a los recursos limitados de memoria.
- No existen cargadores de clases (*class loaders*) definidos por el usuario. Sólo existen los predefinidos.
- No se permiten los grupos de hilos o hilos *daemon*. Cuando se quiera utilizar grupos de hilos se utilizarán los objetos *Colección* para almacenar cada hilo en el ámbito de la aplicación.
- No existe la finalización de instancias de clases. No existe el método Object.finalize().
- Limitada capacidad para el manejo de excepciones debido a que el manejo de éstas

depende en gran parte de las APIs de cada dispositivo por lo que son éstos los que controlan la mayoría de las excepciones.

Debido a que el verificador de clases de J2SE es muy pesado para la verificación de clases en J2ME, la implementación del verificador de clases en KVM requiere de al menos 10 kB de código binario y menos de 100 bytes de RAM dinámica en tiempo de ejecución para clases típicas. El verificador realiza solamente un escaneo lineal del código intermedio, sin la necesidad de un algoritmo de flujo de datos iterativo costoso. Así el nuevo verificador de clases opera en dos fases como se ilustra en la Figura 2.4.

En la fase número 1, las clases tienen que ser ejecutadas a través de una herramienta de pre-verificación para poder remover ciertos códigos intermedios y aumente las clases con atributos adicionales llamados StackMap aumentando la velocidad de verificación en tiempo de ejecución. La fase de preverificación es realizada típicamente en una estación de trabajo que el desarrollador usa para escribir y compilar aplicaciones.

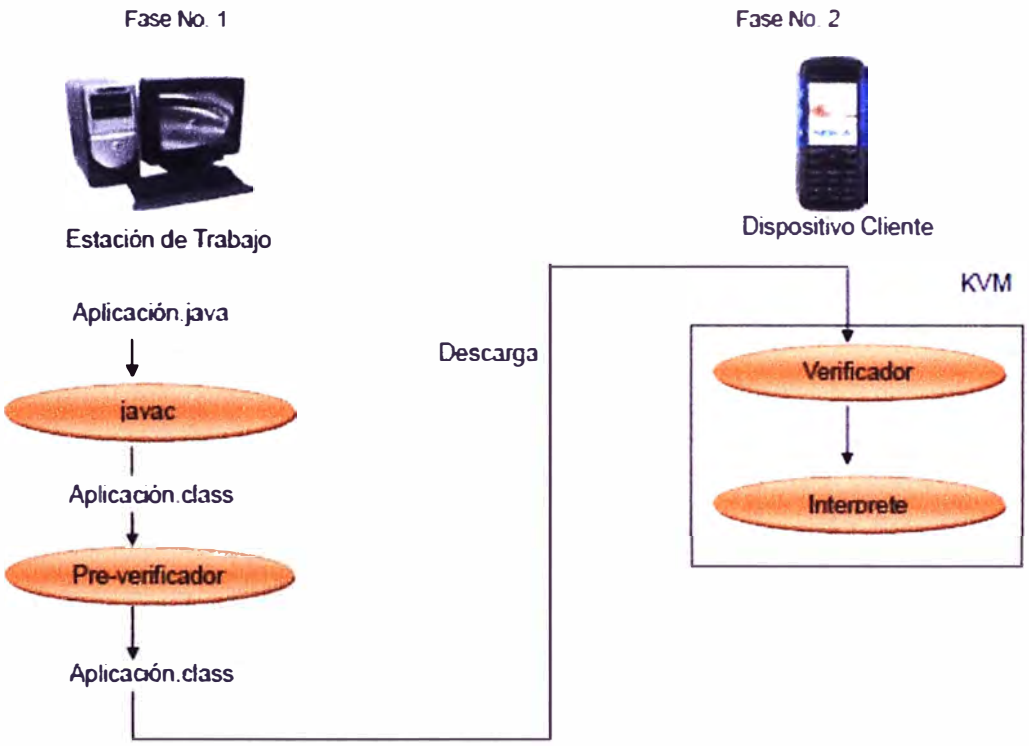

Figura 2.4 Preverificación de clases en CDLC/KVM. [2]

En la fase número 2, el componente verificador en tiempo de ejecución de la máquina virtual usa los atributos adicionales StackMap generados por el preverificador para realizar la preverificación de la clase actual de forma eficiente. A través de estas fases se realizan las siguientes comprobaciones:

- Observar que el código no sobrepase los límites de la pila de la Máquina Virtual.
- Comprobar que no se utilizan variables locales antes de ser inicializadas.

- Comprobar que se respetan los campos, métodos y los modificadores de control de acceso a clases.

b. CVM

La CVM (Compact Virtual Machine) ha sido tomada como Máquina Virtual Java de referencia para la configuración CDC y soporta las mismas características que la Máquina Virtual de J2SE. Está orientada a dispositivos electrónicos con procesadores de 32 bits de gama alta y en torno a 2Mb o más de memoria RAM. Las características que presenta esta Máquina Virtual son:

- Sistema de memoria avanzado.
- Tiempo de espera bajo para el recolector de basura.
- Separación completa de la VM del sistema de memoria.
- Recolector de basura modularizado.
- Portabilidad.
- Rápida sincronización.
- Ejecución de las clases Java fuera de la memoria de sólo lectura (ROM).
- Soporte nativo de hilos.
- Baja ocupación en memoria de las clases.
- Proporciona soporte e interfaces para servicios en sistemas operativos de tiempo real.
- Conversión de hilos Java a hilos nativos.
- Soporte para todas las características de Java2 v1.3 y librerías de seguridad, referencias débiles, Interfaz Nativa de Java (JNI), invocación remota de métodos (RMI), Interfaz de depuración de la Máquina Virtual (JVMDI).

2.3.2 Configuraciones

Una configuración es el conjunto mínimo de APIs Java que permiten desarrollar aplicaciones para un grupo de dispositivos. Éstas APIs describen las características básicas, comunes a todos los dispositivos:

- Características soportadas del lenguaje de programación Java.
- Características soportadas por la Máquina Virtual Java.
- Bibliotecas básicas de Java y APIs soportadas.

Como ya se ha visto con anterioridad, existen dos configuraciones en J2ME:

- CLDC, orientada a dispositivos con limitaciones computacionales y de memoria
- CDC, orientada a dispositivos con no tantas limitaciones..

a. CDC (Conected device configuration)

Es la configuración dirigida a dispositivos con cierta capacidad computacional. Usa la CVM como máquina virtual, que es muy similar a la máquina virtual usada por J2SE. Los paquetes que posee esta configuración se pueden observar en la Tabla 2.1

Tabla 2.1 Paquetes de CDC [2]

Nombre	Descripción
java.io	Clases e interfaces estándar de E/S
java.lang	Clases básicas del lenguaje
java.lang.ref	Clases de referencia
java.lang.reflect	Clases e interfaces de reflexión
java.math	Paquete de matemáticas
java.net	Clases e interfaces de red
java.security	Clases e interfaces de seguridad
java.security.cert	Clases de interfaces de seguridad
java.text	Paquete de texto
java.util	Clases de utilidad estándar
java.util.jar	Clases y utilidades para archivos JAR
java.util.zip	Clases y utilidades para archivos ZIP y comprimidos
javax.microedition	Clases e interfaces para conexión genérica CDC

Cada uno de estos paquetes son las librerías que permite la CVM usar a través de las APIs correspondientes.

Entre estos dispositivos se tienen:

- Decodificadores de televisión digital.
- Televisores con Internet.
- Algunos electrodomésticos.
- Sistemas de navegación en automóviles, etc.

Estos dispositivos deben cumplir los siguientes requisitos:

- Procesador de 32 bits.
- Disponer de 2MB o más de memoria total.
- Poseer la funcionalidad completa de la Máquina Virtual Java2.
- Conectividad a algún tipo de red.

b. CLDC (Conected limited device configuration)

Es la configuración dirigida a dispositivos con muy pocas capacidades de procesamiento, y usa la máquina KVM. Entre estos dispositivos se tienen:

- Teléfonos móviles.
- Buscapersonas (pagers).
- PDAs.
- Organizadores personales, etc.

Estos dispositivos deben cumplir los siguientes requisitos:

- Disponer entre 160 y 512 KB de memoria total disponible. 128
- KB como mínimo de memoria no volátil para la MVJ y las bibliotecas CLDC, y 32 KB de

memoria volátil para la MV entiempo de ejecución.

- Procesador de 16 ó 32 bits con al menos 25MHz de velocidad.
- Bajo consumo, debido a su funcionamiento con baterías.
- Conexión a alguna red, normalmente sin cable, con conexión intermitente y ancho de banda limitado (9600bps).

Las librerías admitidas para esta configuración son las mostradas en la Tabla 2.2.

Tabla 2.2 Librerías admitidas. [2]

Nombre	Descripción
java.io	Clases e interfaces estándar de E/S. Subconjunto de J2SE
java.lang	Clases e interfaces de la Máquina Virtual. Subconj. de J2SE
java.util	Clases, interfaces y utilidades estándar. Subconjunto de J2SE
javax.microedition	Clases e interfaces de conexión genérica CDLC

2.3.3 Perfiles

Es un conjunto de APIs que se encuentran orientados a un determinado tipo de actividades o tareas. A través de un perfil, se puede identificar la funcionalidad de un dispositivo, pues las APIs que contiene lo hacen pertenecer a un grupo de dispositivos con una aplicación similar.

Mientras que un perfil define las características de un dispositivo predeterminado, una configuración lo hace pertenecer a una familia de dispositivos. Esto significa que cuando se propone la creación de una aplicación, se tienen tanto las APIs de perfil como las de configuración.

Acá lo más importante es que los perfiles no son aislados, sino que dependen primordialmente de la configuración. Es decir, una configuración carece de funcionalidad si no tiene asignado un perfil determinado. Se puede decir entonces que un perfil es un conjunto de APIs que dotan a una configuración de una utilidad y una funcionalidad.

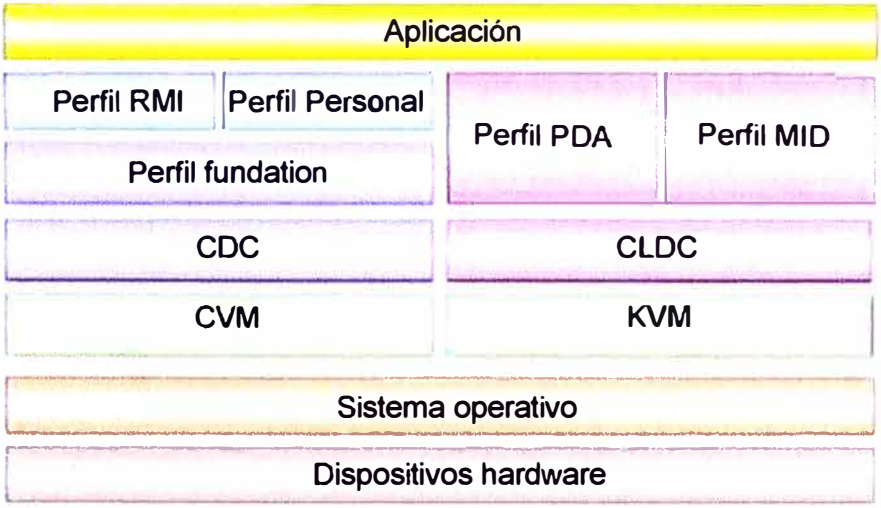

Figura 2.5 Capas del entorno de ejecución. [2]

La Figura 2.5 ayuda entender mejor el entorno de ejecución de J2ME.

Los perfiles al igual que la máquina virtual están predeterminados por la configuración que se pretende usar en el desarrollo de una aplicación J2ME.

A continuación se describen las características básicas de los perfiles que se ejecutan sobre las configuraciones.

Perfiles de la configuración CDC:

- Foundation profile
- Personal profile
- RMI profile.

Perfiles de la configuración CLDC:

- PDA profile
- Mobile information Device Profile (MIDP)

a. Foundation Profile

Este perfil define una serie de APIs sobre la CDC orientadas a dispositivos que carecen de interfaz gráfica como, por ejemplo, decodificadores de televisión digital.

Este perfil incluye gran parte de los paquetes de la J2SE, pero excluye totalmente los paquetes "java.awt" *Abstract Windows Toolkit* (AWT) y "java.swing" que conforman la interfaz gráfica de usuario (GUI) de J2SE. Si una aplicación requiriera una GUI, entonces sería necesario un perfil adicional.

Los paquetes que forman parte del *Foundation Profile* se muestran en la Tabla 2.3.

Tabla 2.3 Paquetes del Foundation Profile. [2]

Nombre	Descripción
java.lang	Soporte del lenguaje Java
java.net	Incluye sockets TCP/IP y conexiones HTTP
java.security	Incluye códigos y certificados
java.text	Incluye soporte para internacionalización
java.util	Añade soporte completo para zip y otras funcionalidades (java.util.Timer)
java.io	Clases Reader y Writer de J2SE

b. Personal Profile

Este perfil es el encargado de proporcionar un entorno gráfico completo y funcional a un dispositivo. Le proporciona capacidades web, soporte de Applets de java y, sobre todo, es un entorno con soporte gráfico AWT, que permite mediante coordenadas, dibujar objetos en la pantalla del dispositivo. Para la ejecución de este perfil, debe implementarse previamente el foundation profile. Los paquetes que contiene la "personal profile" se muestran en la Tabla 2.4.

Tabla 2.4 Paquetes del Personal Profile. [2]

Nombre	Descripción
java.applet	Clases necesitadas para crear applets o que son usadas por ellos
java.awt	Clases para crear GUIs con AWT
java.awt.datatransfer	Clases e interfaces para transmitir datos entre aplicaciones
java.awt.event	Clases e interfaces para manejar eventos AWT
java.awt.im	Clases e interfaces para definir métodos editores de entrada
java.awt.im.spi	Interfaces que añaden el desarrollo de métodos de editores de entrada para cualquier entorno de ejecución Java
java.awt.image	Clases para crear y modificar imágenes
java.beans	Clases que soportan JavaBeans
javax.microedition	Interfaces que usa el Personal Profile para la comunicación

c. RMI profile

Este perfil es un subconjunto de un perfil de J2SE, con diversas limitaciones debido a las capacidades computacionales de los dispositivos móviles que implementan la CVM. La RMI profile es un subconjunto del perfil RMI para J2SE, que necesita el foundation profile, y que no posee las siguientes propiedades del RMI de J2SE:

- Java.rmi.server.disableHTTP.
- Java.rmi.activation.port.
- Java.rmi.loader.packagePrefix.
- Java.rmi.registry.packagePrefix.
- Java.rmi.server.packagePrefix.

d. PDA profile

Como su nombre lo indica, es un perfil construido para PDAs, sobre todo las de gama baja, como palms con una pantalla y un puntero. En sus inicios, soportaba una resolución de al menos 20000 pixeles de pantalla y la presencia de un puntero, todo esto sobre CLCD. Actualmente es una configuración dirigida al manejo de información personal, como agendas, y accede al calendario y a la base de datos de contactos para hacer aplicaciones con ella.

e. Mobile Information Device Profile (MIDP)

- Dispositivos con reducida capacidad computacional.
- Conectividad limitada (9600 bps).
- Capacidad gráfica muy reducida (mínimo display de 96x54 pixel monocromo).
- Entrada de datos alfanumérica reducida.
- 128 Kb de memoria no volátil para componentes MIDP.
- 8 Kb de memoria no volátil para datos persistentes de aplicaciones.

- 32 Kb de memoria volátil en tiempo de ejecución para la pila Java.

APIs relacionadas con:

- La aplicación (semántica y control de la aplicación MIDP).
- Interfaz de Usuario.
- Almacenamiento persistente.
- Trabajo en red.
- Temporizadores.

Los paquetes que forman parte del *Device Profile* se muestran en la Tabla 2.5.

Tabla 2.5 Paquetes del Personal Profile. [2]

Nombre	Descripción
java.microedition.lcdui	Clases es interfaces para GUIs
java.microedition.rms	*Record Management Storage.* Soporte para el almacenamiento persistente del dispositivo
java.microedition.midlet	Clases de definición de la aplicación
java.microedition.io	Clases e interfaces de conexión genérica
java.io	Clases e interfaces de E/S básica
java.lang	Clases e interfaces de la máquina virtual
java.util	Clases e interfaces de utilidad estándar

Las aplicaciones que se realizan utilizando MIDP reciben el nombre de *MIDlets* (por simpatía con *APPlets*). Se dice así que un MIDlet es una aplicación Java realizada con el perfil MIDP sobre la configuración CLDC.

2.3.4 Paquetes opcionales

Muchos dispositivos Java ME incluyen tecnologías adicionales como Bluetooth, multimedia, mensajes inalámbricos, o conectividad a bases de datos. Para aprovechar estas tecnologías mediante típicas APIs de Java, existen paquetes adicionales a los ya mencionados. Los fabricantes de los dispositivos pueden añadir estos paquetes según se necesite para utilizar las distintas posibilidades del dispositivo.

Además de las configuraciones, perfiles y paquetes opcionales, los fabricantes pueden definir nuevas clases Java para aprovechar características propias de cada dispositivo. Estas clases se denominan Clases de Licencia Abierta (LOCs). Una LOC define clases disponibles para todos los desarrolladores. Las Clases de Licencia Cerrada (LCCs) definen clases que sólo están disponibles para el fabricante del dispositivo. Los programas que usen estas clases puede que pierdan la característica de portabilidad, incluso entre dispositivos que tengan la misma configuración y perfil.

2.3.5 MIDlets

Como ya se ha citado una aplicación creada con el perfil MIDP se denomina MIDlet. Esta sección pretende dar unas nociones básicas sobre los MIDlets, viendo cuáles son

sus propiedades, y conociendo cómo es su ciclo de vida y estados por los que puede pasar. Además se dará una rápida introducción al paquete Java dedicado a la creación de MIDlets, exponiendo además un ejemplo básico.

En los dispositivos susceptibles de ejecutar MIDlets, existe un software encargado de gestionar estas aplicaciones denominado Gestor de Aplicaciones (AMS). Este software será el que permita ejecutar, pausar y destruir las aplicaciones. ElAMS realiza dos funciones principales, gestionar el ciclo de vida de los MIDlets y controlar los estados por los que pasa mientras está en ejecución.

a. Ciclo de vida

El ciclo de vida de un MIDlet consta de 5 fases: descubrimiento, instalación, ejecución, actualización y borrado.

- **Descubrimiento:** Es la fase en la que se selecciona, con ayuda del AMS, la aplicación a descargar para ser instalada. El AMS proporciona varias formas de descarga de la aplicación, que se podrán llevar o no a cabo según las posibilidades del dispositivo. Por ejemplo, se puede realizar conectando el dispositivo a un PC mediante un cable o de forma inalámbrica vía Bluetooth.
- **Instalación:** Una vez descargada la aplicación, se pasa al proceso de instalación del mismo, controlado por el AMS que informará al usuario de cómo se va desarrollando todo el proceso. Cuando el MIDlet está instalado, todas las clases, archivos y almacenamiento persistente del dispositivo están a su disposición.
- **Ejecución:** La ejecución de la aplicación se podrá realizar con ayuda del AMS, que además gestionará los estados en los que se encuentra el MIDlet, en función de los eventos que se produzcan.
- **Actualización:** El AMS debe de ser capaz de detectar si un MIDlet que se descargaes una actualización de uno ya existente en el dispositivo e informar al usuario de lo mismo.
- **Borrado:** También el AMS es encargado de borrar el MIDlet del dispositivo, pidiendo antes confirmación al usuario e informando de todo el proceso.

b. Estados de un MIDlet en ejecución

Un MIDlet durante su ejecución se puede encontrar en tres estados diferentes: activo, suspendido y destruido. Como ya se ha citado los estados son gestionados por el AMS. El cambio de un estado a otro lo puede realizar el mismo MIDlet llamando al método adecuado, aunque también lo puede realizar el mismo AMS porque lo vea conveniente. Por ejemplo, si el MIDlet se ejecuta en un teléfono y recibe una llamada o un SMS, entonces el AMS suspenderá el MIDlet. En general, se cambia de un estado a otro llamando a los métodos startApp(), pauseApp() o destroyApp(). En la Figura 2.6 se muestra cuales son las transiciones posibles de un estado a otro y mediante qué métodos

se pueden realizar estas transiciones.

Figura 2.6 Transiciones y métodos posibles de un estado a otro. [12]

En una ejecución típica el MIDlet pasará por los tres estados, y normalmente lo hará de la siguiente manera. En primer lugar se realiza la llamada al constructor del MIDlet pasando al estado suspendido durante un corto período de tiempo.

El AMS por su parte crea una nueva instancia del MIDlet. Cuándo el dispositivo está preparado para ejecutar el MIDlet, el AMS invoca al método startApp() para entrar en el estado activo, en el que el MIDlet ocupará todos los recursos necesarios para su ejecución.

Durante este estado el MIDlet puede pasar al estado suspendido por una acción del usuario, o bien por el AMS que reduciría en todo lo posible el uso de los recursos del dispositivo por parte del MIDlet.

Esto se realizaría llamando al método pauseApp() y, para volver al estado activo, habría que llamar de nuevo a startApp(). Tanto en el estado activo como suspendido, el MIDlet puede pasar al estado destruido realizando una llamada al método destroyApp().

Esto puede ocurrir porque el MIDlet haya finalizado su ejecución o porque una aplicación prioritaria necesite ser ejecutada en memoria en lugar del MIDlet. Una vez destruido el MIDlet se liberan todos los recursos ocupados.

c. Estructura de un MIDlet

Una vez visto el ciclo de vida y los estados de un MIDlet se va a ver cómo sería su estructura básica. Así se verá someramente la API empleada para crear un MIDlet. Tampoco se entrará en demasiado detalle pues, a pesar de que en el proyecto se emplea, no es el objetivo primordial de éste.

Los paquetes que se usan principalmente para la creación de un MIDlet son:

- **javax.microedition.midlet**, que sería el paquete fundamental para la creación de un MIDlet y que define las relaciones de éste con su entorno.,

- **Javax.microedition.lcdui**, que provee una API para la implementación de interfaces gráficas de usuario.

c.1. Clase MIDlet

Para la creación de una aplicación MIDP se tiene que partir de la clase MIDlet, ya que cualquier MIDlet debe heredar esta clase. Esta clase se la puede encontrar en el paquete javax.microedition.midlet.

Los métodos de esta clase están bastante ligados a los estados del MIDlet, anteriormente mencionados. Por un lado contiene los métodos startApp(), pauseApp(), y destroyApp(), que se ejecutarán al entrar en los estados correspondientes.

Otros métodos, como notifyDestroyed() o notifyPaused(), indican al AMS que el MIDlet desea entrar en el estado destruido y suspendido respectivamente. Para más detalles sobre el resto de los métodos consultar la especificación de MIDP 2.0.

Además de la clase MIDlet en este paquete se encuentra la excepción midletStateChangeException, que es lanzada cuando no es posible cambiar de un estado a otro.

c.2. Interfaces gráficas de usuario

La regla básica del funcionamiento de un MIDlet es visualizar determinadas informaciones y dirigir consultas al usuario en forma de una serie de formularios sencillos, visualizados en la pantalla del dispositivo. El control de esta pantalla se basa en la clase Display, que representa un manejador de la pantalla y los dispositivos de entrada. Todo MIDlet debe poseer como mínimo un objeto de esta clase, que se obtendrá llamando al método Display.getDisplay(). La llamada a este método se deberá realizar en el constructor del MIDlet, para asegurar que exista desde el principio de la ejecución. En la Figura 2.7 se puede ver la jerarquía de clases derivada de Display.

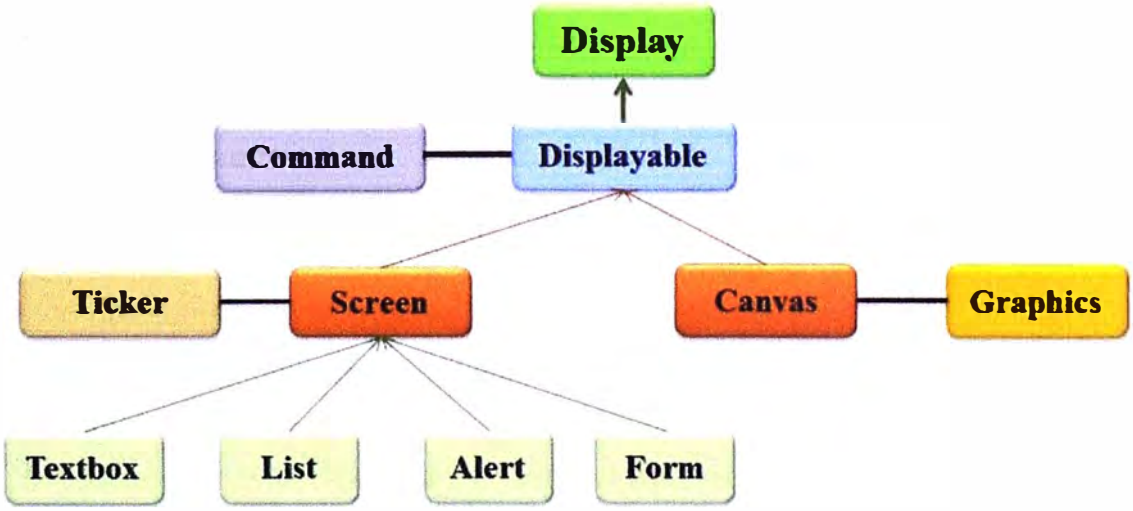

Figura 2.7 Jerarquía de clases derivada de Display. [12]

Un MIDlet tendrá en general distintas pantallas que irán alternándose en el dispositivo mostrando distintas informaciones al usuario. La clase Displayable representa las pantallas de la aplicación, que podrán ser de diferente tipo. Para cambiar de una a otra el objeto Display proporciona el método setCurrent.

Dentro de la clase Displayable se encuentra la clase Screen que representa las pantallas de alto nivel, que proporcionan componentes típicos para la interfaz de usuario como formularios, cajas de texto, botones, etc. Éstas no dejan mucha libertad en cuanto al aspecto se refiere, ya que éste depende más que nada de la implementación, pero lo que si aporta es una gran portabilidad entre los distintos dispositivos en los que se puede ejecutar.

Por otro lado existen las pantallas de bajo nivel, englobadas dentro de la clase Canvas, que dan un control completo sobre todo lo que aparece en pantalla. Además ofrecen un control total sobre los recursos del dispositivo y se podrán controlar eventos de más bajo nivel. Estas pantallas son necesarias sobre todo en el desarrollo de juegos.

Para la realización de este proyecto interesarán principalmente las de alto nivel. Por ello a continuación se detallan las distintas pantallas de este tipo, derivadas de la clase Screen:

- **Alert:** La clase alerta muestra un aviso al usuario, a la que se le puede configurar un cierto tiempo de espera antes de mostrar la siguiente pantalla. Una alerta puede contener un título, texto y una imagen opcional. Su propósito fundamental es el de informar de errores u otras situaciones excepcionales.
- **List:** Permite construir pantallas que poseen una lista de opciones. Esto será muy útil para crear menús de manera independiente.
- **TextBox:** Esta clase es una pantalla que permite al usuario escribir o editar un texto.
- **Form:** Es una pantalla que puede contener muchos elementos de distinto tipo, como imágenes, textos, campos editables de texto, barras de progreso o grupos de selección. Estos elementos pertenecerán a la clase Item.

c.3 Soporte de eventos

Las clases que van a dedicarse al control de interacciones por parte del usuario deben implementar la interfaz CommandListener, para lo que tendrán que definir su método commandAction(), que será llamado cuando la pantalla activa en ese momento reciba un evento. En este método se llevará a cabo el código necesario según el comando que se reciba.

Los comandos serán instancias de la clase Command, que tiene algunas propiedades interesantes. En general, se puede decir que un comando mantiene información sobre un evento. Serían equivalentes a los botones de Windows por establecer una analogía. La

especificación MIDP supone en el dispositivo un máximo de tres botones. Pero se pueden añadir muchos más comandos, y el dispositivo los irá agrupando entre estos tres botones y menús emergentes de la mejor forma posible.

Al crear un comando, además de especificar la etiqueta, que será el nombre que se muestre en pantalla, habrá que indicar su tipo y prioridad. El tipo indicará como será la acción de ese comando. Así BACK se utiliza para volver a la pantalla anterior o EXIT para salir de la aplicación en curso. Otros tipos posibles son CANCEL, HELP, ITEM, OK, SCREEN, y STOP. Por otro lado, la prioridad también será bastante útil, ya que podrá servir por ejemplo al AMS a la hora de colocar los comandos en la pantalla.

CAPÍTULO III
DISEÑO DE LOS PROGRAMAS REQUERIDOS POR EL SISTEMA

En el presente capítulo se explican los paquetes que fundamentales para el desarrollo del subsistema, los cuales son WMA y MMAPI, estas dos herramientas son necesarias para implementar el envío de MMS y la captura de fotos con el dispositivo móvil. También se desarrollan cada una de las partes fundamentales que componen los códigos de los MIDlets MMS y FotoControl, el código competo se colocará en los Anexos A y B respectivamente.

3.1 Mensajería inalámbrica

En la actualidad la generación de SMS ha evolucionado a Enhanced Message Service (EMS servicio de mensajería mejorado) y a Multimedia Message Service (MMS Servicio de Mensajes Multimedia), que no se limitan a la transmisión de solo texto.

3.1.1 WMA - JSR 120 Wireless Messaging API 1.1

Es un paquete opcional para la configuración CLDC, la WMA 1.1 se publicó por primera vez en agosto de 2002, teniendo como objetivo brindar las especificaciones necesarias para el envío y recepción de mensajes. El nombre del paquete que contiene las principales interfaces es javax.wireless.messaging, aunque también hace uso del paquete javax.microedition.io correspondiente al *Generic Connection Framework* (GCF) que soporta operaciones de entrada-salida y conexiones de red.

En su forma más simple un mensaje está compuesto por la dirección del destinatario y los datos a enviar, en cuanto a la dirección la interfaz más general es Message y contiene métodos para las direcciones y marcas de tiempo. Para la parte de datos WMA 1.1 soporta dos tipos, los datos de texto mediante la interfaz TextMessage y datos binarios mediante interfaz BinaryMessage, estas interfaces son descendientes directas de Message, y ambas emplean ciertos métodos para manejar el contenido del mensaje.

Antes de realizar la transmisión de mensajes es necesario abrir una conexión en modo cliente o servidor, para ello se implementa MessageConnection obteniendo un objeto a partir de la clase Connector con su método Open, al cual se le pasa la URL como parámetro, si ésta se encuentra en forma completa se abre en modo cliente y solo se pueden enviar mensajes, si se especifica el puerto relacionado con la aplicación se abre en modo servidor y es posible tanto enviar como recibir mensajes. Cabe mencionar

que en la URL se incluye el protocolo, ya sea SMS o CBS (Cell Broadcast Service).

La interfaz MessageConnection consta de métodos para el envío y recepción de mensajes, para la recepción de mensajes utiliza un evento escuchador llamado MessageListener. Asociado con los métodos existen una serie de errores posibles como por ejemplo SecurityException.

En la Figura 3.1 se muestra la estructura de interfaces y clases junto a sus relaciones para la JSR 120.

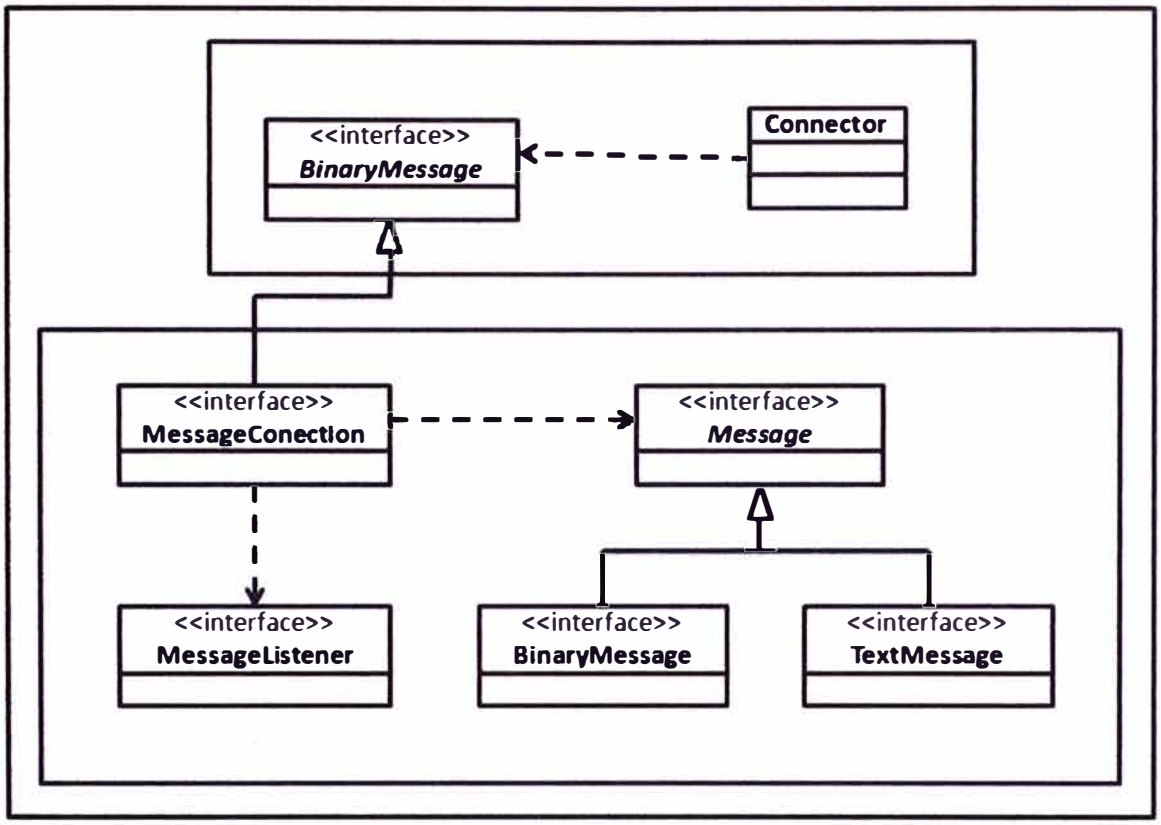

Figura 3.1 Estructura de la JSR 120. [6]

La Tabla 3.1 presenta un resumen de las interfaces y clases junto con sus métodos.

Tabla 3.1 Interfaces y clases junto con sus métodos. [6]

Interfaz/Clase	Descripción	Método
Message	Interfaz base de la cual heredan TextMessage y BinaryMessage, entre otras. Los protocolos de mensajería que son accedidos a través de esta interfaz son del tipo guardar y enviar a diferencia de los datagramas, por eso es definida nuevamente.	getAddress(), getTimestamp(), setAddress()
BinaryMessage	Interfaz que proporciona los métodos para obtener y establecer el contenido binario. Los objetos que la implementan son simples contenedores.	getPayloadData(), setPayloadData()

TextMessage	Interfaz que proporciona los métodos para obtener y establecer el contenido de texto. Los objetos que la implementan son simples contenedores.	getPayloadText(), setPayloadText()
Connector	Clase utilizada para crear objetos del tipo Connection	open()
MessageConnection	Interfaz que define la funcionalidad básica para enviar y recibir mensajes. Contiene métodos de fabricación para crear nuevos mensajes y para calcular el número de segmentos necesarios según protocolo subyacente. Es instanciada al llamar Connector.open().	newMessage(), receive(), send(), setMessageListener(), numberOfSegments()
MessageListener	Interfaz que provee un mecanismo para la notificación de mensajes entrantes.	notifyIncomingMessage()

Las interfaces y clases mostradas brindan un mecanismo general para establecer una aplicación de mensajería, pero también es necesario conocer los formatos para las URL de acuerdo con el portador específico del mensaje.

En la documentación de la JSR se encuentra detallados los formatos para el envío de SMS y CBS a través de GSM, y CDMA solo en el caso del primero.

Con respecto a la seguridad, para el caso MIDP 1.0 la JSR no especifica ningún mecanismo formal, y en muchas situaciones es el usuario final quien define la opción de conceder o denegar acceso a los recursos.

Para el MIDP 2.0 los permisos deben concederse al momento de abrir una conexión o al enviar y recibir mensajes.

3.1.2 WMA 2.0 - JSR 205 Wireless Messaging API 2.0

WMA 2.0 constituye una evolución sobre la versión WMA 1.1 y se publicó en junio de 2004, su mayor aporte radica en el soporte para mensajes de múltiples partes que se emplean en el envío de Mensaje Multimedia, siendo esta la principal diferencia entre ambas.

La WMA 2.0 cuenta con la subinterfaz MultipartMessage, la cual deriva directamente de Message, y representa un mensaje que contiene múltiples partes como en el caso de los mensajes multimedia, definiendo un contenedor de una o más MessageParts, y proporciona métodos para manejar las direcciones del emisor y el receptor, la cabecera de los mensajes, la identificación ID del primer mensaje y las partes del mensaje.

La Tabla 3.2 muestra la interfaces y clases agregadas, junto con un resumen de sus respectivos métodos. La Figura 3.2 ilustra la estructura de interfaces y clases, junto a sus

relaciones para la JSR 205.

Tabla 3.2 Resumen de interfaces y clases con sus métodos para la JSR 205. [6]

Interfaz/Clase	Descripción	Método
MultipartMessage	Es una subinterfaz de Message que contiene métodos para manipular las MessageParts. Además permite especificar elementos propios de un mensaje multiparte como por ejemplo el asunto, las direcciones CC o BCC entre otros.	addAddress() addMessagePart() getAddress() getAddresses() getHeader() getMessagePart() getMessageParts() getStartContentId() getSubject() removeAddress() removeAddresses() removeMessagePart() removeMessagePartId() removeMessagePartLocation() setAddress() setHeader() setStartContentId() setSubject()
MessagePart	Sus instancias pueden ser agregadas a un MultipartMessage. Cada MessagePart posee el elemento contenido, el tipo MIME, el content-id. El contenido puede ser de cualquier tipo.	getContent() getContentAsStream() getContentID() getContentLocation() getEncoding() getLength() getMIMEType()

La Tabla 3.2 muestra la interfaces y clases agregadas, junto a sus respectivos métodos.

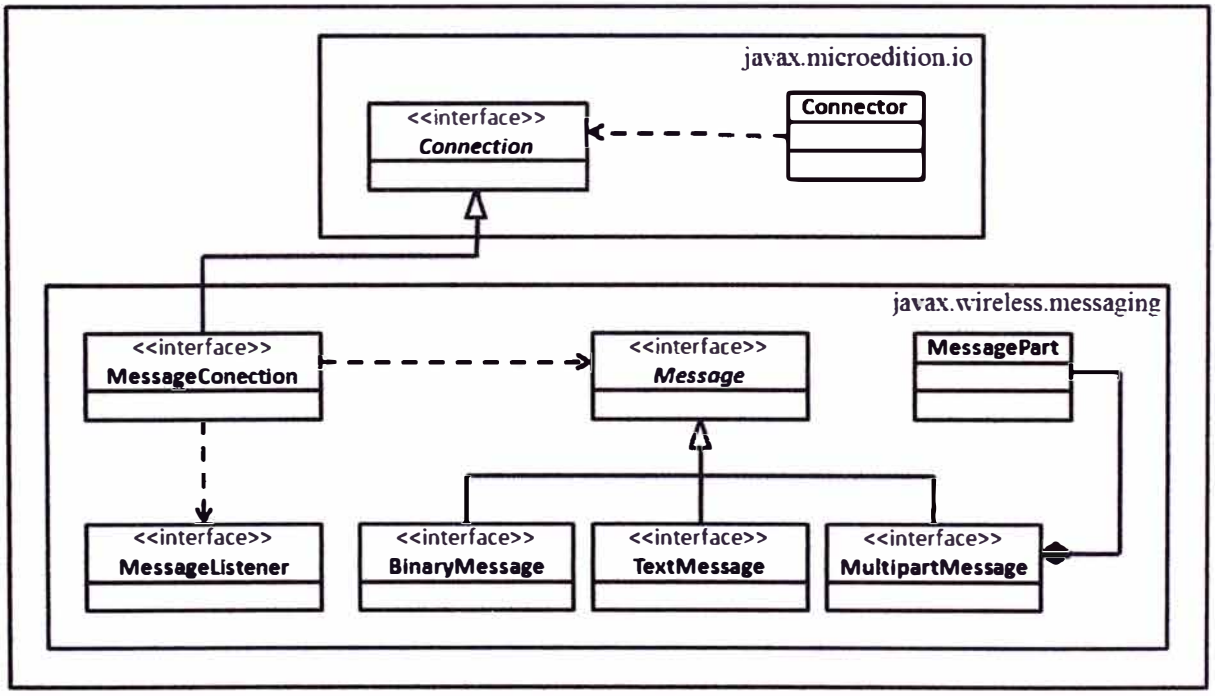

Figura 3.2 Estructura de la JSR 205. [6]

3.2 Análisis de las partes que componen el MIDlet MMS

El API de mensajería inalámbrica tiene sus bases en la definición del entorno (GCF) *Generic Connection Framework* descrito en la especificación CLDC y provee al desarrollador soporte para:

- Short Messaging Service (SMS), para el envío y recepción de mensajes cortos de texto, que soporta conexiones servidor y cliente.
- Cell Broadcast Service (CBS), para la recepción de mensajes enviados en forma Broadcast, que soporta solamente conexiones cliente.

3.2.1 Componentes del API MMS

La interfaz Message determina propiedades comunes a todos los mensajes. Cada método tiene una dirección y un sello de tiempo asociado a él. En un mensaje la dirección es la encargada de indicar el destinatario de ese mensaje.

El formato de la dirección depende del protocolo utilizado para enviar el mensaje (SMS ó CBS), donde una dirección se especifica como una dirección URL con la estructura:

sms:// número _de_teléfono:puerto

sms://+34 950 111 222 sms://:3456

sms://+34 950 111 222:3456 cbs://:3456

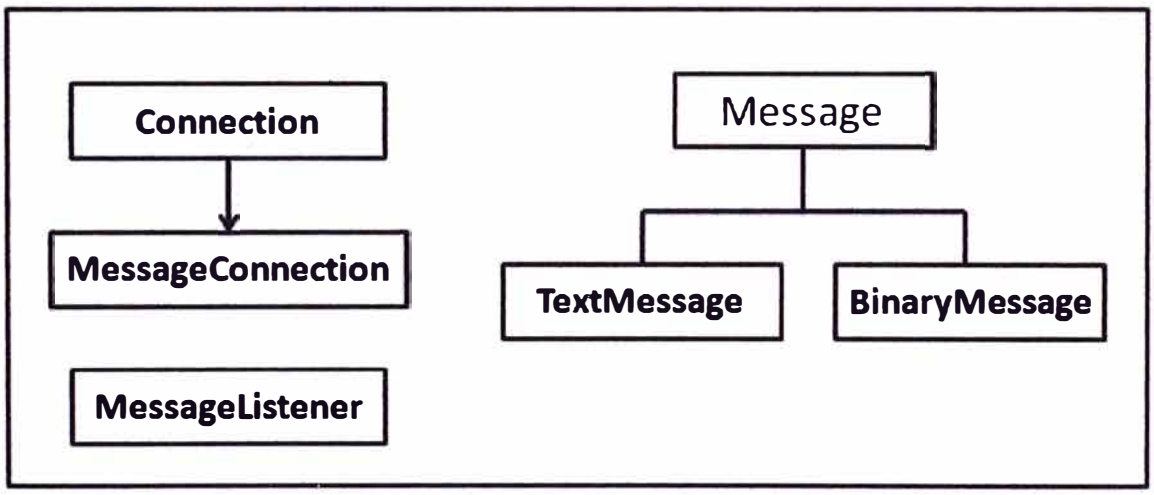

Figura 3.3 Componentes del API WMA. [7]

- Las interfaces *TextMessage* y *BinaryMessage* amplían la interfaz *Message* para permitir el acceso a mensajes de texto y binarios respectivamente, proporcionando métodos *set()* y *get()* para volcar y recuperar el contenido de un mensaje.
- La interfaz *MessageConnection* proporciona la capacidad de crear instancias de tipo *Message* y la capacidad de recibirlos y enviarlos. El tipo de mensaje creado se indica mediante las constantes TEXT_MESSAGE y BINARY_MESSAGE.
- La interfaz también permite especificar un receptor de tipo *MessageListener* que será

invocado cuando se reciba un mensaje a través de la llamada a su método *NotifyIncommingMessage().*

3.2.2 Envío de Mensajes

Para proceder a enviar un mensaje SMS se requiere invocar en primer lugar el método *newMessage()* la cual es una instancia de MessageConnection, luego completar el contenido con los datos binarios o de texto que constituyan el mensaje a enviar y finalmente invocar el método *send().*

Observaciones:

- Un mensaje SMS de datos está limitado a 160 o 170 caracteres.
- Mientras que un mensaje SMS binario está limitado a 140 bytes.

3.2.3 Pasos para el envío de un mensaje

```
//Se compone la dirección URL de destino con el número de teléfono y el puerto de comunicación
String destino = "sms://" +tel+":" + PUERTO_SMS;
MessageConnection con = null;
//Se abre la conexión con el destinatario
Con= (MessageConnection)Connector.open(destino);
 //Se Crea una instancia del Mensaje de texto
TextMessagemensaje=(TextMenssage)con.newMessage(MessageConnection.TEXT_ME
SSAGE);
// Se fija el destinatario
Mensaje.setAddress (destino);
  // Se fija el contenido del mensaje
Mensaje.setPayloadText (tbMensaje.getString());
  // Se envía el mensaje
Con.send(mensaje);
// Se indica al usuario que el mensaje fue enviado
Alert aviso = new Alert (" Envio SMS", "Mensaje enviado", null, AlertType. ERROR);
Dysplay.SetCurrent (aviso, formulario);
// Se cierra la conexión abierta con el destinatario
Con.close
```

3.2.4 Recepción de mensajes

La recepción de mensajes es simple, porque una vez abierta conexión con el servidor la aplicación solamente debe invocar al método receive(), que devolverá el siguiente mensaje disponible en el puerto que se indique. En caso de que no haya ningún mensaje en espera, el método bloquea la tarea hasta que llegue uno o hasta que una tarea

diferente cierre la conexión.

Para saber si hay mensajes pendientes o no, la aplicación registra un receptor de eventos con la conexión establecida con el servidor invocando al método *notifyIncomingMessage*. Cada vez que llega un mensaje a la conexión, el sistema invocara este método en el receptor.

En el momento en que se recibe la notificación ya es posible recurrir al método receive() para recuperar el mensaje.

3.2.5 Pasos para la recepción de mensajes

```
// Se comprueba la conexión y se lee el mensaje
Mensaje= con.receive();
If(mensaje != null && mensaje instanceofTextMessage)
//Se verifica el origen del envío
String origen= mensaje.getAddress();
//Se crea un mensaje indicando el origen y el texto del mensaje recibido
Texto = new StringItem ("", origen+ "\n ->" +((TextMessage)mensaje).getPayloadText());
//y se lo presenta en pantalla
Formulario.append(texto);
```

3.2.6 Segmentación de mensajes

La especificación WMA requiere soporte para la concatenación de mensajes, indicando que se deben soportar al menos tres segmentos SMS para un único mensaje

En ciertas ocasiones el envío de mensajes más largos puede hacer que se lance una excepción IOException por que la implementación no soporta la segmentación en más de los tres segmentos exigidos en la especificación.

El API proporciona el método *numberOfSegments()* para determinar si un mensaje puede ser enviado y cuantos segmentos deberán ser usados para hacerlo, antes de ser enviado, incluso aunque no esté abierta la conexión. El método devuelve 0 si el mensaje no se puede enviar.

3.2.7 Seguridad

La especificación WMA no define un mecanismo de seguridad propio, sino que utiliza el entorno de seguridad que proporciona el propio perfil. MIDP considera las operaciones de red como operaciones privilegiadas, es decir, es necesario un permiso para realizar operaciones como abrir conexión de red o enviar y recibir mensajes.

Si la aplicación no dispone de privilegios para realizar la operación el sistema lanzará una excepción de tipo SecurityException.

Los permisos se invocan mediante propiedades MIDlet-Permissions en el fichero JAD por ejemplo:

- Javax.microedition.io.Conector // Para abrir conexión para el protocolo de mensajería
- MessageConnection.send// Para el envío de mensajes por un puerto determinado
- MessageConnectio.receive // Para permitir a la aplicación recibir mensajes a través de un puerto especifico

El Código completo para el envío y recepción de Mensajes Multimedia se encuentra en el Anexo A.

3.3 MMAPI - JSR 135 Mobile Media API

Es un paquete opcional, con soporte obligatorio para JWTI (Tecnología Java para la Industria Inalámbrica), que permite el manejo y administración de servicios y objetos multimedios en dispositivos restringidos en recursos. Las principales características se muestran en la Tabla 3.3.

Tabla 3.3 Características de JSR 13A5 PI de Medios Móviles MMAPI [8]

Característica	Descripción
Soporte para generación de tonos, reproducción y grabación de medios basados en línea de tiempos	Esta API da soporte a cualquier contenido de audio y video basado en el tiempo. La generación de tonos es un tipo especial de medio parametrizado por su frecuencia y duración.
Destinatario principal en CLDC	Esta API está diseñada para un dispositivo basado en CLDC, sin excluir otros entornos como CDC. CLDC es el denominador común más básico.
Proceso de tamaño reducido en memoria	CLDC impone estrictos límites para consumo de memoria. El diseño de esta API hace énfasis en esto tanto como sea posible.
Independencia de protocolos y contenidos	El diseño de la API no favorece ningún protocolo o contenido en particular, sino a todos en general.
Modular Sólo audio vs. Multimedios en general	Se puede implementar una porción o subconjunto de esta API para dar soporte a un determinado tipo de contenido (por ejemplo, audio básico). De esta forma se puede aprovechar aquellas partes que tengan soporte en el perfil, ya que algunos pueden no tener 100% del soporte necesario.
Extensible	Se puede agregar funcionalidad sin impactar negativamente en la compatibilidad hacia atrás.
Opcionalidad para implementación	Esta API ofrece el rango más amplio de características. Sin embargo, existe la posibilidad de implementaciones que no soporten la totalidad de las mismas, dejando algunas de ellas "sin implementar".

El proceso multimedios se puede dividir en dos partes:

- **Manejo del protocolo de provisión de datos.** El manejo del protocolo se refiere a la lectura de datos desde una fuente (por ejemplo, un archivo, dispositivo de captura, o servidor de entrega-por-demanda) hacia un sistema de procesamiento de multimedios.
- **Manejo del contenido de los datos.** Constituye el procesamiento en sí de los datos

(decodificación, interpretación) y "armado" del medio hacia los dispositivos

En la Figura 3.4 se aprecia la separación entre la fuente de datos y el procesamiento:

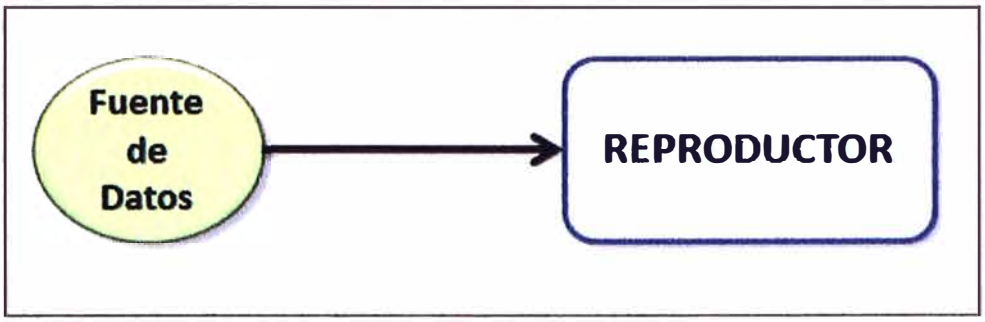

Figura 3.4 Esquema de manejo multimedios básico [8]

El objeto fuente de datos es una clase DataSource (manejo del protocolo), el objeto *reproductor* es una clase Player (manejo del contenido).

La fuente de datos encapsula los detalles de la lectura de los datos, abstrayendo su origen verdadero (archivo, servidor sobre demanda, mecanismo propietario de entrega de contenido). Expone métodos que permite al reproductor leer los datos para su procesamiento.

El reproductor lee datos de la fuente de datos y los procesa, entregando la salida de dicho procesamiento a un dispositivo de salida. Contiene métodos para controlar la reproducción de los medios, y de sincronización básica. También brinda controles específicos para determinados tipos de contenido.

Un mecanismo de fábrica (factory) o de construcción, la clase Manager implementa las clases Players desde los DataSources. Para mayor comodidad, Manager también permite crear los reproductores desde localizadores de contenido (por ejemplo, el localizador para la cámara es "capture://video") y de secuencia de datos de entrada.

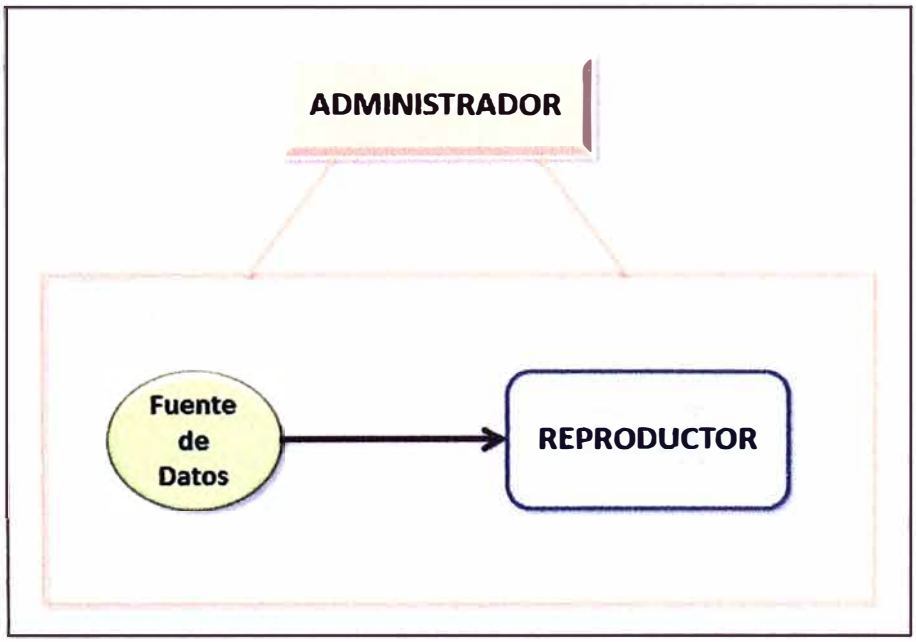

Figura 3.5 JSR 135 - Mecanismo de factory en multimedios [8]

Tabla 3.4 Secuencia de administración de medios. [8]

Acción	Comentarios
Se ejecuta el método createPlayer()	Método punto de entrada superior para esta API Player Manager.createPlayer(String urlString) ▪ urlString Especifica el protocolo y la ubicación de los medios con el formato <protocolo>:<ubicación de contenido>
Manager crea un Datasource	El administrador crea un objeto DataSource para manejar el protocolo específico de entrega de datos, se determina por el tipo de datos.
Manager crea un Player	En función del tipo de contenido, el Administrador crea un objeto reproductor de contenido Player para manejar la presentación de los datos. Se devuelve el objeto Player a la aplicación para su utilización

El objeto reproductor Player contiene métodos generales para control de flujo de información y su presentación, algunos de ellos se exponen en la Tabla 3.5.

Tabla 3.5 Algunos métodos de la clase Placer. [8]

Metodo	Comentarios
Player.realize()	Construye porciones del reproductor Player sin adquirir recursos exclusivos.
Player.prefetch()	Adquiere el resto de los recursos y procesa la mayor cantidad posible de datos para reducir la latencia de inicio, a través de un mecanismo de captación previa o lectura en adelanto.
Player.start()	Inicia el reproductor tan pronto como sea posible.
Player.setMediaTime (long time)	Establece el "tiempo del medio" en el reproductor, recibiendo dicho parámetro como argumento en microsegundos.

Una de las características más importantes de esta API es el control finamente granular que expone. Cada objeto reproductor Player proveerá controles específicos del tipo de contenido, a través de los métodos getControls y getControl métodos:

- *Control[ ] Player.getControls()*
- *Control Player.getControl(String controlType)*

Con ello se exponen características que son únicas para determinado tipo particular de medios, por ejemplo, para un reproductor de secuencias MIDI (Interfaz Digital para Instrumento Digital), el método getControl devuelve un puntero a MIDIControl.

3.4 Análisis del código del MIDlet CamaraMIDlet

Este MIDlet ayudará con la captura de imágenes usando la cámara integrada del celular. A continuación se procede a analizar cada una de las partes del código:

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
```

Con la primera línea se importa la clase MIDlet, que es obligatoria para que el programa se ejecute en el dispositivo móvil, y con la segunda línea se importan los

elementos que se van a utilizar en la interfaz de usuario.

A continuación se importan los paquetes que no permitirán utilizar los tipos de variables Player y VideoControl respectivamente.

```
import javax.microedition.media.*;
import javax.microedition.media.control.*;
```

Luego de importar los paquetes necesarios para la construcción del MIDlet, se procede a crear la class CamaraMidlet.

```
public class CamaraMidlet extends MIDlet implements CommandListener,Runnable
{
   private Display pantalla;
   private Form p_principal;
   private Command cmdPlay;
   private Item miVideoItem;
  // variables para video
  private Player miPlayer;
  private VideoControl miVidcontrol;
```

En la primera línea se declara la clase principal del programa, que es pública. Hereda de la clase MIDLet e implementa las interfaces CommandListener y Runnable. También se declaran una serie de variables que se utilizará en el programa.

Se procede a crear el constructor de la clase. Observe que tiene el mismo nombre que la clase (mayúsculas y minúsculas incluidas) y además no tiene tipo de retorno, ni siquiera void. Aquí se debe inicializar las variables.

```
public CamaraMidlet(){
     // Se obtiene el objeto display del Midlet.
pantalla=Display.getDisplay(this);
```

Al ejecutarse un MIDlet, éste crea un objeto display, que es el encargado de mostrar información en la pantalla. Para poder utilizarlo, se tiene que obtener una referencia a este objeto. Esto es lo que hace precisamente la siguiente línea mediante el método getDisplay() del objeto estático Display.

Se crea la pantalla (formulario) y se le añade los comandos Salir y Play

```
p_principal=new Form("Camara Midlet");
p_principal.addCommand(new Command("Salir",Command.EXIT,1));
cmdPlay=new Command("Play",Command.SCREEN,1);
p_principal.addCommand(cmdPlay);
```

Se añade los comandos (mediante el método addCommand() del objeto screen) a la lista de comandos y se establece que clase permanece a la "escucha" de esos comandos

utilizando la clase setCommandListener().

p_principal.setCommandListener(this); *//Modo escucha a la espera de comandos*

En este caso, el método encargado de procesar los comandos está dentro de la propia clase CamaraMidlet, por lo que se utiliza el operador this. Si se quisiera tener una clase separada encargada de procesar los comandos, se la indicaría aquí. En concreto, el método que se encarga de procesar los comandos es commandAction(). Para eliminar un comando se puede utilizar removeCommand().

```
/* Aquí se incluye el código que ejecutará el MIDlet ejecute cuándo se active. */
public void startApp() {
    // Se selecciona la pantalla a mostrar
    pantalla.setCurrent(p_principal); //Muestra el formulario principal
```

Mediante el método setCurrent() del objeto display (aquel del que se obtuvo la referencia al principio del constructor) se selecciona la pantalla actual para ser mostrada. El método startApp() es el que se ejecuta en primer lugar.

```
  public void pauseApp() {
/* Aquí se incluye el código que se quiere que el MIDlet ejecute cuándo entre en el estado
de pausa (Opcional). */
  }
```

Esta línea puede parecer extraña, ya que es un método vacío (sin código). Como ya se vio, hay que implementar todas las clases heredadas de MIDlet (pauseApp, destroyApp y startApp), por lo que, aunque no contenga código, hay que declararla.

```
/* Aquí se incluye el código que se quiere que el MIDlet ejecute cuándo sea destruido.
Normalmente aquí se liberaran los recursos ocupados por el MIDlet como memoria, etc.
(Opcional) */
  public void destroyApp(boolean unconditional) {
    if (miPlayer!=null){
      miPlayer.close();
    }
  }
  public void commandAction(Command c,Displayable d){
    //Salir
    if (c.getCommandType()==Command.EXIT){
      destroyApp(false);
      notifyDestroyed();
```

Cuando el usuario genera un comando, se llama al método commandAction(). Este método recibirá dos parámetros. El comando que se generó, y un objeto de la clase

Displayable, que contiene la pantalla del comando.

Se cierra la aplicación con los métodos destroyApp() y notifyDestroyed().

```
    }
    else{
      Form waitForm=new Form("Cargando ...");
      //Se selecciona la pantalla a mostrar
      pantalla.setCurrent(waitForm);
      Thread t=new Thread(this);
      t.start();
    }
  }
```

En el método run() se pone el código que se quiere que se ejecute en un hilo separado.

```
  public void run(){
    playArchivo();
  }
```

Se implementa el método playArchivo() el cual ayudará reproducir la imagen deseada

```
  public void playArchivo(){
```

Para tratar excepciones se debe usar la instrucción try (que significa intentar en español).

```
    try{
      miPlayer=Manager.createPlayer("capture://video");
      miPlayer.realize();
      if ((miVidcontrol=(VideoControl)miPlayer.getControl("VideoControl"))!=null){
        miVideoItem=(Item)
        miVidcontrol.initDisplayMode(VideoControl.USE_GUI_PRIMITIVE, null);
        p_principal.append(miVideoItem);
      }
      miPlayer.start();
//se retira el comando play de la pantalla
      p_principal.removeCommand(cmdPlay);
      //Se selecciona la pantalla a mostrar
      pantalla.setCurrent(p_principal);
    }
```

Entre las llaves de try se escribe el código que hará funcional el programa.

Para capturar la excepción que puede generar este código se necesita otra

instrucción llamada catch (capturar).

```
    // Entre las llaves de catch se escribe el código que se quiera para tratar el error.
     catch(Exception e){
        showException(e);
        return;
     }
     pantalla.setCurrent(p_principal);
  }
  public void showException(Exception e){
     Alert a=new Alert("Error",e.toString(),null,null);
      a.setTimeout(Alert.FOREVER);
```

Para crear una alerta se utiliza su constructor que tiene la siguiente forma:

Alert (String título, String texto_alerta, Image imagen_alerta, AlertType tipo_alerta)

El título aparecerá en la parte superior de la pantalla. El texto de alerta contiene el cuerpo del mensaje que se quiere mostrar. El siguiente parámetro es una imagen que se mostrará junto al mensaje. Si no se quiere imagen se le pasa null como parámetro. El tipo de alerta puede ser ALARM, CONFIRMATION, ERROR, INFO, WARNING.

La diferencia entre uno y otro tipo de alerta es básicamente el tipo de sonido o efecto que produce el dispositivo. Se puede establecer el tiempo del mensaje con el método setTimeout(int tiempo) donde se puede especificar el tiempo en milisegundos. También se puede hacer que el mensaje se mantenga hasta que se pulse un botón del dispositivo indicando setTimeout(Alert.FOREVER)

Ahora solo queda mostrar la alarma en la pantalla.

```
     pantalla.setCurrent(a, p_principal);
  }
// llave final
}
```

El código completo para la captura de foto con la cámara de un celular se colocará en el Anexo B.

CAPÍTULO IV
SIMULACION Y PRUEBAS DEL SOFTWARE DE ENVÌO DE MENSAJE

En este capítulo se muestran las pruebas de los programas realizados en el lenguaje de programación J2ME. Para las simulaciones se utilizará el entorno integrado de programación (IDE) NetBeans, el cual puede ser descargado de: http://netbeans.org.

Pero antes de empezar a usar este IDE o cualquier otro es necesario tener en cuenta que se requiere un JDK para correr e interpretar el código Java; el cual se puede descargar de: http://www.oracle.com/technetwork/java/javase/downloads/index.html.

4.1 Simulación del MIDlet para el envío de MMS

La pantalla principal del MIDlet permite utilizar los campos de Asunto, Destinatario, la opción de salida y una pestaña de Menú (Figura 4.1).

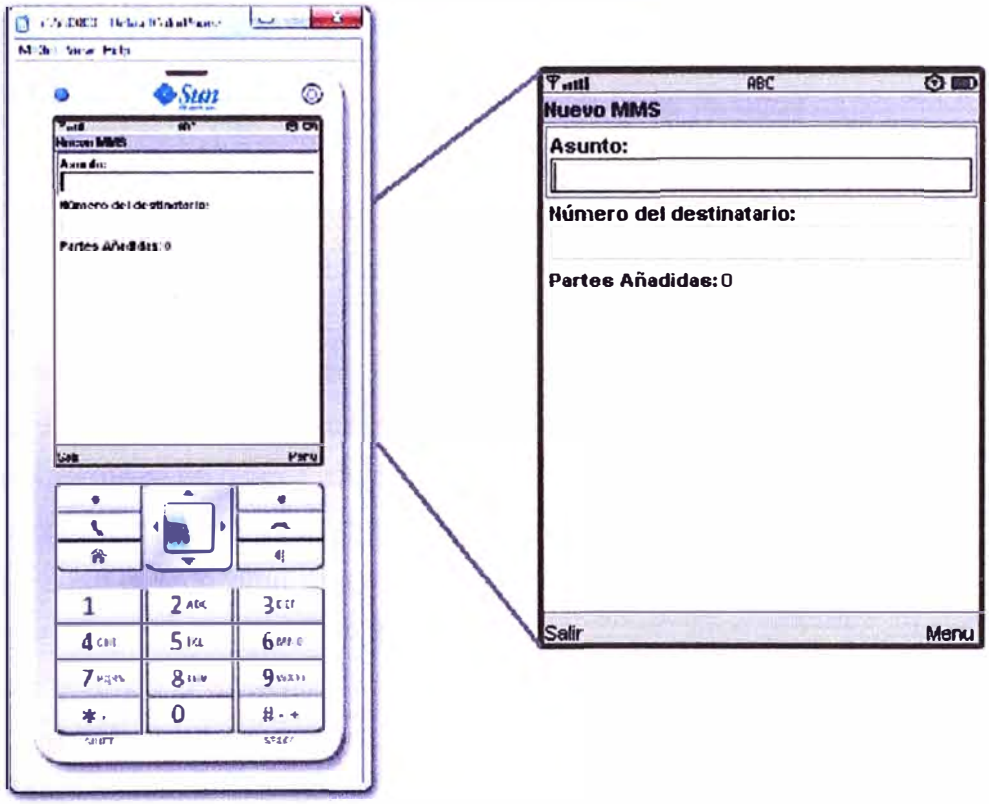

Figura 4.1 Pantalla inicial del MIDlet CapturarFoto

Se coloca como Asunto: Probando envío de MMS y el Número del destinatario: 5550001, este es el número del Celular B, cabe señalar que el número del celular A es el 5550000, luego al pulsar la opción de menú se permite elegir entre enviar el MMS ó la Añadir (Figura 4.2).

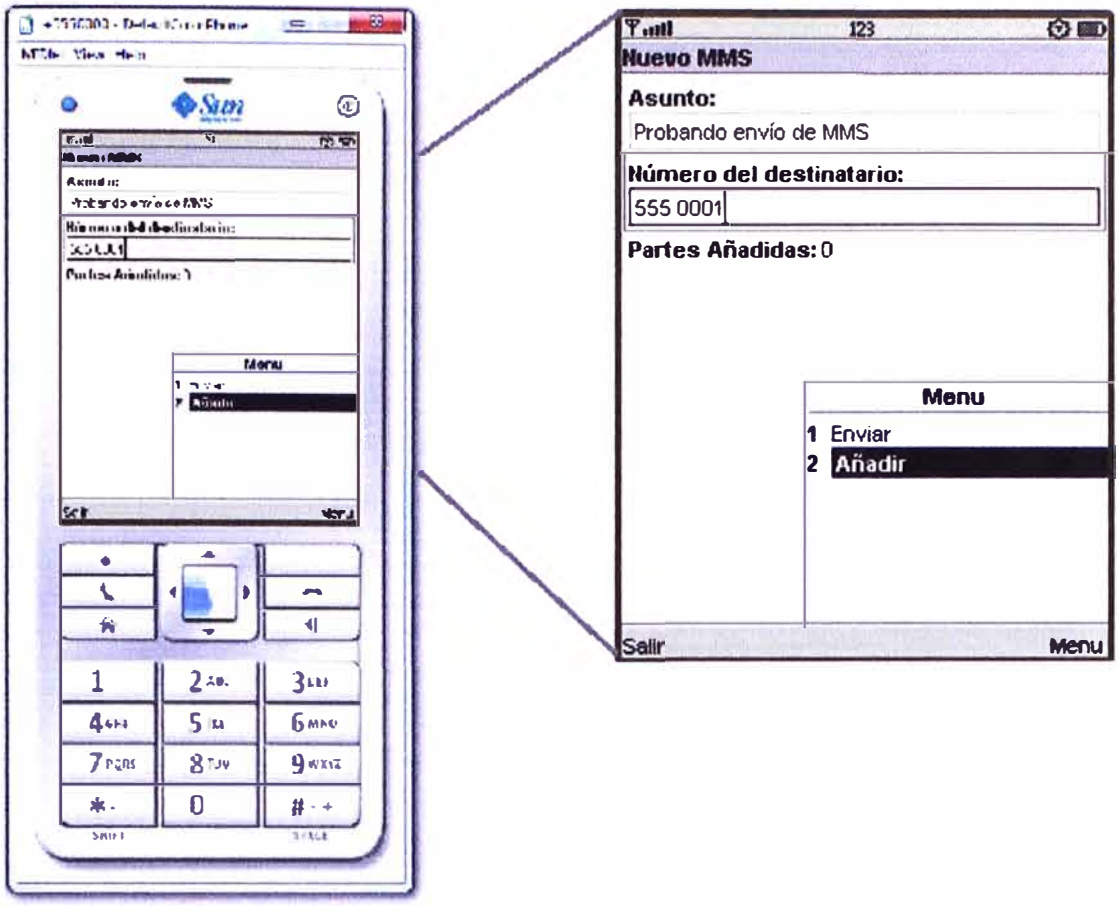

Figura 4.2 Pantalla con los campos llenados

El programa consulta que es lo que se desea añadir al MMS, existen las opciones de añadir Texto, lo que vendría a ser el contenido del mensaje o una imagen. Se elige la opción de añadir Texto (Figura 4.3).

Al elegir la opción de añadir Texto, aparece un editor de texto, en esta ocasión se añade el Texto: Hola a todos, este es un MIDlet para enviar MMS, posteriormente se presiona la opción de OK (Figura 4.4).

De igual forma si se quiere añadir una imagen al MMS se elige la opción respectiva (Figura 4.5).

En este caso el programa proporciona la opción de enviar 5 imágenes prediseñadas. Se elige la imagen Diablo (Figura 4.6).

Una vez añadido el contenido y la imagen se procede a enviar el MMS, nótese que el número de partes añadidas es igual a 2 (Figura 4.7).

Finalmente se recibe el MMS en el celular B (Figura 4.8), en dicho mensaje se puede apreciar:

- El número que envía el MMS: 5550000 (Celular A).
- La Fecha y Hora en que fue enviado el MMS.
- El Contenido: Hola a todos, este es un MIDlet para enviar MMS.
- La Imagen añadida al MMS.

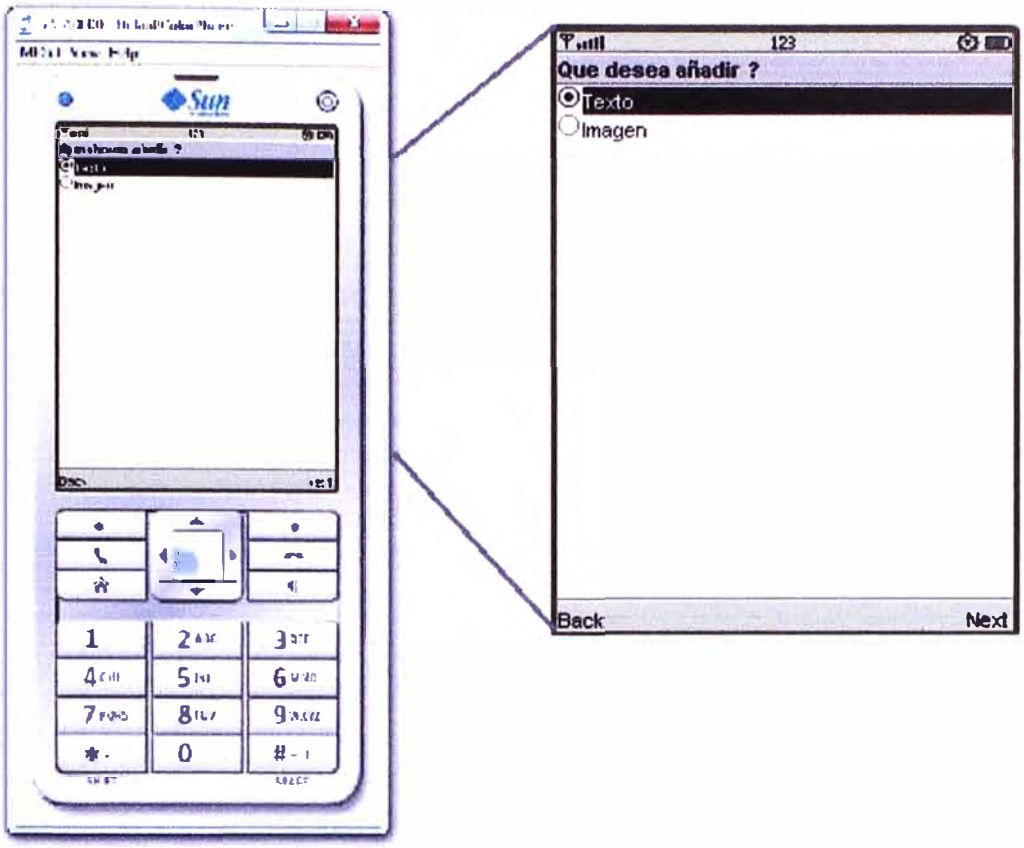

Figura 4.3 Consulta de lo que se desea añadir al MMS

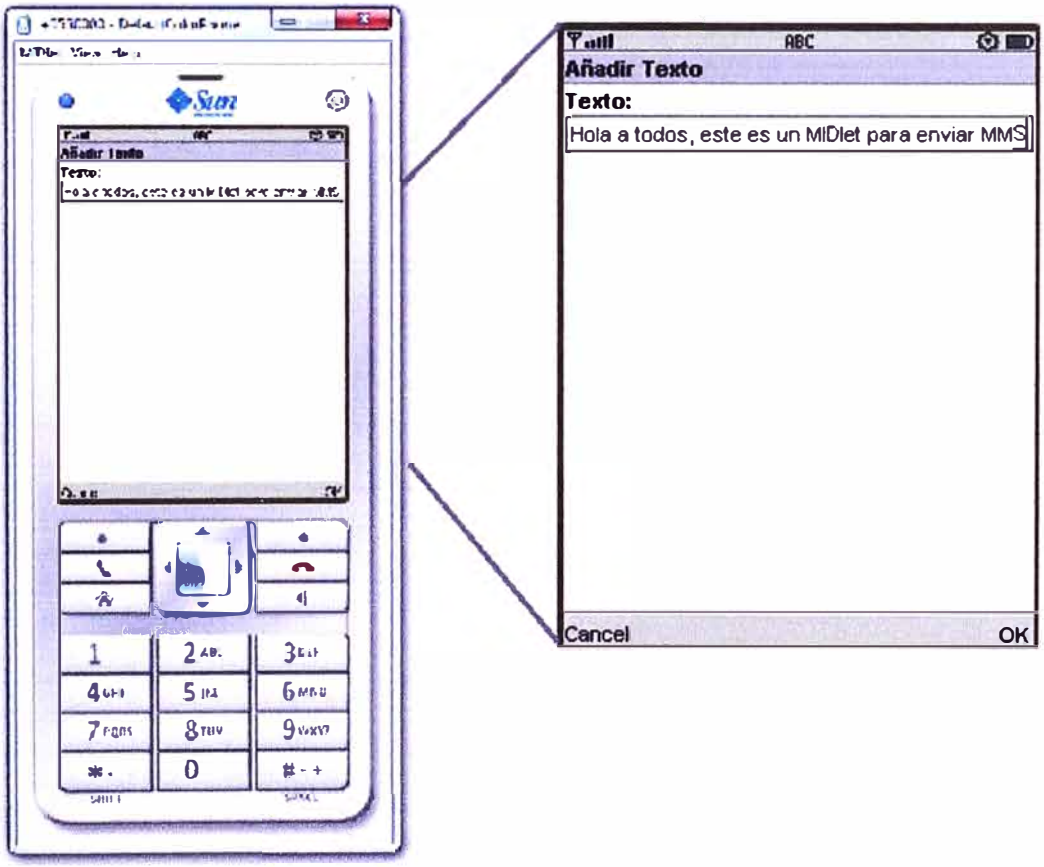

Figura 4. 4 Añadiendo texto

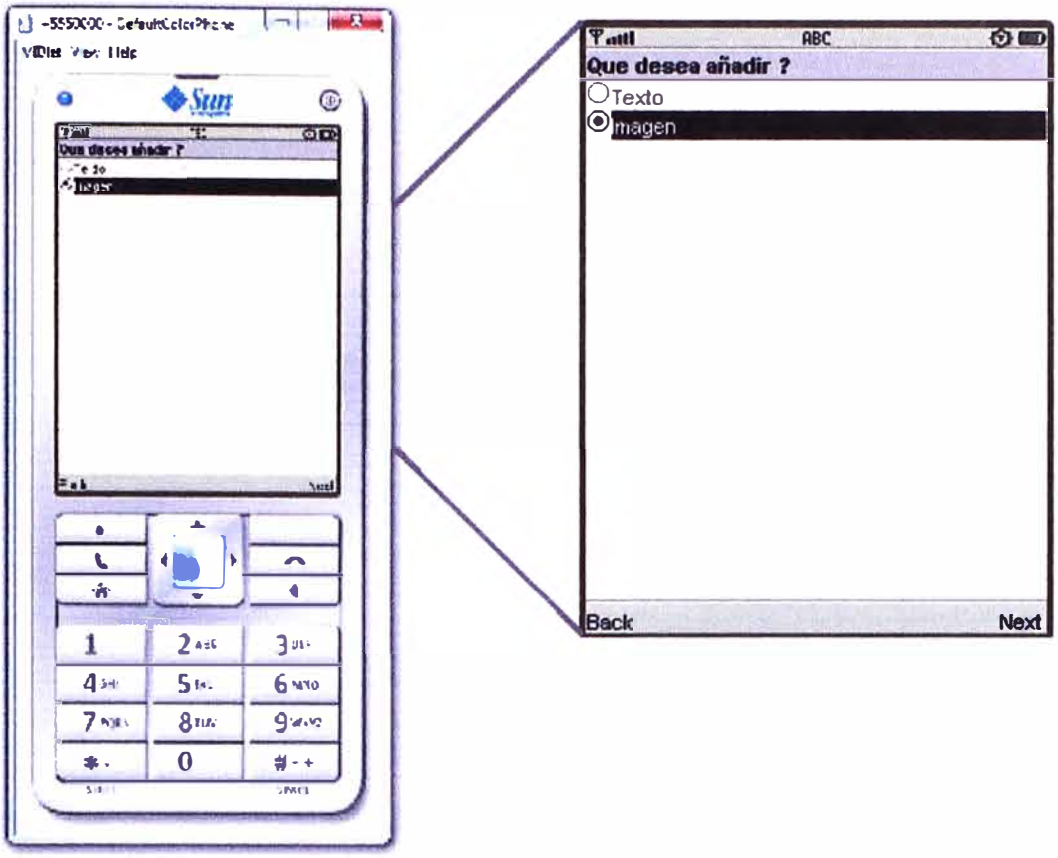

Figura 4.5 Añadiendo imagen

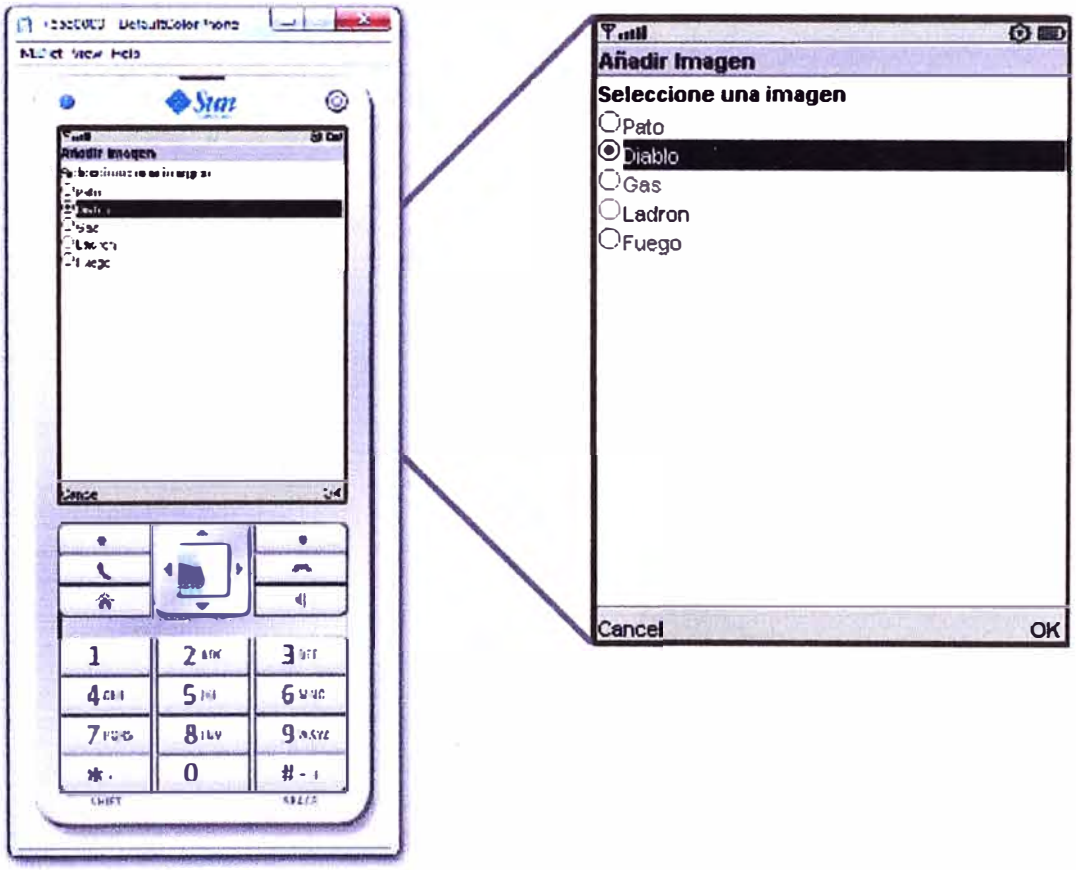

Figura 4.6 Selección de imagen

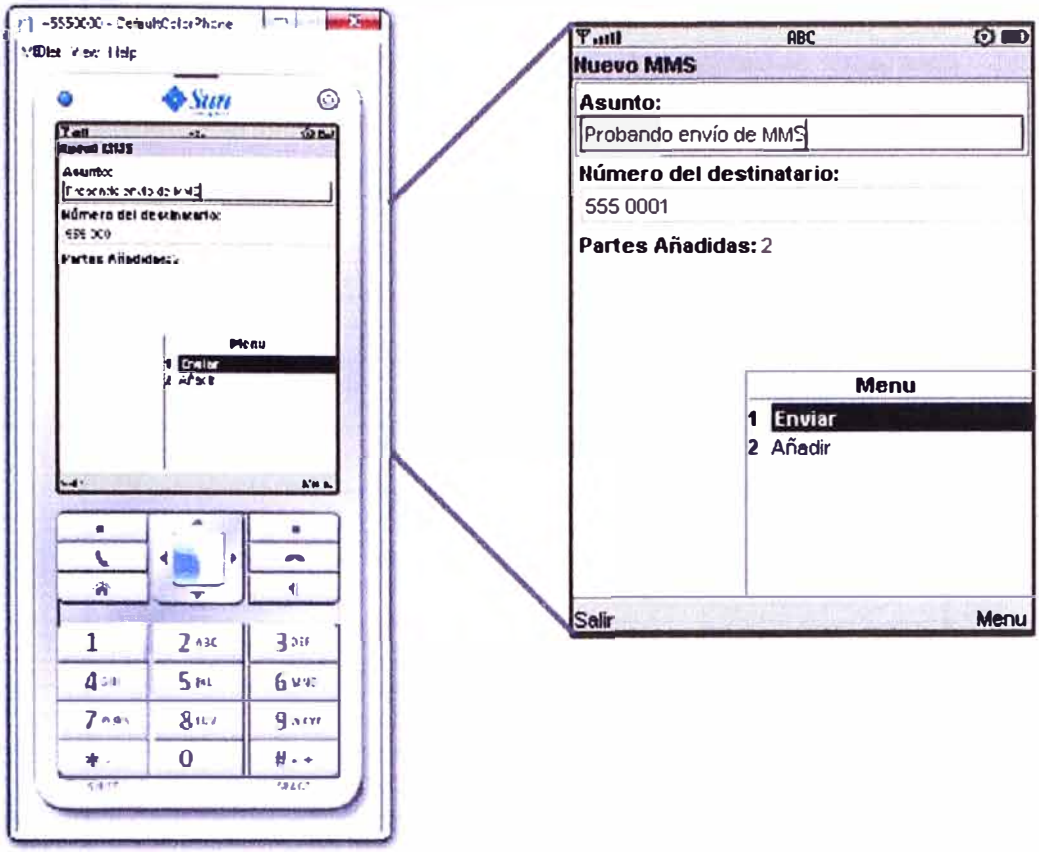

Figura 4.7 Envío de MMS

Figura 4.8 Recepción de MMS

4.2 Simulación del MIDlet FotoControl

En esta ocasión para realizar la simulación del MIDlet el programa cargará un pequeño video y lo reproducirá, simulando la cámara del dispositivo móvil (Figura 4.9).

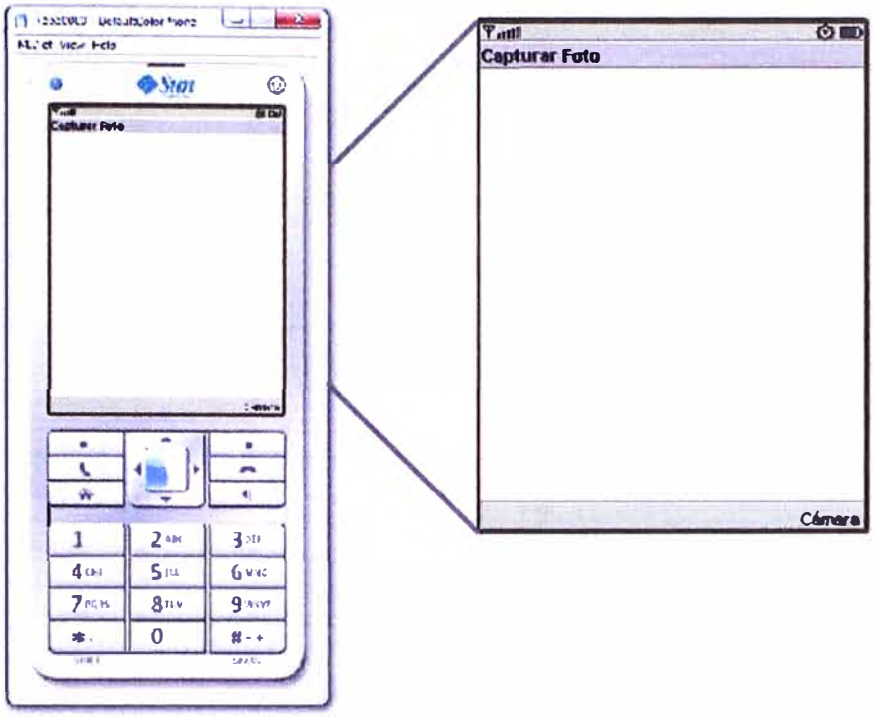

Figura 4.9 Captura de foto

Para activar la cámara se presiona la opción Cámara, en este caso el MIDlet reproducirá un pequeño video que contiene varias escenas (Figura 4.10).

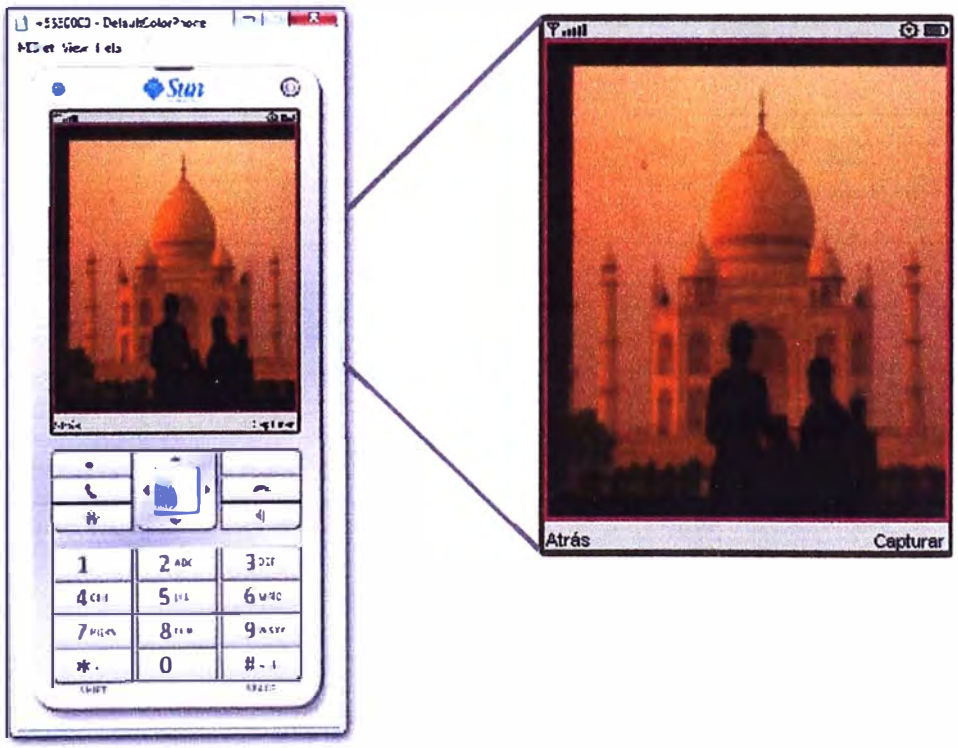

Figura 4.10 Reproducción de video

Para realizar la captura de la pantalla se presiona la opción Capturar, en este caso se realizó justo cuando se mostraba en pantalla a una señorita hablando por celular (Figura 4.11).

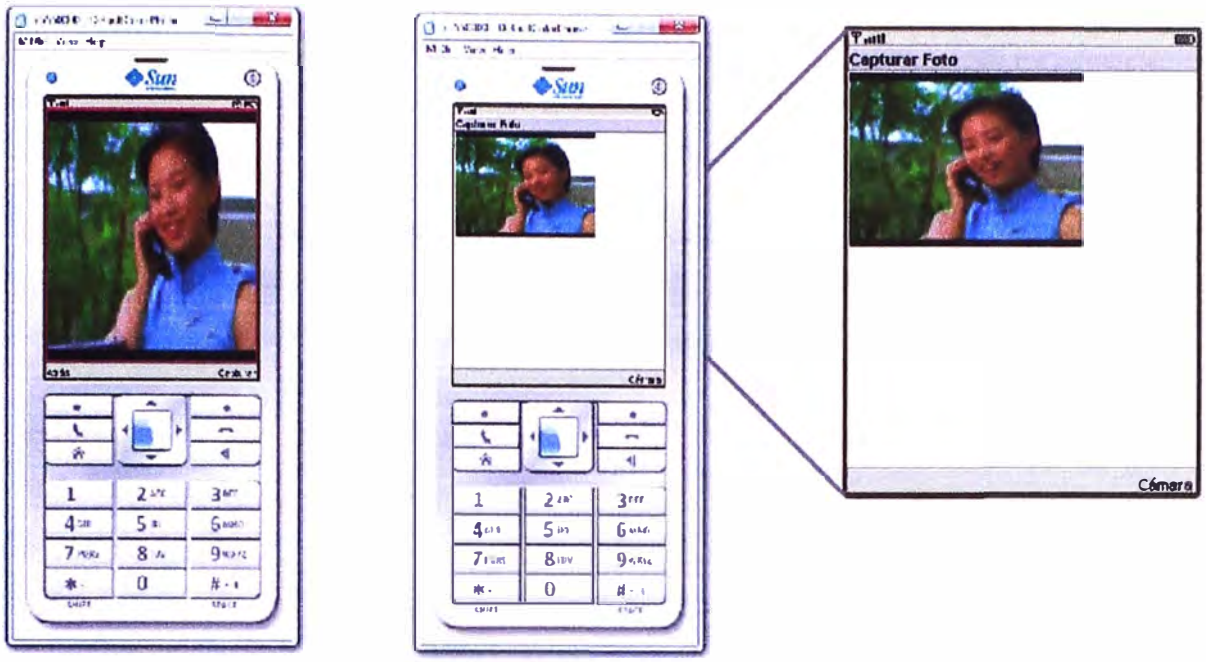

Figura 4.11 Captura de pantalla

Posteriormente el programa permite la opción de volver a visualizar la cámara y realizar nuevas capturas (Figura 4.12).

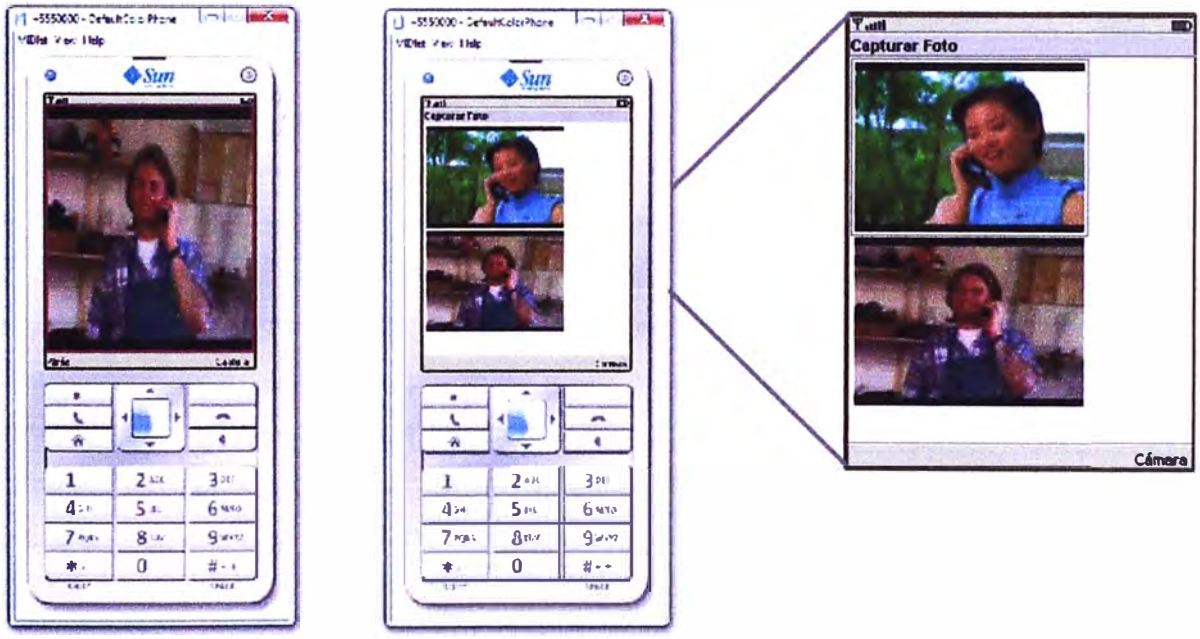

Figura 4.12 Visualización y captura

En el modo de visualización de la cámara existe la opción de Atrás, esto permitirá visualizar todas las fotos capturadas por el programa, tal como se puede apreciar en la Figura 4.12.

CAPÍTULO V
CONSIDERACIONES PARA EL DESARROLLO DEL SISTEMA DE CONTROL

En este capítulo se indica los aspectos importantes que tienen que ser considerados al momento de implementar el sistema de control, se explica cada una de las partes que componen este subsistema del sistema de vigilancia, las cuales se dividen en:

- Selección del dispositivo móvil
- Selección del microcontrolador PIC
- Selección del sensor de movimiento
- Puerto Usart
- Comandos AT
- Diagrama de flujo del programa
- Funcionamiento del sistema

5.1 Selección del dispositivo de comunicación GSM

Las principales características que se deben considerar para la selección de los terminales son: puerto serial de comunicaciones, capacidad de envío y recepción de mensajes, y protocolo de comunicaciones.

- **Puerto Serial de Comunicaciones:** Es necesario que el dispositivo celular posea un puerto de comunicaciones accesible para la aplicación, mediante algún tipo de conector. De igual forma debe existir la suficiente información técnica del dispositivo de manera que no cause ningún tipo de daño en su manipulación. El puerto del dispositivo debe permitir la comunicación bidireccional con el microprocesador.
- **Capacidad de envío y recepción de mensajes (Short Message Service):** Uno de los servicios que ofrece GSM es la posibilidad de envío de mensajes SMS. Al elegir un terminal GSM se asegura que tanto el terminal de envío como el de recepción posean la capacidad de enviar y recibir mensajes SMS.
- **Protocolo de comunicaciones:** El protocolo de comunicación del terminal debe permitir la comunicación entre el microcontrolador-teléfono y teléfono-computador.

De acuerdo a los diferentes tipos de marcas de teléfono se ha desarrollado diversos protocolos de comunicaciones siendo los más utilizados los comandos AT. Una forma de saber si el teléfono soporta comandos AT es que el móvil cuente con la tecnología GPRS, hoy en día casi todos los teléfonos soportas este tipo de comunicación.

Dentro del mercado existen diferentes tipos de terminales. Entre los más importantes se encuentran: teléfonos Nokia, Samsung, Motorola, Siemens, SonyEricsson y MODEMS para PC y de circuito impreso.

Para elegir un celular se tiene un sin número de opciones, solo hay que tener presente que se debe conocer los pines del puerto de comunicación del celular, esto es saber que pin es de Tx, Rx y el pin de Tierra.

Debido a que existe una mayor cantidad de soporte para aplicaciones con terminales Nokia se eligió a un teléfono de esta marca para el desarrollo del proyecto, el modelo elegido es el nokia 3220b (Figura 5.1), porque es fácil de encontrar las características de su puerto de comunicación.

Figura 5.1 Celular Nokia 3220

El teléfono Nokia 3220 funciona en la red inalámbrica GSM a 850/1800/1900 MHz. Tiene conexión y transmisión de datos PC Suite para el Nokia 3220. La transmisión que soporta es EGPRS (Clase 6) hasta 177.6 Kbps GPRS(Clase 10) hasta 80 Kbps y HSCSD(Clase 6) hasta 43.2 Kbps.

Para la comunicación entre el celular y el PIC se requiere de tres hilos; Rx, Tx, GND.

En el puerto de comunicación del teléfono estos corresponden a los pines 6, 7,8 respectivamente. Para la conexión se hace uso del cable de datos correspondiente al Nokia 3220 que es el DKU-5 (CA-42). El extremo donde está el conector USB se cortó para llegar e identificar los pines 6, 7,8. En la Figura 31 se ilustra el puerto de comunicación del celular con los pines a utilizar.

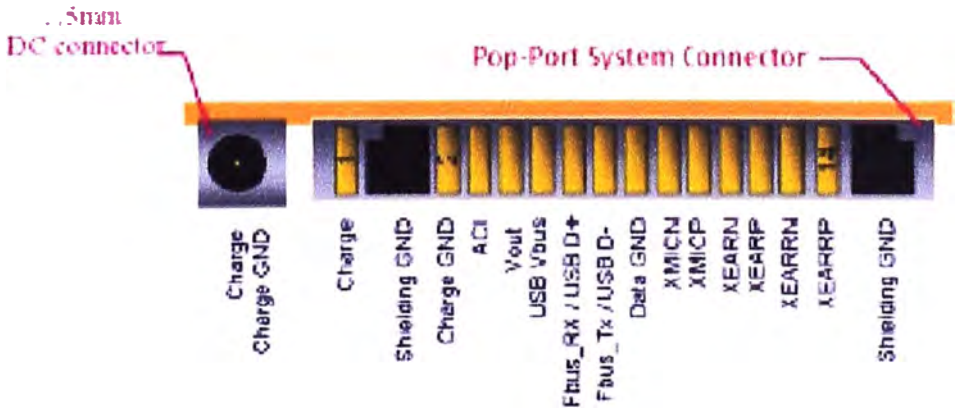

Figura 5.2 Puerto de comunicación del Nokia 3220

El cable de datos del Nokia viene con el celular lo cual fue de gran ayuda pues facilitó la conexión al puerto de datos del celular; caso contrario, se habría optado por soldar cables al referido puerto. En el cable DKU-5 (Figura 5.3) se identificó los conectores de Tx, Rx, GND los cuales corresponden a los siguientes colores.

- Tx Verde
- Rx Blanco
- GND Rojo

Figura 5.3 Cable de datos DKU-5 (CA-42)

A continuación en la Figura 32 se muestra el cable DKU-5 para la conexión del celular receptor con el PIC16F877A.

Vale la pena mencionar que para la conexión de los terminales se debe cruzar la conexión; esto quiere decir que el Tx del Nokia se debe conectar con el Rx del PIC, y viceversa. Un punto importante a considerar es el voltaje de salida del celular el cual es 3.6 V para este modelo y el voltaje de funcionamiento del PIC es de 5V.

Para este caso existen dos maneras de realizar la conexión entre el celular y el microcontrolador, que serán analizados más adelante.

5.2 Selección del microcontrolador

Al momento de elegir el microcontrolador hay aspectos que se deben analizar antes de adquirirlo, a continuación se mencionan algunos de ellos.

a) La complejidad del proyecto electrónico: Si el proyecto va a ser complejo, es decir manejará muchos datos de entrada y salida, es bueno pensar en un microcontrolador PIC que brinde los puertos necesarios para el proyecto; no se desea que el microcontrolador PIC tenga puertos insuficientes ni por demás, lo que se busca es un PIC que se acomode a las necesidades. Una buena técnica de diseño es utilizar los simuladores de circuitos con PIC como el caso de Proteus y otros similares, este tipo de programas brinda la posibilidad de ver funcionando virtualmente al proyecto antes de comprar el microcontrolador PIC.

b) La precisión: Todos los microcontroladores PIC poseen un circuito de reloj u oscilador para sincronizar los ciclos de operación interna, dado que el proyecto no requiere de tiempos muy precisos, se podría utilizar uno con oscilador interno y así ahorrarse el costo del cristal de cuarzo y un par de condensadores cerámicos, además el circuito impreso quedará más sencillo de diseñar y construir.

c) La temperatura de trabajo: Tener en cuenta el ambiente en el que funcionará el microcontrolador PIC, ya que la temperatura de trabajo afectará en su normal operación o incluso puede terminar averiándose el PIC. En su hoja de datos (Datasheet) se especifica el rango de temperaturas en las cuales el microcontrolador puede trabajar eficientemente y de manera segura, la mayoría de ellos opera en rangos de temperatura entre los -40 grados a 150 grados Celsius. Por ejemplo si se requiere construir un circuito que será instalado en un automóvil y cerca al motor, deberá utilizar un microcontrolador que opere a altas temperaturas o en ambiente industrial.

d) El encapsulado: Si el circuito debe ser instalado en un lugar de poco espacio, quizás deba utilizar un microcontrolador de montaje en superficie o (SMD) de varios tipos, comúnmente en encapsulado SOIC y SSOP.

e) La versatilidad: Es una de las característica más importantes que se tiene en cuenta, ya que usualmente cuando se desarrolla un proyecto, un tiempo después se puede pensar en hacerle algunas mejoras al programa del microcontrolador (actualizarlo) y si se ha elegido uno con la memoria de programa muy corta, entonces se tiene que migrar a un microcontrolador PIC mas "grande", lo cual ocasionará mayores gastos.

f) El tipo de memoria: Otra importante característica es el tipo de memoria del microcontrolador, existen versiones CMOS, EPROM, ROM de alta velocidad programables una sola vez (OTP) y también con memoria FLASH regrabables un número importante de veces. Si se elige un PIC con memoria del tipo OTP se debe recordar que una vez grabado ya no se lo puede actualizar o utilizar en otro proyecto, si el diseño no es definitivo es recomendable utilizar un microcontrolador PIC del tipo FLASH.

g) El precio del microcontrolador: En ocasiones se piensa que mientras más avanzado es un microcontrolador este será más costoso, pero esto no siempre es verdad, por ejemplo el PIC 16F84A es más costoso que el PIC 16F628 pero este último posee el doble de memoria FLASH, tiene dos temporizadores, comparadores, oscilador interno etc. Si se sigue comparando se encuentra el PIC 16F648, el cual posee el doble de memoria que el PIC16F628 y tienen un costo similar, otro caso es el del PIC 16f88 el cual posee conversores análogo digital e iguales o mejores características que los anteriores y con un precio prácticamente igual. Desde luego que se encontrarán microcontroladores más costosos debido a sus características técnicas como la velocidad de operación,

numero de puertos, capacidad de memoria y uso de puertos como el USB y hasta el manejo de radiofrecuencia como los RFPIC.

h) Consumo de Energía: Es otro punto importante que tiene que ver con la potencia que utilizará el PIC. Se debe minimizar el consumo de energía, no solo por los gastos económicos que conlleva sino que también es importante el cuidado del medio ambiente (Tecnología Verde).

Viendo todo lo mencionado y teniendo en claro hasta donde se desea abarcar en el presente informe, se decidió utilizar el PIC 16F877A el cual se cuenta con mucha información, no solo del datasheet, sino que es unos de PIC más usados y el más recomendado para iniciarse en el mundo de los Microcontroladores PIC.

A continuación se menciona cada una de sus características. La disposición de sus pines se muestra en la Figura 5.4.

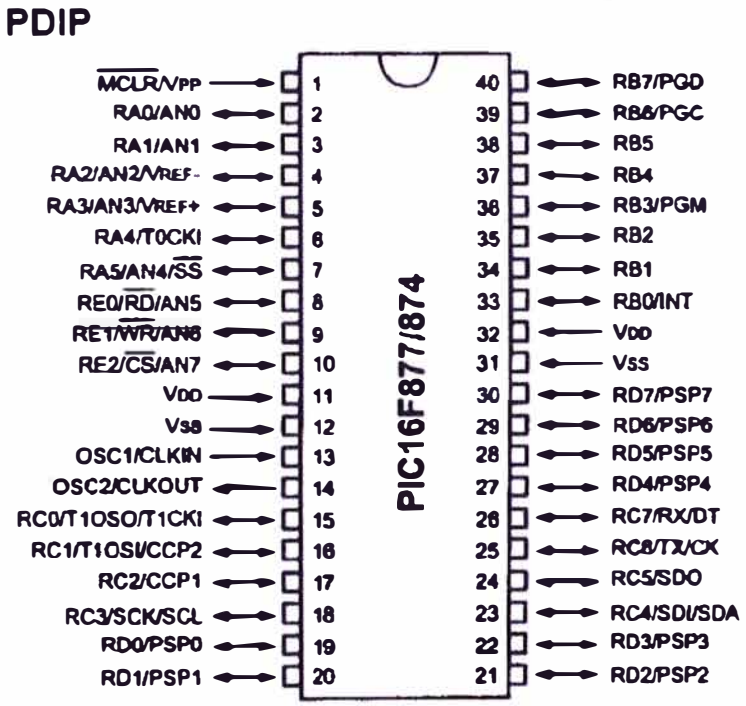

Figura 5.4 Microcontrolador PIC16F877A

CPU:

- Tecnología RISC
- Sólo 35 instrucciones que aprender
- Todas las instrucciones se ejecutan en un ciclo de reloj, excepto los saltos que requieren dos ciclos de reloj.
- Frecuencia de operación de 0 a 20 MHz (200 nseg de ciclo de instrucción)
- Opciones de selección del oscilador

Memoria:

- Hasta 8k x 14 bits de memoria Flash de programa

- Hasta 368 bytes de memoria de datos (RAM)
- Hasta 256 bytes de memoria de datos EEPROM
- Lectura/escritura de la CPU a la memoria flash de programa
- Protección programable de código
- Stack de hardware de 8 niveles

Reset e interrupciones:

- Hasta 14 fuentes de interrupción
- Reset de encendido (POR)
- Timer de encendido (PWRT)
- Timer de arranque del oscilador (OST)
- Sistema de vigilancia Watchdog timer.

Otros:

- 33 pines de Entrada/Salida
- Modo SLEEP de bajo consumo de energía
- Programación y depuración serie "In-Circuit" (ICSP) a través de dos patitas
- Rango de voltaje de operación de 2.0 a 5.5 volts
- Alta disipación de corriente de la fuente: 25mA
- Rangos de temperatura: Comercial, Industrial y Extendido
- Bajo consumo de potencia: Menos de 0.6mA a 3V, 4 Mhz, 20 μA a 3V, 32 Khz, menos de 1μA corriente de standby (modo SLEEP).

Tabla 5.1 Periféricos del PIC16F87X

Periférico	**PIC16F874 PIC16F877**	**Características**
5 Puertos paralelos	PortA,B, C,D,E	Con líneas digitales programables individualmente
3 Timers	Timer0	Contador/Temporizador de 8 bits con pre-escalador de 8 bits
	Timer1	Contador/3 Timers Temporizador de 16 bits con pre-escalador
	Timer2	Contador/Temporizador de 8 bits con pre-escalador y post-escalador de 8 bits y registro de periodo
2 módulos CCP	Captura	16 bits, 1.5 nseg de resolución máxima
	Comparación	16 bits, 200 nseg de resolución máxima
	PWM	10 bits
1 Convertidor A/D	AN0,...,AN7	de 10 bits, hasta 8 canales
Puertos Serie	SSP	Puerto Serie Síncrono
	USART/SCI	Puerto Serie Universal
	ICSP	Puerto serie para programación y depuración "in circuit"
Puerto Paralelo Esclavo	PSP	Puerto de 8 bits con líneas de protocolo

5.3 Selección del sensor de movimiento

Para el presente informe se elegirá un sensor de bajo costo y pequeño tamaño. Entre los diferentes tipos de sensores de movimientos que existen en el mercado existe uno cuya conexión con el PIC es relativamente fácil y es el sensor de movimiento PIR.

El sensor PIR (Pasive Infra Red) es un dispositivo piroeléctrico (detector de calor). Lo que mide es el cambio de calor, no la intensidad de calor. El calor medido es el calor irradiante cercano al infrarrojo que no es visible. Este sensor detecta movimiento mediante un promedio del calor irradiado en el tiempo. Como respuesta al cambio el sensor cambia el nivel lógico de su PIN (0-1). Este sensor (Figura 5.5) es de bajo costo y tamaño, por lo que se utiliza en sistemas de alarmas, iluminación y robótica.

- Voltaje de alimentación = 5 VDC
- Rango de medición = hasta 6 m
- Salida = estado de un pin TTL (Transistor-Transistor Logic)
- Polaridad de activación de salida seleccionable.
- Mínimo tiempo de calibración.

El sensor PIR cuanta con 3 terminales, 2 para alimentación y uno de salida (detección de movimiento). La conexión al microcontrolador solo requiere del uso de este último terminal.

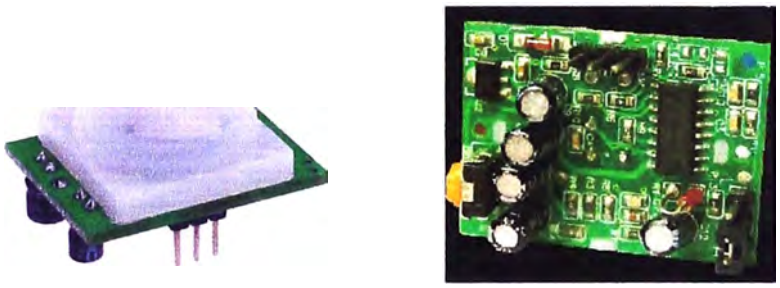

Figura 5.5 Sensor de movimiento PIR

El PIR está fabricado de un material cristalino que genera carga eléctrica cuando se expone a la radiación infrarroja. Los cambios en la cantidad de radiación producen cambios de voltaje que son medidos por un amplificador. Este sensor contiene unos filtros especiales llamados LENTES FRESNEL que enfocan las señales infrarrojas sobre el elemento sensor, cuando las señales infrarrojas del ambiente donde está el sensor cambian, el amplificador activa las salidas para indicar movimiento, esta salida permanece activa durante unos segundos lo que permite que el microcontrolador sepa si es que hubo movimiento.

El espectro electromagnético de la radiación infrarroja, tiene una longitud de onda más larga que la luz visible no puede ser vista pero si puede ser detectada y los objetos que generan calor también generan radiación infrarroja (Figura 5.6).

		Longitud de onda (μm)	Longitud de onda (A°)
Luz ultraviotela (UV)		menor a 0.4	menor a 4000
Luz visible	Violeta	0.46	4600
	Azul	0.5	5000
	Verde	0.56	5600
	Amarillo	0.59	5900
	Ambar	0.61	6100
	Rojo	0.66	6600
Luz infrarroja (IR)		mayor a 0.7	mayor a 7000

Figura 5.6 División de la luz

El PIR viene prediseñado para la detección del cuerpo humano (Figura 5.7). Este sensor funciona detectando cambios en el promedio de captura de calor irradiado cerca al infrarrojo (6 metros radio). Pero si uno se queda quieto frente al sensor, este no detecta más. En teoría si un objeto que no emite calor se mueve el sensor no lo detectaría, por ejemplo un vaso rodando.

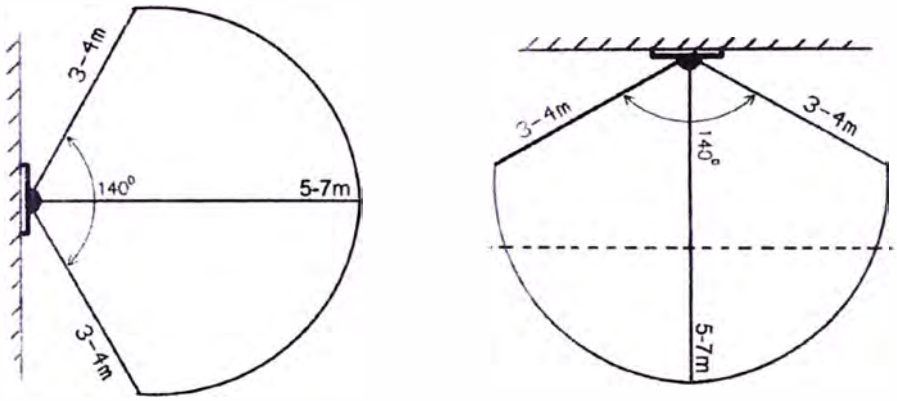

Figura 5.7 Rango de acción de PIR

El sensor PIR posee un cabezal de 3 entradas, esto debe ser conectado al circuito de manera que el pin – (negativo) se conecte a la tierra, el pin + (positivo) se conecte a los 5 volts (power) y el pin out se conecta al pin in/out del microcontrolador PIC (Figura 5.8).

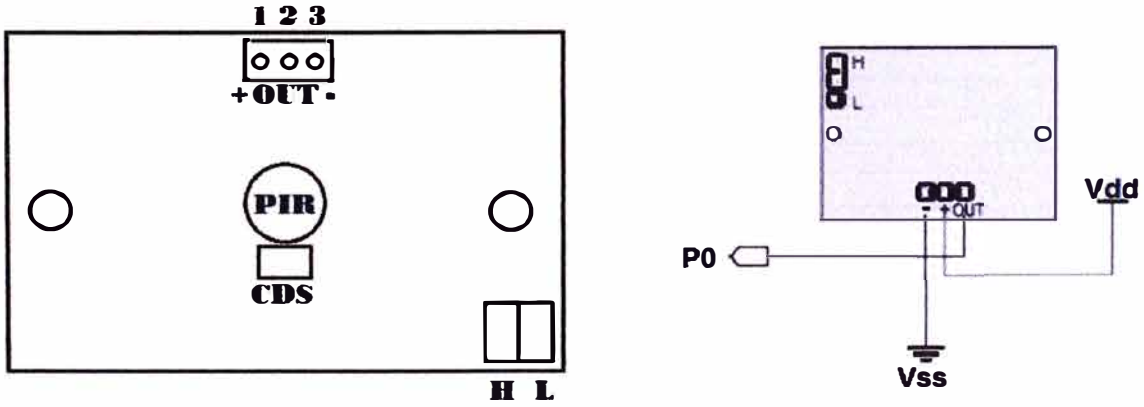

Figura 5.8 Conexiones PIR

También se puede conectar el sensor directamente al protoboard y conectar las señales desde ahí. Se debe recordar las señales de los pines cuando se trabaje de esta forma.

Cuando se prende se debe esperar entre 10 a 60 segundos para que el sensor haga una captura del lugar. Lo ideal sería que no haya personas dentro el rango del sensor.

5.4 Puerto USART del PIC 169F877A

La USART (Universal Synchronous Asynchronous Receiver Transmitter) es uno de los dos periféricos contenidos en el PIC que le permiten realizar comunicación en serie. El otro es el MSSP (Master Synchronous Serial Port), el cual no se va a utilizar en este informe. La USART, también conocida como SCI (Serial Communications Interface) puede configurarse como una unidad de comunicación en serie para la transmisión de datos asíncrona con dispositivos tales como terminales de computadora o computadoras personales, o bien para comunicación síncrona con dispositivos tales como convertidores A/D o D/A, circuitos integrados o memorias EEPROM con comunicación serie, etc.

La gran mayoría de los sistemas de comunicación de datos digitales actuales utilizan la comunicación en serie, debido a las grandes ventajas que representa esta manera de comunicar los datos:

- **Económica**. Utiliza pocas líneas de transmisión inclusive puede usar sólo una línea.
- **Confiable**. Los estándares actuales permiten transmitir datos con bits de paridad y a niveles de voltaje o corriente que los hacen poco sensibles a ruido externo. Además por tratarse de información digital, los cambios en amplitud de las señales (normalmente causadas por ruido) afectan muy poco o nada a la información.
- **Versátil**. No está limitada a usar conductores eléctricos como medio de transmisión, pudiendo usarse también: fibra óptica, aire, vacío, etc. Además el tipo de energía utilizada puede ser diferente: luz visible, infrarroja, ultrasonido, pulsos eléctricos, radio frecuencia, microondas, etc.

Una gran cantidad de periféricos se comunican actualmente en serie con una microcomputadora: líneas telefónicas, terminales remotas, unidades de casete magnético, el ratón, teclados, etc.

Comunicación en paralelo

En este caso se utiliza una línea física por cada bit del dato a comunicar además de posibles líneas para protocolo. Esquemáticamente en la siguiente figura se muestra como se transmitiría el dato de 8 bits 1001 0111= 97h.

Este tipo de comunicación se puede realizar mediante el PIC usando el puerto D como puerto de datos y las líneas del puerto E como líneas de protocolo.

La principal ventaja de la comunicación en paralelo es la alta velocidad de

transmisión, ya que se envían simultáneamente todos los bits de un dato. No obstante, si la distancia entre el transmisor y el receptor es grande, puede ser que el costo de las líneas sea tan alto que se vuelva incosteable este método de comunicación.

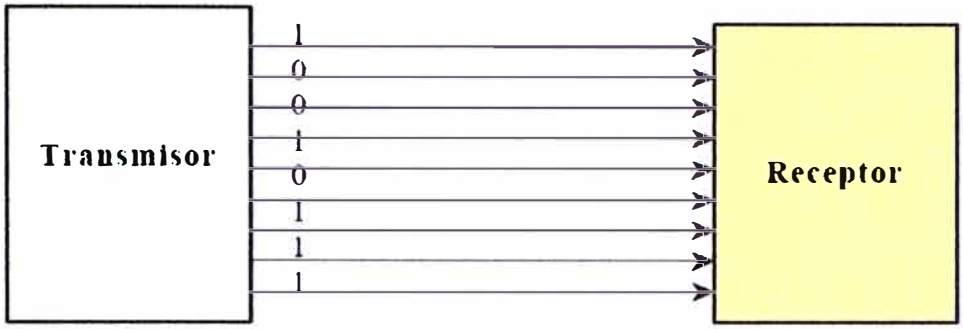

Figura 5.9 Comunicación paralela

Comunicación en Serie: En cambio, la comunicación en serie sólo utiliza una línea para la transmisión de datos, y opcionalmente alguna línea o líneas para protocolo. Por ejemplo, en la siguiente figura se muestra como se transmitiría en serie el mismo dato (97h):

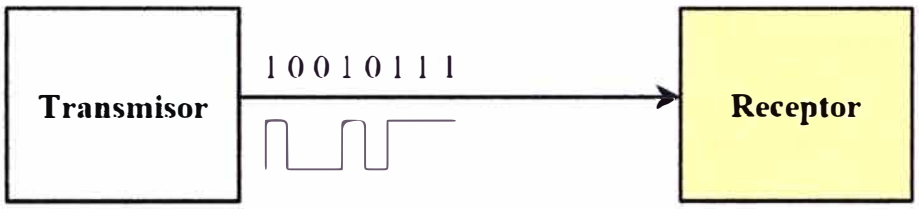

Figura 5.10 Comunicación serie

La desventaja obvia de la comunicación serie es que los bits de un dato se envían de a uno por uno, de manera que mientras que la comunicación en paralelo envía en un ciclo un dato de 8 bits, a la comunicación serie le toma más de 8 ciclos (ya que además del dato en la comunicación serie se requiere agregar algunos bits de sincronización.

Sin embargo, debido a que la comunicación serie requiere sólo una línea para la transmisión esto abarata los costos en líneas de transmisión y no sólo esto, ya que este hecho también hace posible que los datos puedan ser enviados no necesariamente por un conductor eléctrico, sino inclusive por aire o por el vacío si en lugar de pulsos eléctricos se usan impulsos electromagnéticos, tales como: ondas de radio, microondas, pulsos luminosos, infrarrojo, ultrasonido, láser (a través de fibra óptica), etc.

Protocolo de comunicación serie

A diferencia de la comunicación en paralelo, en la comunicación en serie se hace necesario establecer métodos de sincronización para evitar la interpretación errónea de los datos transmitidos. Para ilustrar esto se considera la siguiente información en serie:

...0100110001001100100...

Figura 5.11 Ejemplo secuencia de datos

Esta información puede interpretarse de diversas maneras, (aún si se recibe a la velocidad adecuada) dependiendo del punto de inicio de separación de datos, por ejemplo, una posible interpretación sería como sigue:

... 01	00110001	00110010	0 ...

Figura 5.12 Ejemplo de interpretación

Que interpretado como códigos ASCII corresponde a los caracteres '1' y '2'. Sin embargo, otra posible interpretación es:

... 010	01100010	01100100

Figura 5.13 Ejemplo de interpretación

Que corresponde a los caracteres 'b' y 'd' (código ascii 98 y código ascii 100).

Sincronización de bit. Una manera de resolver el problema anterior es la sincronización de bits que puede realizarse por varios métodos:

a) Enviar por una línea adicional una señal de reloj que indique el centro de las celdas de bits en la línea de datos (**datos no - auto reloj**).

b) Enviar con cada bit y por la misma línea de datos información que permita extraer la señal de reloj (**datos auto reloj**).

c) Lograr mediante alguna estrategia que los relojes de transmisión y de recepción se mantengan en fase continuamente.

Codificación no auto reloj.- En la figura siguiente se muestran las tres codificaciones de una línea de datos:

RZ.- Una celda de bit es 1 si contiene un impulso positivo y un 0 si no lo contiene.

NRZ.- La celda contiene un 1 o 0 de acuerdo al nivel de la señal (constante) en la celda.

NRZI.- La celda de bit contiene un 1 si hay una transición y un 0 si no la hay.

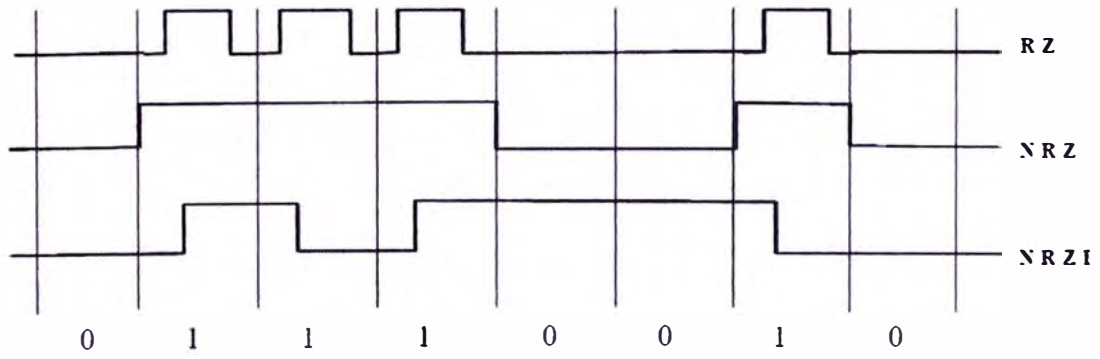

Figura 5.14 Ejemplo codificación no autoreloj

Como puede verse, en estos sistemas (RZ, NRZ y NRZI) las secuencias de ceros no contienen ninguna transición que permita ubicar la situación de las celdas de bit. De hecho, el formato NRZ no la contiene ni en los unos.

Codificación auto reloj.- Algunos métodos que contienen en la misma línea de datos información adicional para determinar la velocidad del reloj a costa de disminuir la cantidad de información útil a la mitad que los métodos no-auto reloj. En la siguiente figura se muestran los más utilizados, como son:

PE.- Codificación de fase.

FSC.- Codificación por cambio de frecuencia

FM.- Modulación de frecuencia.

MFM y **M2FM**.- Modulación de frecuencia modificadas.

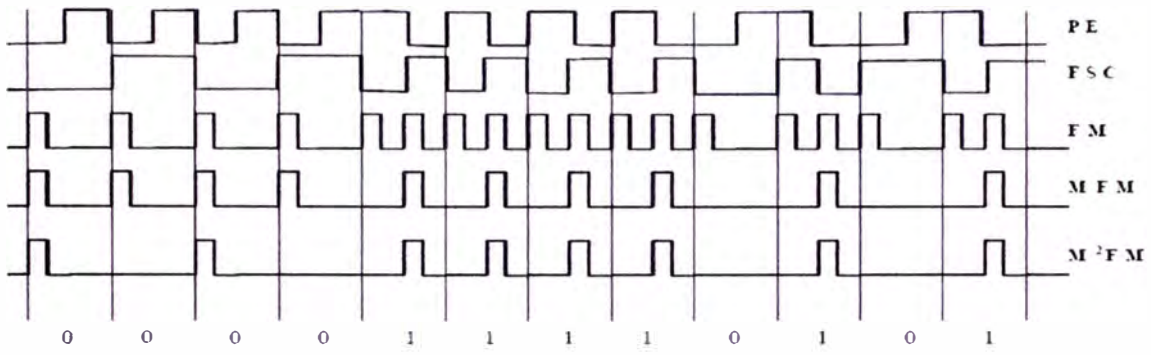

Figura 5.15 Codificación autoreloj

Los métodos auto reloj son muy útiles cuando la velocidad de transmisión no es constante, por ejemplo, cuando los datos han sido grabados en un medio magnético giratorio, por ejemplo discos, cintas magnéticas, etc.

Sincronización de carácter.-Algunos sistemas utilizan líneas adicionales que envían impulsos para indicar el inicio de un bloque de caracteres. Otros sistemas que no requieren líneas adicionales a la línea de datos son:

Método Asíncrono. Cada carácter va señalizado mediante dos bits: **un bit de inicio y un bit de paro**, estos dos bits permiten al receptor reconocer el inicio y el final de cada carácter. La especificación RS404 de EIA (Electronic Industries Association) define las características del método asíncrono para transmisión en serie de acuerdo a las siguientes reglas:

a) Cuando no se envían datos la línea debe mantenerse en estado 1.

b) Cuando se va a mandar un carácter se envía primero un bit de inicio de valor 0.

c) A continuación se envían todos los bits del carácter a transmitir al ritmo marcado por el reloj de transmisión.

d) Después del último bit del carácter enviado se envía un bit de paro de valor 1.

Método Síncrono. Cada mensaje o bloque de transmisión va precedido de unos caracteres de sincronismo. Así, cuando el receptor identifica una configuración de bits igual a la de los caracteres de sincronismo da por detectado el inicio y el tamaño de los datos.

Observación: Para el usuario de un microcontrolador que posee una USART o sistema similar la manera detallada como el sistema logra establecer la comunicación resulta transparente a él, ya que sólo tiene que configurar el protocolo del transmisor y del receptor para que estos logren la comunicación adecuada, es decir, el usuario usualmente sólo debe configurar: tipo de comunicación (síncrona o asíncrona) velocidad de transmisión en Baudios (bits por segundo) longitud de los datos bits de inicio y de paro, bits de paridad, etc.

La USART del PIC16F877A

La USART del PIC puede ser configurada para operar en tres modos:

Modo Asíncrono (full duplex (transmisión y recepción simultáneas)),

Modo Síncrono – Maestro (half duplex)

Modo Síncrono – Esclavo (half duplex)

En este proyecto se usará el modo asíncrono.

5.5 Comandos AT

Los comandos AT (attention command) son instrucciones codificadas que conforman un lenguaje de comunicación entre el hombre y un Terminal MODEM.

Los comandos AT son cadenas de caracteres ASCII que comienzan con AT y terminan con un retorno de carro (ASCII 13). Cada vez que el MODEM recibe un comando este lo procesa y emite su respuesta dependiendo como se lo haya configurado al MODEM. El software del teléfono se comunica con el MODEM por medio de comandos AT. Este software le permite al teléfono en si comunicarse por medio de menús y el programa de comunicaciones transmite estas selecciones al MODEM en el formato que este requiere. De esta manera el MODEM realiza la tarea que se le ha comunicado. Para el uso de aplicaciones más específicas se necesita el uso de aplicaciones como *Hyperterminal* en el caso de Windows y *Minicom* en el caso de Linux.

La implementación de los comandos AT corre a cuenta del dispositivo GSM y no depende del canal de comunicación a través del cual estos comandos son enviados, ya sea cable serial, canal infrarrojo, Bluetooth, etc.

5.5.1 Objetivo de los comandos AT

Los comandos AT deben ser usados para el desarrollo de nuevos programas de comunicaciones y ajustar propiedades avanzadas del teléfono y módems inalámbricos.

Entre las funciones más usuales de los comandos AT se tienen:

- Configurar el teléfono para una conexión inalámbrica, a través de infrarrojos o por el sistema de bus o cable.
- Configurar el MODEM interno del teléfono para una conexión inalámbrica, a través de infrarrojos o por el sistema de bus o cable.

- Solicitar información sobre la configuración actual o estado operativo del teléfono o MODEM.
- Probar la disponibilidad del teléfono o MODEM.

5.5.2 Ejecución de comandos AT

De los modos de operación mencionados, en el proyecto se utiliza el modo de comandos off-line, ya que cuando se emite un comando desde el sistema, se espera una respuesta al comando enviado por parte del teléfono, sin establecer una conexión para tener una transferencia continua de datos. Para ejecutar los comandos AT es necesario tener la lista de comandos que reconoce el teléfono.

Además para establecer conexión del teléfono con el microprocesador PIC, es necesario utilizar un cable de datos. El objetivo de utilizar los comandos AT es el extraer la información del teléfono y a la vez cargar información para responder a un determinado evento. Para el estudio de los comandos AT, se ha hecho una división de los comandos utilizados en el proyecto.

a. Comandos de configuración

Este tipo de comandos permiten cambiar la configuración interna del MODEM integrado del teléfono entre los que se tiene:

- **AT,** este es un comando de atención, cuya función es monitorear si existe una buena conexión en el canal de comunicación, si la conexión es buena el teléfono responde OK, en la Figura 5.16 se ilustra el código de programación que permite realizar esta acción a través del microprocesador PIC.

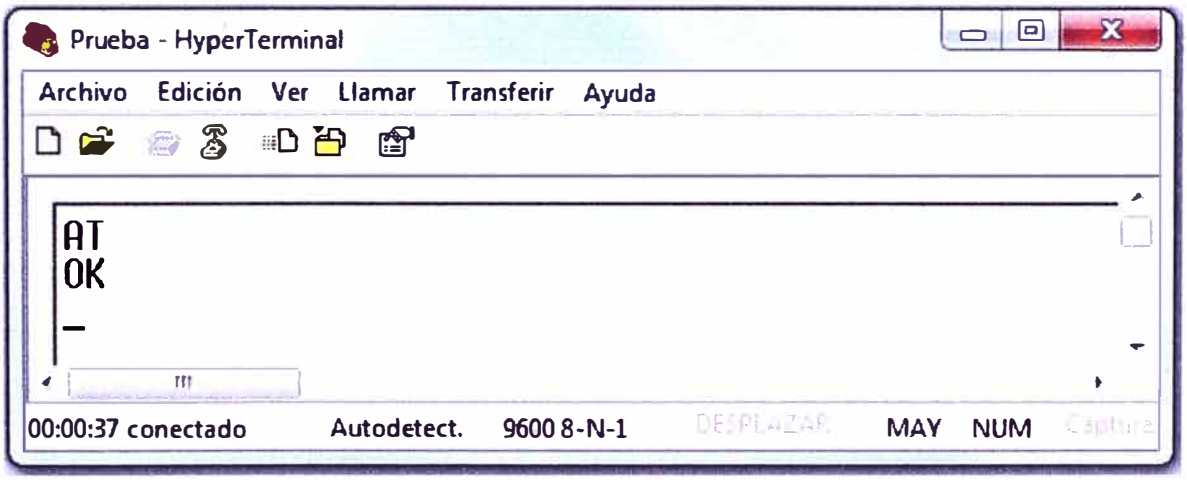

Figura 5.16 Ejecución del comando AT

- **AT + CMGF = "A",** este comando permite elegir el modo de interpretación de los datos por parte del teléfono, si la equivalencia de A es igual a "1", los datos son interpretados en modo texto, es decir la secuencia de caracteres que se envía al teléfono son ASCII normales. Si la equivalencia de A es igual a "0", los datos son interpretados en modo PDU (Protocol Data Unit), por lo tanto los datos enviados al teléfono deben ser

interpretados como caracteres HEX (hexadecimales). para el sistema se ha tomado en cuenta la primera opción, debido a que la comunicación entre el sistema y el teléfono será monitoreada, por tanto la interpretación de datos será mucho más sencilla. El teléfono al recibir este comando responde con OK, indicando que la petición ha sido aceptada. En la figura 5.17 se ilustra el código de programación que permite realizar esta acción a través del microprocesador PIC.

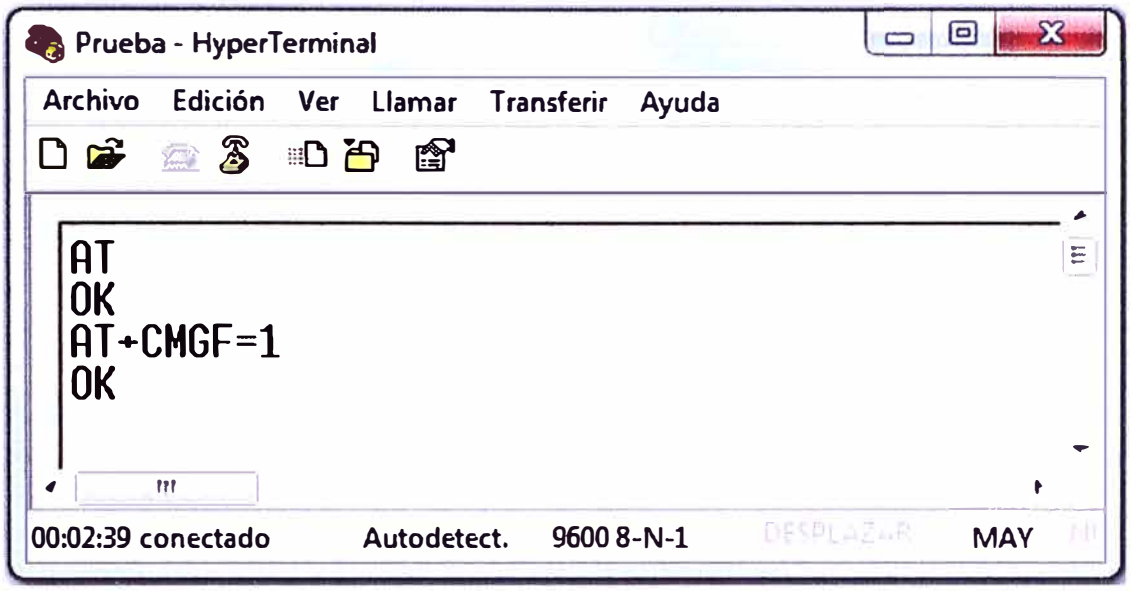

Figura 5.17 Ejecución del comando AT+CMGF=1

b. Comandos para envío de SMS

Luego de que el teléfono ha sido configurado en el modo texto, una de las necesidades del sistema es el envío de SMS, mediante el cual podrá responder a un evento solicitado.

Se analiza el menú de mensajes en el software de cualquier teléfono celular, se observa que posee básicamente tres librerías como son: Buzón de Mensajes Recibidos, Buzón de Mensajes enviados y Buzón de Mensajes o elementos no Enviados. Cuando un mensaje llega a un teléfono celular este es almacenado en la carpeta de mensajes recibidos; dentro de esta carpeta el mensaje aparecerá como un mensaje no leído y una vez que es abierto, el mensaje pasará a ser un mensaje leído, este caso dentro de los comandos AT es interpretado como REC UNREAD y REC READ respectivamente.

Ahora en el caso opuesto, un usuario desea enviar un mensaje, escribe la información dentro de un SMS y lo envía, esta información puede tomar dos caminos, primero el mensaje no es enviado por cuestiones de congestión de red, el teléfono guarda el mensaje dentro de la librería Buzón de Mensajes no Enviados. Pero si la red no presenta congestión el mensaje es enviado con éxito, razón por la cual el teléfono guarda este mensaje en la librería Buzón de Mensajes Enviados, estas dos situaciones dentro de los comandos AT son vistas como STO UNSET y STO SET respectivamente.

Al hablar de este tema, se trata de indicar que si el sistema desea enviar o recibir un mensaje, debe saber a qué librería debe apuntar en el teléfono para alcanzar su objetivo.

Pero como se utiliza un teléfono GSM, hay que tomar en cuenta que posee dos memorias que tienen las misma librerías, por lo tanto el sistema también tiene que identificar cual es la memoria en la que se está trabajando. Para enviar un SMS, el teléfono requiere de dos comandos que son: el AT+CMGW y AT+CMSS.

El Comando **AT+CMGW = "# Telefónico",** permite cargar en el teléfono la información del SMS, al número que se especifica dentro de los parámetros del comando (# Telefónico). La Figura 5.18 indica la utilización de este comando.

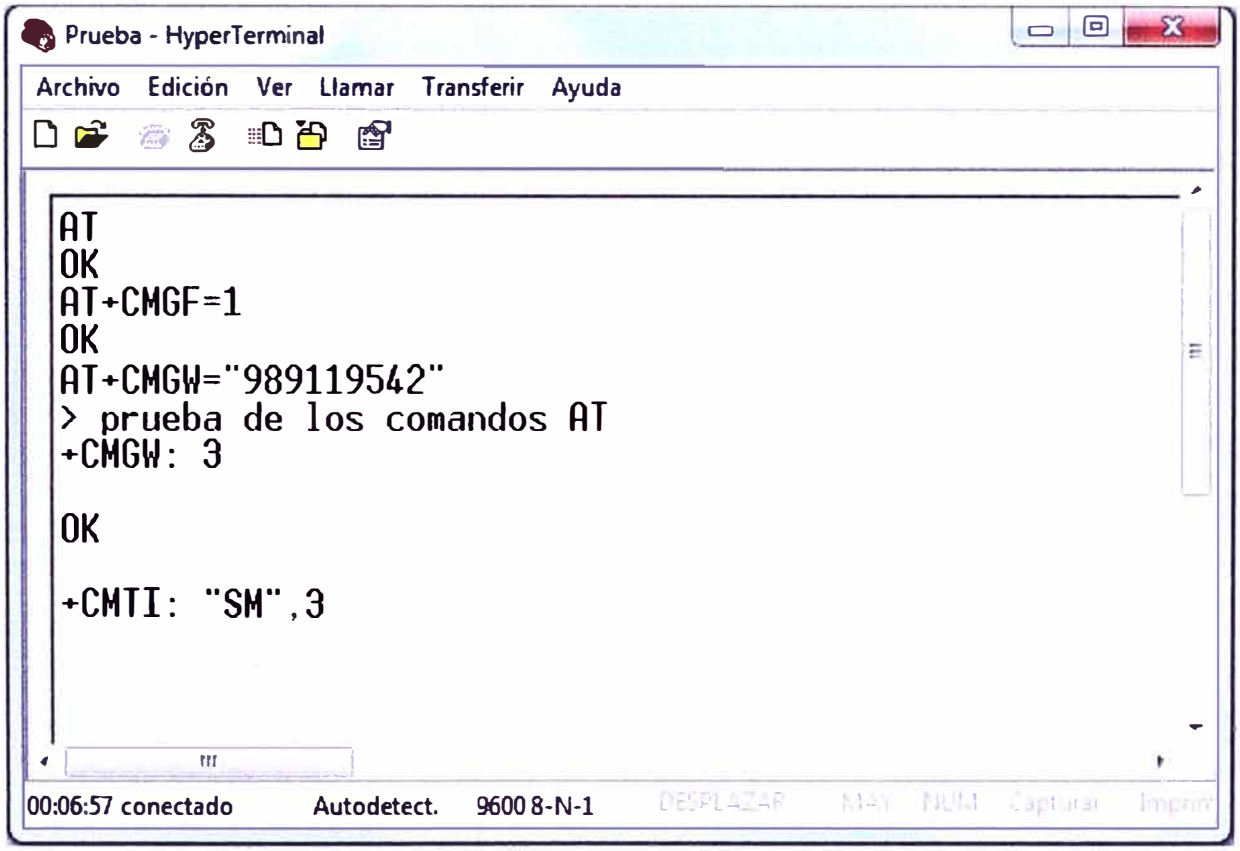

Figura 5.18 Comando AT+CMGW, para cargar un SMS en el teléfono

Cuando el comando es enviado con el número telefónico, el teléfono responde con un signo mayor que ">", de esta manera indica que se debe ingresar la información del mensaje, una vez que la información ha sido ingresada se adhiere el carácter ctrl.+Z, de esta manera indica el final del mensaje al teléfono, por tanto si la ejecución del comando ha sido correcta el teléfono responde OK.

El Comando **AT+CMPS = "LOCALIDAD MEM",** permite enviar el SMS cargado en el teléfono, al igual que haría la tecla "SEND". El dominio "Localidad de MEM", indica la posición del mensaje dentro de la memoria (ya sea en la memoria interna del teléfono o en la SIM) al recibir este comando el teléfono responde OK cuando se ha ejecutado en forma correcta, como se muestra en la Figura 5.19.

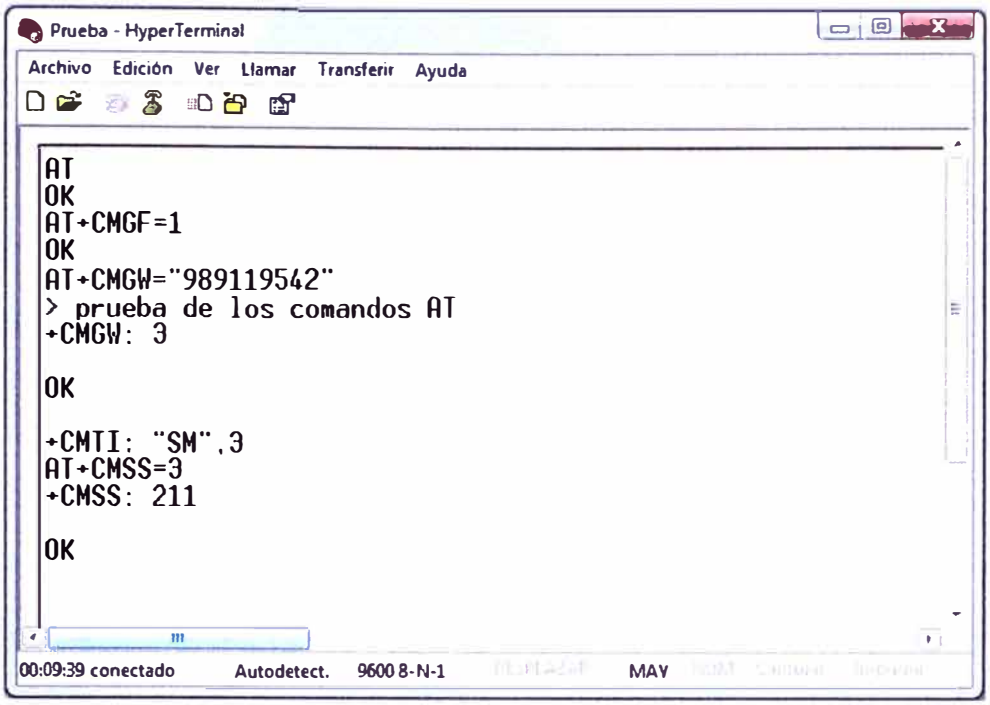

Figura 5.19. Comando AT+CMSS=3, que envía el SMS

c. Comandos para recepción de SMS

Cuando se carga un mensaje en el teléfono, éste se guarda dentro de la librería Buzón de Mensajes no Enviados en la memoria SIM, para el caso de recepción de mensajes, necesariamente el sistema debe apuntar a la memoria del teléfono donde el mensaje recibido es guardado. En Figura 5.20, muestra la extracción del SMS desde la memoria SIM, en el cual se encuentra por defecto. Nótese que no se realizó ninguna instrucción previa para cambiar de memoria, antes de la extracción del SMS.

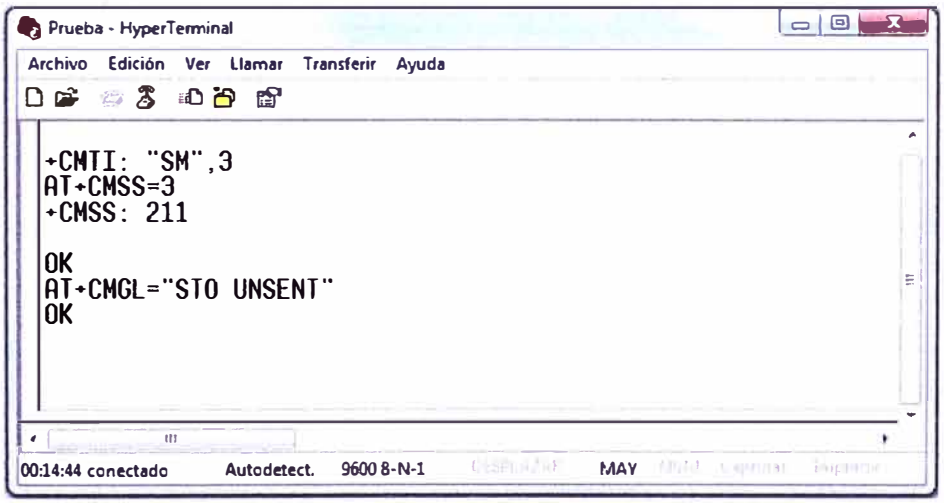

Figura 5.20 Extracción del SMS mediante el Comando AT+CMGL="STO UNSENT"

Con esto surge la necesidad de tener un comando que permite pasar de la memoria SIM hacia la memoria interna del teléfono y viceversa.

Posteriormente, el sistema debe extraer el SMS. Para ello se tiene las siguientes instrucciones:

El comando AT+CPMS = "Memoria", permite cambiar de memoria internamente dentro del teléfono, en el área "Memoria" se debe especificar a qué memoria se debe apuntar, identificando a la SIM como "SM" y a la memoria interna del teléfono como "ME". La Figura 5.21, muestra la ejecución de dicho comando.

Una vez que el sistema se encuentre en la memoria, necesita un comando que le permita elegir entre las librerías en donde se encuentra el mensaje. Para esto se tiene la siguiente instrucción:

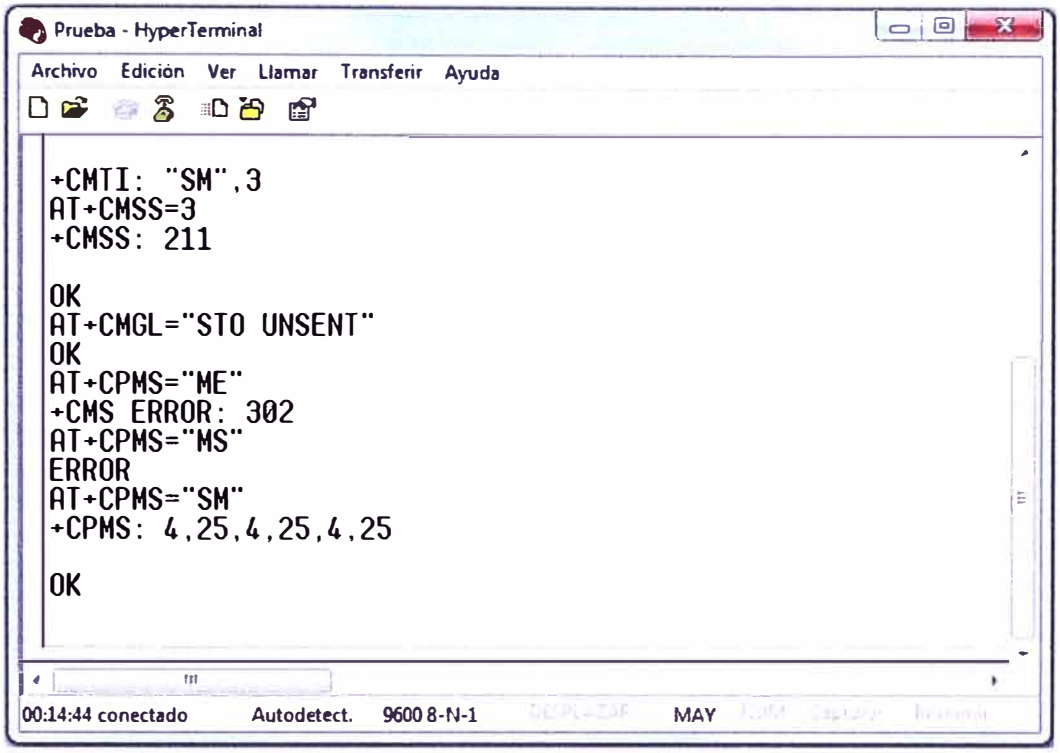

Figura 5.21 Cambio de Memoria mediante el comando AT+CPMS

El comando AT+CMGL = "LIBRERÍA", permite desplegar los mensajes contenidos en una librería. Para apuntar una librería, en el dominio "LIBRERÍA" el sistema debe ingresar las siglas asignadas por los comandos AT para cada librería. Por ejemplo, se desea ver el contenido de todos los mensajes existentes en el teléfono; se debe utilizar la instrucción "ALL", para obtener el listado de los mensajes recibidos en el dominio. Hay que ingresar "REC UNREAD", para el listado de mensajes leídos "REC READ", para el listado de mensajes no enviados "STO UNSENT" y para los mensajes enviados "STO SENT".

En la Figura 5.22, se muestra un ejemplo donde al digitalizar este comando con el dominio "ALL", se despliega la lista de mensajes contenidos en todas las librerías. Mientras que en la Figura 5.23, se muestra la utilización de los dos comandos, para apuntar la memoria interna del teléfono y como paso siguiente se lee un mensaje recibido que se encuentra en la librería de los mensajes no leídos ("REC UNREAD"), en este caso no se muestra nada puesto que todos los mensajes fueron leídos.

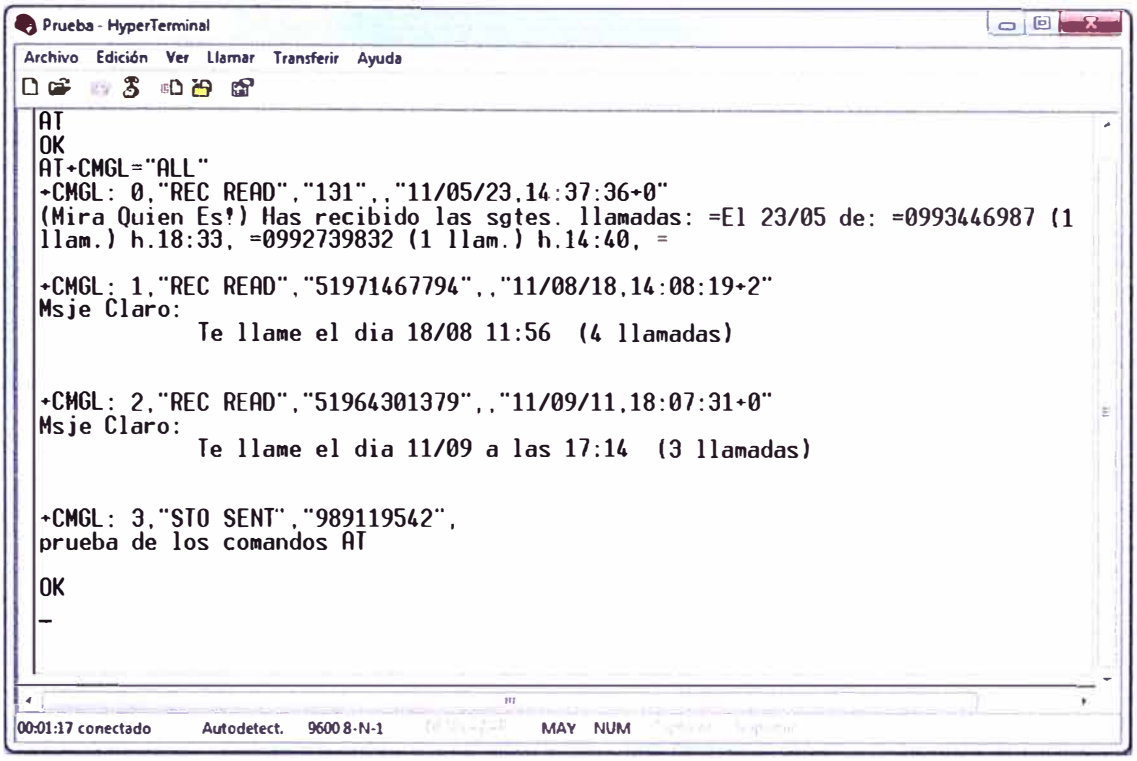

Figura 5.22 Comando AT+CMGL="ALL"

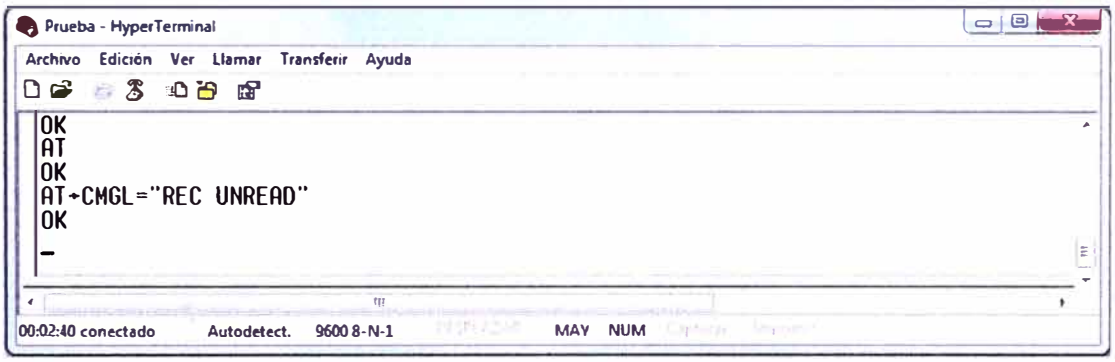

Figura 5.23 Utilización de comandos para leer mensajes "no leídos"

Para borrar un SMS del teléfono, el sistema primero debe apuntar a la localidad de memoria en la que se encuentra el mensaje, posteriormente indicar la librería en la cual está el mensaje y por último con el comando AT + CMGD eliminar el mensaje.

El comando AT+CMGD = "LOCALIDAD MEM", permite eliminar un mensaje de una determinada localidad de memoria en el teléfono. El dominio LOCALIDAD MEM" indica la posición que un mensaje ocupa en la memoria del teléfono. En la Figura 5.24, se indica el proceso para borrar un mensaje leído de la memoria interna del teléfono. Con los comandos AT+CPMS = "ME", AT+CMGL = "REC READ" y AT+CMGL=1, Se indica al teléfono que el primer mensaje de la librería Buzón de Mensajes Recibidos, debe ser borrado.

Este resultado se muestra en la Figura 5.25, mediante la solicitud del comando AT + CMGL = "REC READ". Al desplegarse los mensajes leídos se observa que el mensaje 1 ha sido eliminado.

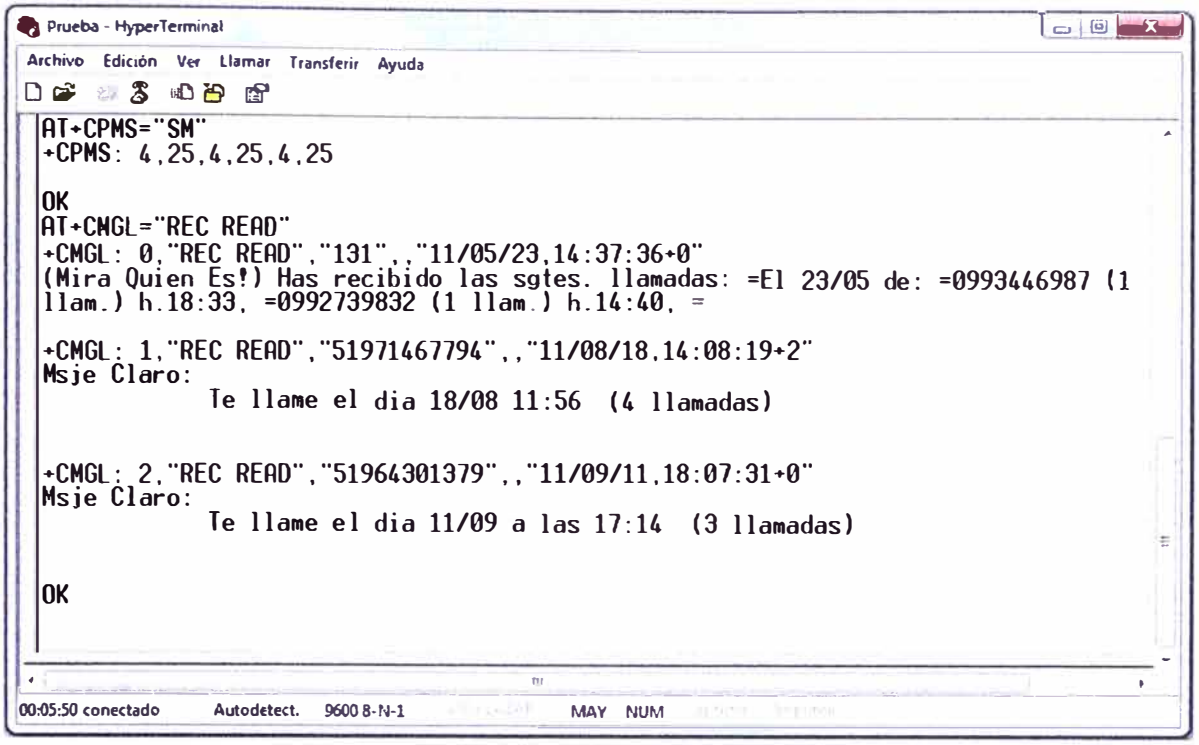

Figura 5.24 Lectura de mensajes leídos

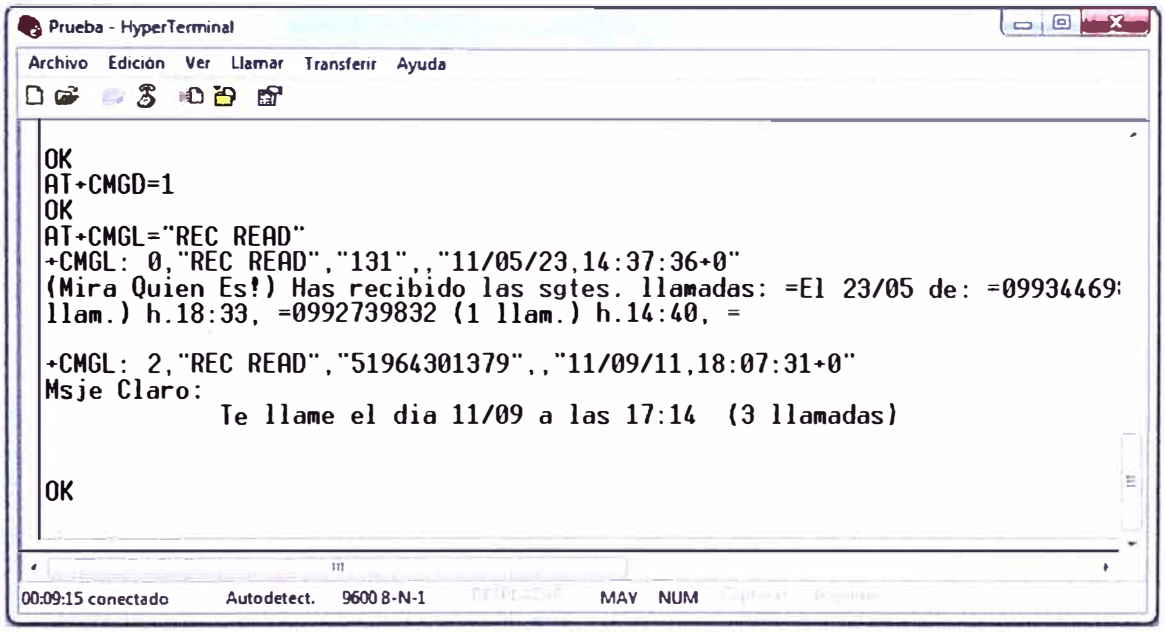

Figura 5.25 Comando AT+CMGD=1 para borrar el mensaje.

La información devuelta por los comandos de envío y recepción de SMS, muestran datos importantes como: número de teléfono de donde se emitió el mensaje, hora, fecha, localidad de memoria que ocupa el mensaje.

d. Código de resultado y error

Cuando se envía un comando desde el computador hacia el MODEM integrado, la respuesta es terminada por un código de resultado Resol Code.

Este es el mensaje que envía el MODEM interno del teléfono celular hacia el computador. Estos códigos de resultado deben ser usados para confirmar una correcta operación o

identificar un problema con algún comando.

- Código de Resultado OK para un comando Válido
- Código ERROR para un comando inválido

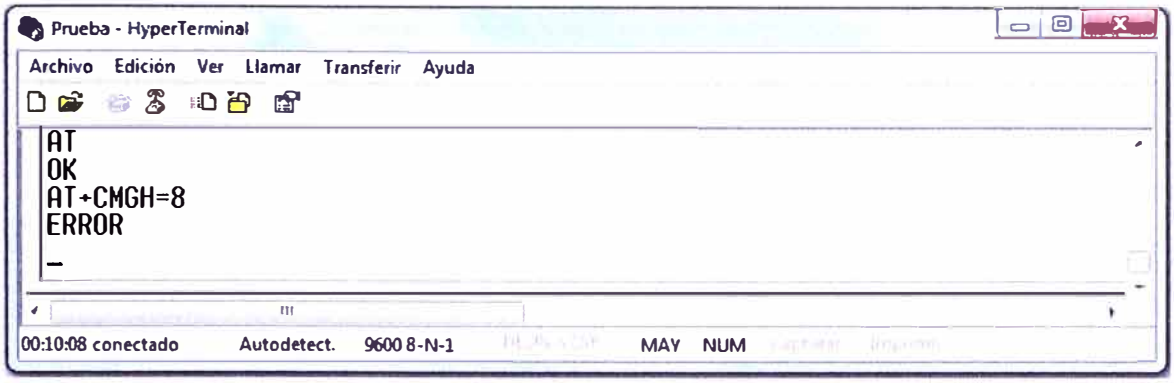

Figura 5.26 Tipos de códigos de Resultado

5.6 Desarrollo del programa para el PIC16F877A

El programa en el microcontrolador PIC debe realizar las siguientes tareas, para cumplir con los objetivos planteados:

- El microcontrolador debe establecer un enlace con el celular receptor por medio del puerto serial.
- Una vez establecida la comunicación entre los dos dispositivos se debe programar el microcontrolador para que transmita los caracteres que forman los comandos AT.

5.6.1 Programa principal

El programa principal del microcontrolador empieza con una definición de registros y variables, asignación de pines de entrada / salida y configuración de registros de comunicación serial asincrónica.

Entre las subrutinas que se emplean en el programa se encuentran: la configuración del módem celular para trabajar en modo de mensajes de texto SMS, lectura de mensajes de texto que contiene el comando para inicio de transmisión, adquisición de datos, procesamiento de la información y envío de la trama de datos como mensaje SMS, en la Figura 5.27 se muestra el diagrama de flujo del programa principal.

5.6.2 Subrutinas utilizadas en el programa

En las páginas siguientes se describe las subrutinas empleadas en el desarrollo del programa.

a. Configuración del equipo celular

El micro PIC emplea los pines RC6 y RC7 del puerto serial asincrónico y las instrucciones HSEROUT y HSERIN para comunicarse con el equipo celular a una velocidad de 9600 bps. Los parámetros seriales y el baud rate se especifican usando la instrucción DEFINE para colocar el registro de transmisión habilitado se utiliza la instrucción

DEFINE HSER_TXSTA 20h //El baud rate se identifica con la instrucción

DEFINE HSER_BAUD 9600 //Para colocar el registro de recepción habilitado se emplea la instrucción.

DEFINE HSER_RCSTA 90h

DEFINE OSC 20 //Esta línea indica que se está utilizando el oscilador de 20 MHz

El PIC para efectuar la interfaz con el equipo celular primero habilita los pines de transmisión y recepción serial, luego envía los comandos de inicio de protocolo, operación con mensajes SMS y funcionamiento en modo de texto, en la Figura 5.28 se observa el diagrama de flujo de esta subrutina.

El equipo GSM responde a cada comando con un reconocimiento, si el comando es correcto envía la palabra OK.

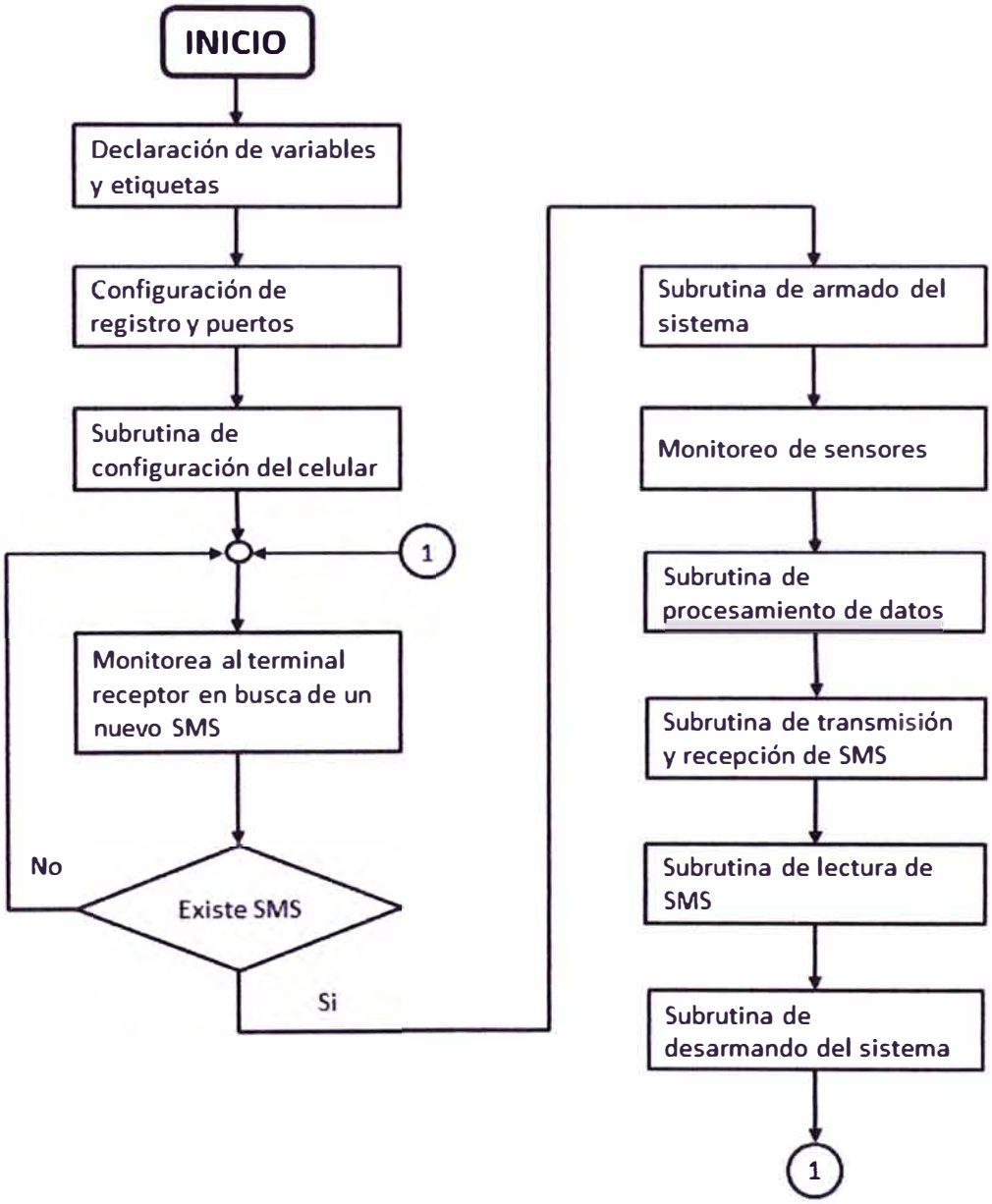

Figura 5.27 Diagrama de flujo del programa principal

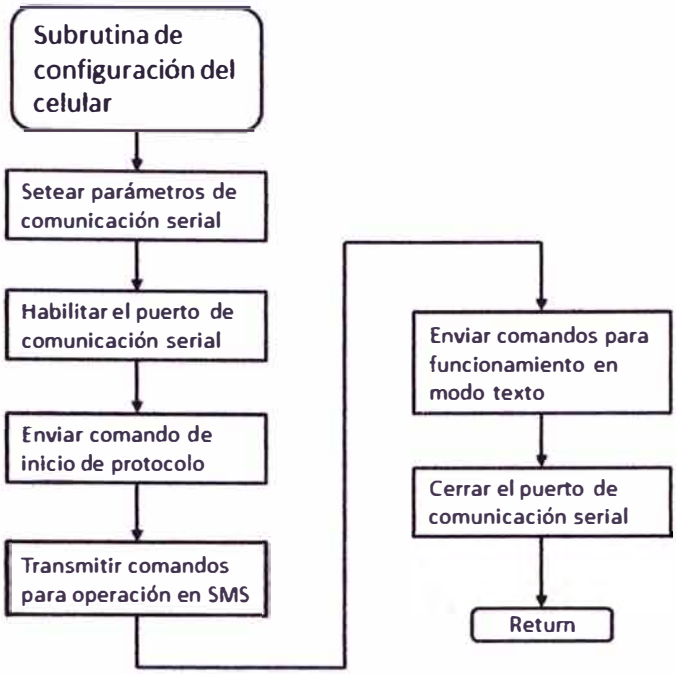

Figura 5.28 Diagrama de flujo de la subrutina de configuración del equipo celular

b. Armado y desarmado del sistema

Esta subrutina se produce cuando el usuario envía a la central un mensaje.

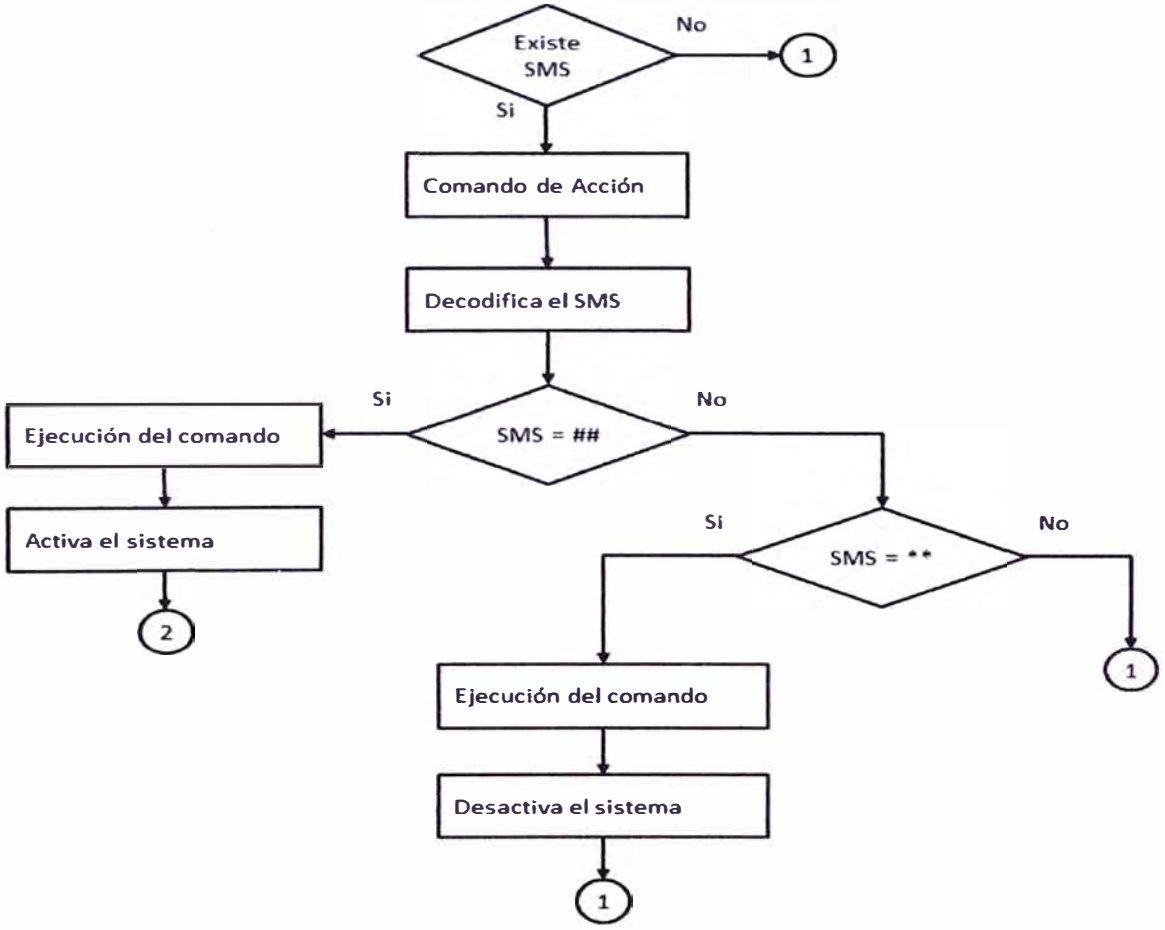

Figura 5.29 Diagrama de flujo de la subrutina de Armado y Desarmado del sistema

c. Monitoreo de sensores

Es el proceso en donde se obtienen los datos de los sensores. A continuación se explica cada una de las subrutinas.

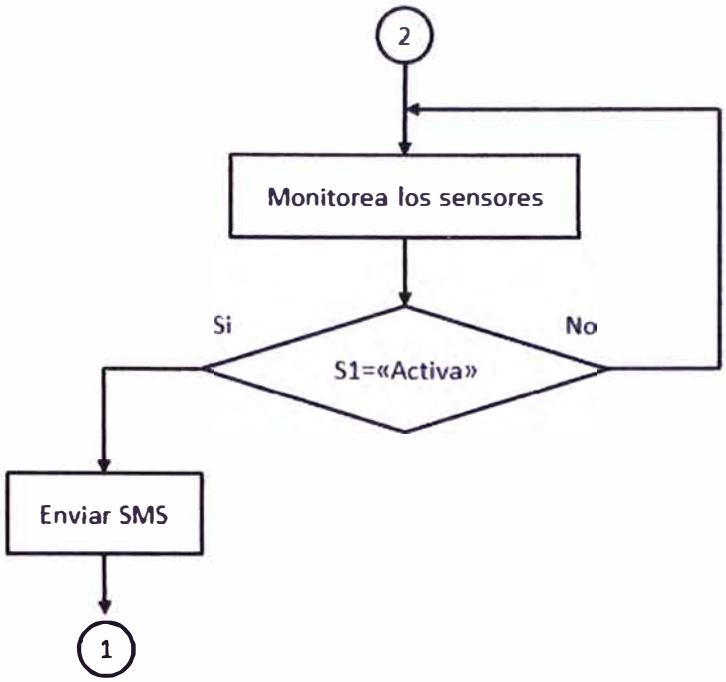

Figura 5.30 Diagrama de flujo de la subrutina de monitoreo de sensores

Subrutina del sensor de movimiento

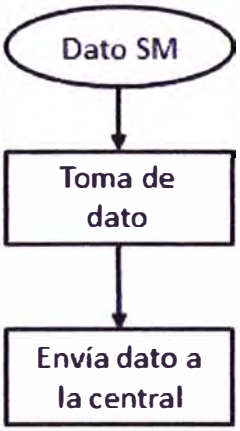

Figura 5.31 Diagrama de flujo de la subrutina del sensor

d. Procesamiento de la información

El microcontrolador una vez que obtiene y almacena la información pasa a una etapa de procesamiento donde lee las variables adquiridas y coloca los segmentos de información en un paquete, cada segmento lleva su propio encabezado que permite su identificación.

El paquete contiene un preámbulo donde se encuentra la identificación del móvil del que procede y el número de mensaje.

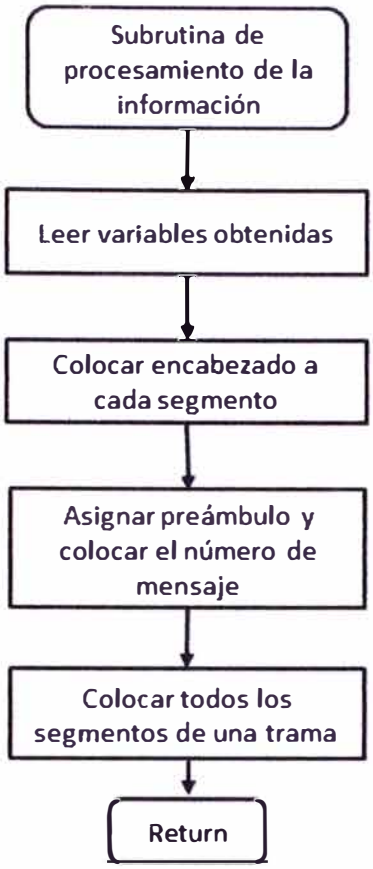

Figura 5.32 Diagrama de flujo de la subrutina de procesamiento de datos

e. Envío y recepción de trama de datos como mensaje SMS.

Esta subrutina se efectúa cada vez que se produce un evento en la adquisición de datos de los sensores, el primer paso del microcontrolador es leer las tramas que se procesaron, después abrir el puerto de comunicaciones, enviar el comando con el paquete de tramas al número telefónico que fue asignado para que llegue el mensaje. Luego se encarga de recibir los mensajes enviados por el usuario a la central y ejecutar cada una de las acciones. En la Figura 5.33 se muestra el diagrama de flujo de esta subrutina.

f. Lectura de mensajes de texto

Esta subrutina (Figura 5.34) lee los mensajes de texto que llegan a la central provenientes del usuario autorizado. Empleando la interrupción por recepción serial el PIC conoce el momento en que llega un mensaje y procede a leerlo enviando el comando respectivo hacia el módem GSM.

Cuando el microcontrolador lee el mensaje SMS primero identifica el terminal de procedencia, si el número corresponde al de la estación central continua con la lectura del contenido del mensaje, por el contrario si el número es desconocido el mensaje es descartado. El contenido del mensaje corresponde a comandos para inicio o finalización de transmisión que son asignados a una variable, luego de esto finaliza la subrutina.

Figura 5.33 Diagrama de flujo de la subrutina de envío y recepción de SMS

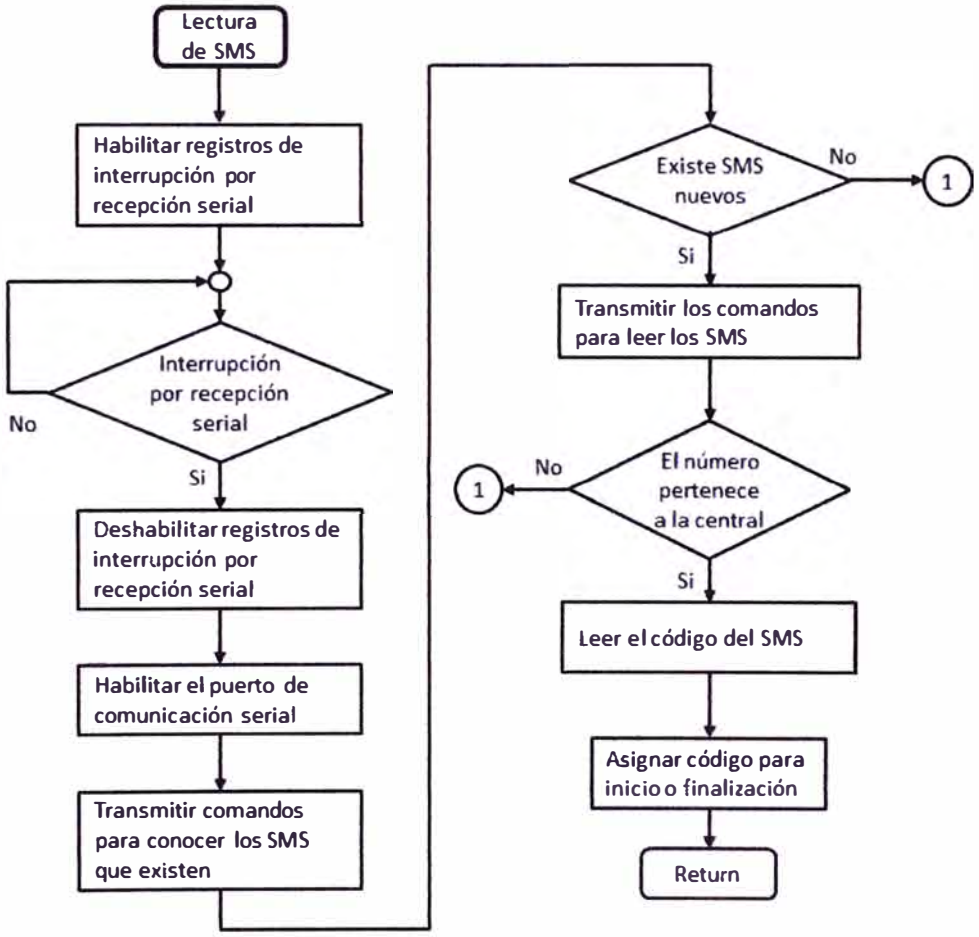

Figura 5.34 Diagrama de flujo de la subrutina de lectura de mensajes

5.7 Proceso de funcionamiento del sistema

Se desarrollan los siguientes aspectos:

- Activación del sensor de movimiento.
- Conexión entre el celular y el microcontrolador PIC.
- Enviando comando al celular para el envío de SMS
- Recepción del mensaje por el microcontrolador PIC.

5.7.1 Activación del sensor de movimiento

Como ya se explicó se eligió un sensor de movimiento que es capaz de conectarse directamente a uno de los puertos de entrada de el microcontrolador PIC. En la Figura 5.35 se muestra la conexión del microcontrolador con el sensor.

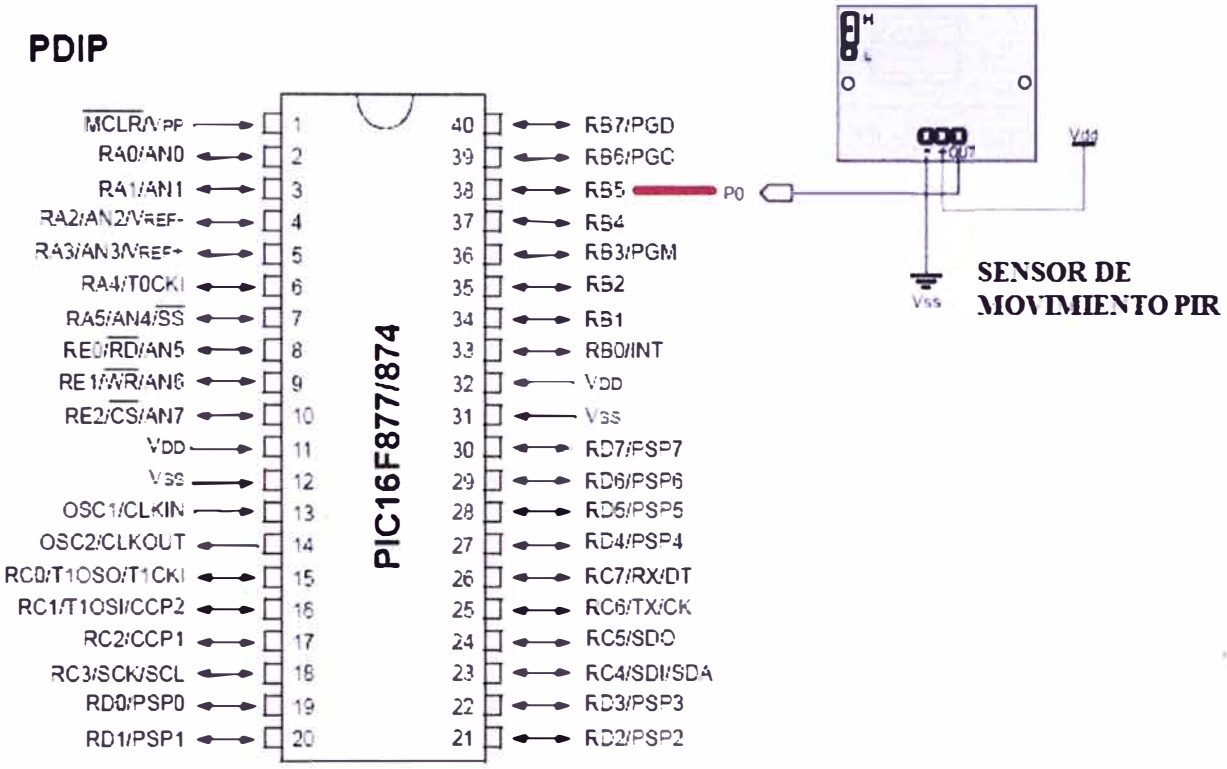

Figura 5.35 Esquema de conexión del sensor al puerto RB5 del PIC

Se debe recordar que una de las características de porque escogió el sensor de movimiento PIR es debido a que brinda una salida de 1 lógico que es 5 V. Dicho esto el PIC al recibir la señal por el puerto RB5, iniciará el proceso de envío del SMS hacia el celular remoto. Para realizar esto el sistema debe enviar la trama (comando AT) hacía el celular que se encuentra conectado al PIC. La trama a enviar sería:

AT+CMGF=1

AT+CMGS="679123456" [Presionar CR]

> SensorMovActv [Ctrl-Z]

5.7.2 Conexión entre el celular y el PIC

El celular tiene un voltaje de salida de 3.6 V y el microcontrolador un voltaje de 5 V.

Es por esa razón que se requiere de un MAX232 que no es otra cosa que un TTL-RS232

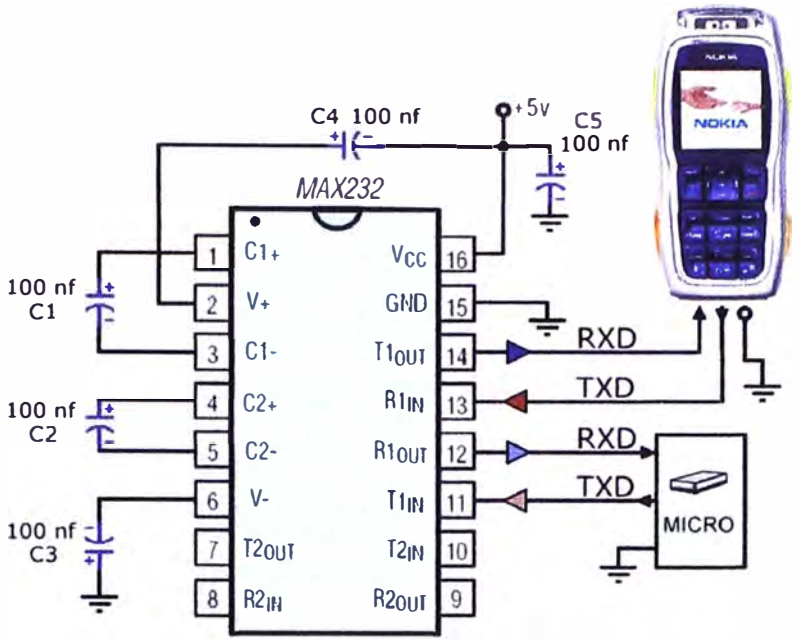

Figura 5.36 Esquema de conexión entre el móvil y el MAX232

Otra forma de conectar el terminal móvil con el microcontrolador es:

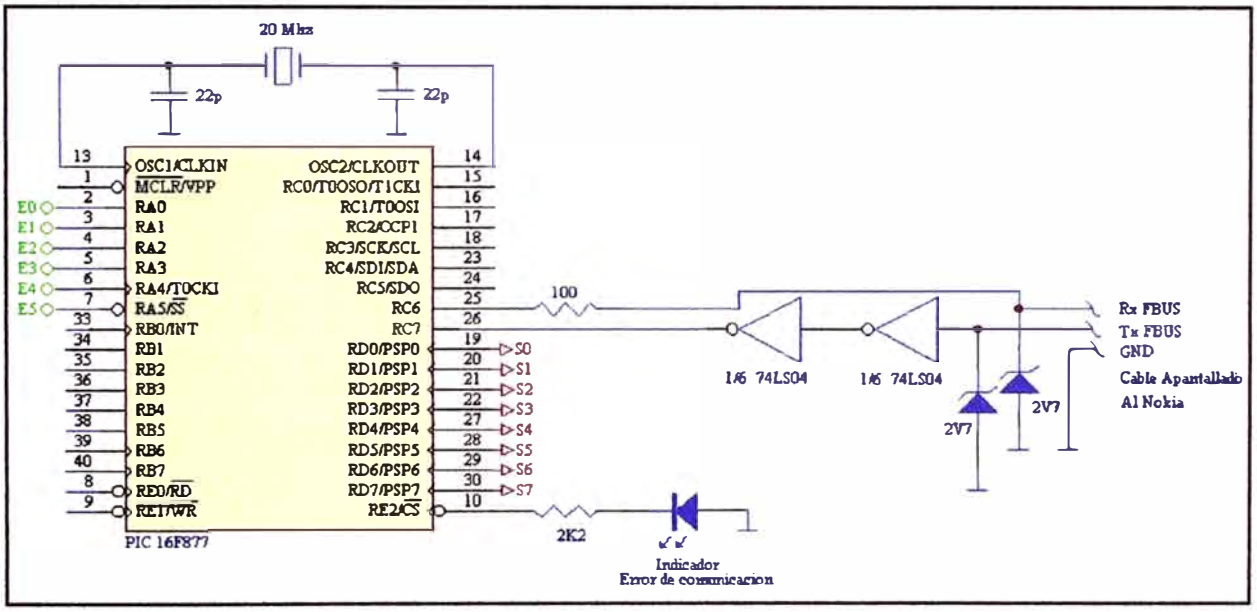

Figura 5.37 Conectando 2 negadores a la salida del puerto de Tx

En la figura se corrige el nivel bajo de voltaje del celular añadiendo dos negadores en serie a la salida del puerto Tx del celular.

También se utiliza un divisor de voltaje para el puerto de Rx del celular ya que el voltaje de salida del microcontrolador es de 5 V y el celular trabaja con 3.6 V.

5.7.3 Enviando comando al celular para el envío de SMS

Esto se puede hacer con el siguiente código de programación. Una vez realizada la conexión el PIC y el celular, cuando se activa el sensor de movimiento el sistema ejecutará el siguiente código.

```
...
while(true){
printf("AT\r\n"); // Le llama la atención al teléfono.
delay_ms(2000); //espera 5 segundos
printf("AT+CMGF=1\r\n"); //configura modo texto
delay_ms(1000);
printf("AT+CMGS=\"679123456\"\r\n"); //679123456 es el destinatario.
delay_ms(1000); //del celular al que se envía el SMS
printf("SensorMovAct");
putc(26); //ascii(26)=(ctrl+z) envia el mensaje
printf("\r\n");
delay_ms(1000);
...
```

5.7.4 Recepción del mensaje por el PIC

El PIC está constantemente revisando si existe un mensaje no leído, esto lo hace con el comando.

AT+CMGL="REC UNREAD"

Si el retorno del celular es diferente de OK entonces, el PIC procederá a leer el mensaje, si el mensaje recibido coincide con alguno de su base de datos ya establecida, el PIC ejecutará la acción programada correspondiente al mensaje, si no coincide con ninguno seguirá revisando la bandeja de entrada con el comando indicado líneas arriba.

CONCLUSIONES Y RECOMENDACIONES

1. El sistema está realizado bajo la Licencia Pública General de GNU, o "GPL de GNU", la cual hace que el software sea altamente escalable, ya que al ser software libre los programadores pueden realizar corrección de errores y mejoras en el código.
2. El sistema aprovecha la Red existente de las operadoras de Telefonía Móvil para poder conectar los terminales, esto a su vez hace del sistema sea dependiente del estado de dicha red, para el envío de MMS, esto disminuye los costos.
3. Si bien el sistema está orientado a la seguridad domiciliaria, puede ser adaptado a un sistema de Domótica y controlar diferentes artefactos como luces, televisores, equipos de sonido, etc. que existen en el domicilio.
4. Se puede modificar el sistema para que las órdenes ya no se envíen por medio de mensajes de texto, sino vía bluetooth enviar ordenes al microcontrolador, para realizar esto se debe estar dentro del rango de alcance de la señal bluetooth
5. Actualmente existen módulo GSM que son fáciles de conectar a un microcontrolador, con estos módulos ya no sería necesario la conexión entre el celular, el max232 y el microcontrolador.
6. Al trabajar con comandos AT se logró la comunicación entre el microcontrolador y el teléfono celular, viene a ser el idioma en que se comunican estos dispositivos.
7. Al utilizar la nube de comunicación GSM se obtiene un rango de alcance bastante aceptable, lo cual depende de la operadora con la que se esté trabajando.

ANEXO A
CÓDIGO PARA EL ENVÍO Y RECEPCIÓN DE MENSAJES MULTIMEDIA

Se crea el MIDlet MMSSend y el MMSReceive, dentro de package Vigilancia.mms

-Código MMSend

```
package Vigilancia.mms;

import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;
import java.io.IOException;
import javax.microedition.io.*;
import java.io.InputStream;
import java.util.Vector;
import javax.wireless.messaging.*;

public class MMSSend extends MIDlet implements CommandListener {
   private static Command CMD_EXIT = new Command("Salir", Command.EXIT, 2);
   private static Command CMD_SEND = new Command("Enviar", Command.ITEM, 1);
   private static Command CMD_ADD_PART = new Command("Añadir",
Command.ITEM, 1);
   private Display display;
   private String appID;
   private TextField subjectField;
   private TextField destinationField;
   private StringItem partsLabel;
   private Alert errorMessageAlert;
   private Alert sendingMessageAlert;
   private Displayable resumeScreen = null;
   private MMSMessage message;
   private PartsDialog partsDialog;

   public MMSSend() {
      appID = getAppProperty("MMS-ApplicationID");
      display = Display.getDisplay(this);
      Form mainForm = new Form("Nuevo MMS");
      subjectField = new TextField("Asunto:", null, 256, TextField.ANY);
      mainForm.append(subjectField);
      destinationField = new TextField("Número del destinatario: ", "mms://", 256,
TextField.ANY);
      mainForm.append(destinationField);
      partsLabel = new StringItem("Partes Añadidas:", "0");
      mainForm.append(partsLabel);
      mainForm.addCommand(CMD_EXIT);
      mainForm.addCommand(CMD_SEND);
      mainForm.addCommand(CMD_ADD_PART);
      mainForm.setCommandListener(this);
      errorMessageAlert = new Alert("MMS", null, null, AlertType.ERROR);
      errorMessageAlert.setTimeout(5000);
      sendingMessageAlert = new Alert("MMS", null, null, AlertType.INFO);
      sendingMessageAlert.setTimeout(5000);
      sendingMessageAlert.setCommandListener(this);
      resumeScreen = mainForm;
      message = new MMSMessage();
   }
```

```
    public void startApp() {
        display.setCurrent(resumeScreen);
    }
    public void pauseApp() {
        resumeScreen = display.getCurrent();
    }
    public void destroyApp(boolean unconditional) {
    }
    public void commandAction(Command c, Displayable s) {
        try {
            if ((c == CMD_EXIT) || (c == Alert.DISMISS_COMMAND)) {
                destroyApp(false);
                notifyDestroyed();
            } else if (c == CMD_ADD_PART) {
                if (partsDialog == null) {
                    partsDialog = new PartsDialog(this);
                }
                partsDialog.show();
            } else if (c == CMD_SEND) {
                promptAndSend();
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
    void show() {
        partsLabel.setText(Integer.toString(partsDialog.counter));
        display.setCurrent(resumeScreen);
    }
    Display getDisplay() {
        return display;
    }
    MMSMessage getMessage() {
        return message;
    }
     private void promptAndSend() {
        try {
            String address = destinationField.getString();
            message.setSubject(subjectField.getString());
            message.setDestination(address);
            String statusMessage = "Sending message to " + address + "...";
            sendingMessageAlert.setString(statusMessage);
            new SenderThread(message, appID).start();
        } catch (IllegalArgumentException iae) {
            errorMessageAlert.setString(iae.getMessage());
            display.setCurrent(errorMessageAlert);
        }
    }
}
 class SenderThread extends Thread {
    private MMSMessage message;
    private String appID;
    public SenderThread(MMSMessage message, String appID) {
        this.message = message;
```

```
        this.appID = appID;
    }
    public void run() {
        String address = message.getDestination() + ":" + appID;
        MessageConnection mmsconn = null;
        try {
            mmsconn = (MessageConnection)Connector.open(address);
            MultipartMessage mmmessage =

(MultipartMessage)mmsconn.newMessage(MessageConnection.MULTIPART_MESSAG
E);
            mmmessage.setAddress(address);
            MessagePart[] parts = message.getParts();
            for (int i = 0; i < parts.length; i++) {
                mmmessage.addMessagePart(parts[i]);
            }
            mmmessage.setSubject(message.getSubject());
            mmsconn.send(mmmessage);
        } catch (Exception e) {
            e.printStackTrace();
        }
        if (mmsconn != null) {
            try {
                mmsconn.close();
            } catch (IOException ioe) {
                ioe.printStackTrace();
            }
        }
    }
}
    class PartsDialog implements CommandListener {
    private static final Command CMD_BACK = new Command("Back", Command.BACK,
1);
    private static final Command CMD_NEXT = new Command("Next", Command.OK, 1);
    private static final Command CMD_OK = new Command("OK", Command.OK, 1);
    private static final Command CMD_CANCEL = new Command("Cancel",
Command.CANCEL, 1);
    private MMSSend mmsSend;
    private List typeList;
    public int counter = 0;
    public PartsDialog(MMSSend mmsSend) {
        this.mmsSend = mmsSend;
        String[] stringArray = { "Texto", "Imagen" };
        typeList = new List("Que desea añadir ?", Choice.EXCLUSIVE, stringArray, null);
        typeList.addCommand(CMD_BACK);
        typeList.addCommand(CMD_NEXT);
        typeList.setCommandListener(this);
    }
    public void show() {
        mmsSend.getDisplay().setCurrent(typeList);
    }
     public void commandAction(Command c, Displayable s) {
        try {
            if (c == CMD_BACK) {
```

```
            mmsSend.show();
        } else if (c == CMD_NEXT) {
            if (typeList.getSelectedIndex() == 0) {
                mmsSend.getDisplay().setCurrent(new TextDialog());
            } else {
                mmsSend.getDisplay().setCurrent(new ImageDialog());
            }
        }
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}
private class TextDialog extends Form implements CommandListener {
    private Displayable mainForm;
    private TextField text;
    private String mimeType = "text/plain";
    public TextDialog() {
        super("Añadir Texto");
        text = new TextField("Texto: ", null, 256, TextField.ANY);
        append(text);
        append("MIME-Type: " + mimeType);
        addCommand(CMD_OK);
        addCommand(CMD_CANCEL);
        setCommandListener(this);
    }
    public void commandAction(Command c, Displayable s) {
        try {
            if (c == CMD_OK) {
                String encoding = "UTF-8";
                byte[] contents = text.getString().getBytes(encoding);
                mmsSend.getMessage()
                        .addPart(new MessagePart(contents, 0, contents.length, mimeType,
                         "id" + counter, "contentLocation", encoding));
                counter++;
                mmsSend.show();
            } else if (c == CMD_CANCEL) {
                mmsSend.show();
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
private class ImageDialog extends Form implements CommandListener {
    private Displayable mainForm;
    private ChoiceGroup cg;
    private String mimeType = "image/png";
    private String[] resouces = { "/fotos/Duke1.png", "/fotos/diablo.jpg", "/fotos/Gas.png",
"/fotos/Ladron.png", "/fotos/fuego.png"};
    private String[] imagesNames = { "Pato", "Diablo", "Gas", "Ladron", "Fuego" };
    public ImageDialog() {
        super("Añadir Imagen");
        cg = new ChoiceGroup("Seleccione una imagen", Choice.EXCLUSIVE,
imagesNames, null);
```

```
            append(cg);
            append("MIME-Type: " + mimeType);
            addCommand(CMD_OK);
            addCommand(CMD_CANCEL);
            setCommandListener(this);
        }
        public void commandAction(Command c, Displayable s) {
            try {
                if (c == CMD_OK) {
                    int index = cg.getSelectedIndex();
                    String resouce = resouces[index];
                    InputStream is = getClass().getResourceAsStream(resouce);
                    byte[] contents = new byte[is.available()];
                    is.read(contents);
                    String contentLocation = imagesNames[index];
                    mmsSend.getMessage()
                            .addPart(new MessagePart(contents, 0, contents.length, mimeType,
                            "id" + counter, contentLocation, null));
                    counter++;
                    mmsSend.show();
                } else if (c == CMD_CANCEL) {
                    mmsSend.show();
                }
            } catch (Exception ex) {
                ex.printStackTrace();
            }
        }
    }
}
    class MMSMessage {
    private String destination;
    private Vector parts = new Vector();
    private String subject;

    private static boolean isValidPhoneNumber(String address) {
        String protocol = "mms://";
        if (!address.startsWith(protocol)) {
            return false;
        }
        String number = address.substring(protocol.length());
        char[] chars = number.toCharArray();
        if (chars.length == 0) {
            return false;
        }
        int startPos = 0;
        if (chars[0] == '+') {
            startPos = 1;
        }
        for (int i = startPos; i < chars.length; ++i) {
            if (!Character.isDigit(chars[i])) {
                return false;
            }
        }
        return true;
```

```
    }
    public String getSubject() {
        return subject;
    }
    public void setSubject(String subject) {
        this.subject = subject;
    }
    public String getDestination() {
        return destination;
    }
    public void setDestination(String destination) {
        if (!isValidPhoneNumber(destination)) {
            throw new IllegalArgumentException("Numero de telefono invalido");
        }
        this.destination = destination;
    }
    public MessagePart[] getParts() {
        MessagePart[] partsArray = new MessagePart[parts.size()];
        parts.copyInto(partsArray);
        return partsArray;
    }
    public void addPart(MessagePart part) {
        parts.addElement(part);
    }
}
```

-Código MMSReceive

```
package Vigilancia.mms;

import java.io.IOException;
import javax.microedition.io.*;
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;
import javax.wireless.messaging.*;

public class MMSReceive extends MIDlet implements CommandListener, Runnable,
MessageListener {
    private static final Command CMD_EXIT = new Command("Exit", Command.EXIT, 2);
    private Form content;
    private Display display;
    private Thread thread;
    private String[] connections;
    private boolean done;
    private String appID;
    private MessageConnection mmsconn;
    private Message msg;
    private String senderAddress;
    private Alert sendingMessageAlert;
    private Displayable resumeScreen;
    private String subject;
    private String contents;

    public MMSReceive() {
```

```
    appID = getAppProperty("MMS-ApplicationID");
    display = Display.getDisplay(this);
    content = new Form("MMS Receive");
    content.addCommand(CMD_EXIT);
    content.setCommandListener(this);
    content.append("Receiving...");
    sendingMessageAlert = new Alert("MMS", null, null, AlertType.INFO);
    sendingMessageAlert.setTimeout(5000);
    sendingMessageAlert.setCommandListener(this);
    resumeScreen = content;
}

public void startApp() {
    String mmsConnection = "mms://:" + appID;
    if (mmsconn == null) {
        try {
            mmsconn = (MessageConnection)Connector.open(mmsConnection);
            mmsconn.setMessageListener(this);
        } catch (IOException ioe) {
            ioe.printStackTrace();
        }
    }
    connections = PushRegistry.listConnections(true);
    if ((connections == null) || (connections.length == 0)) {
        content.deleteAll();
        content.append("" + "Esperando Mensaje Multimedia" + "...");
    }
    done = false;
    thread = new Thread(this);
    thread.start();
    display.setCurrent(resumeScreen);
}
public void notifyIncomingMessage(MessageConnection conn) {
    if ((thread == null) && !done) {
        thread = new Thread(this);
        thread.start();
    }
}
 public void run() {
    try {
        msg = mmsconn.receive();
        if (msg != null) {
            senderAddress = msg.getAddress();
            content.deleteAll();
            String titleStr = senderAddress.substring(6);
            int colonPos = titleStr.indexOf(":");
            if (colonPos != -1) {
                titleStr = titleStr.substring(0, colonPos);
            }
            content.setTitle("Enviado por: " + titleStr);
            if (msg instanceof MultipartMessage) {
                MultipartMessage mpm = (MultipartMessage)msg;
                StringBuffer buff = new StringBuffer("\nAsunto: ");
                buff.append((subject = mpm.getSubject()));
```

```
                buff.append("\nFecha: ");
                buff.append(mpm.getTimestamp().toString());
                StringItem messageItem = new StringItem("Mensaje", buff.toString());
                messageItem.setLayout(Item.LAYOUT_NEWLINE_AFTER);
                content.append(messageItem);
                MessagePart[] parts = mpm.getMessageParts();
                if (parts != null) {
                   for (int i = 0; i < parts.length; i++) {
                      buff = new StringBuffer();
                      MessagePart mp = parts[i];
                      String contentLocation = mp.getContentLocation();
                      byte[] ba = mp.getContent();
                      try {
                         Image image = Image.createImage(ba, 0, ba.length);
                         content.append(buff.toString());
                         ImageItem imageItem =
                            new ImageItem(contentLocation, image,
                               Item.LAYOUT_NEWLINE_AFTER, contentLocation);
                         content.append(imageItem);
                      } catch (IllegalArgumentException iae) {
                         buff.append(new String(ba));
                         StringItem stringItem = new StringItem("Contenido:", buff.toString());
                         stringItem.setLayout(Item.LAYOUT_NEWLINE_AFTER);
                         content.append(stringItem);
                      }
                   }
                }
             }
             display.setCurrent(content);
          }
       } catch (IOException e) {
          e.printStackTrace();
       }
    }
     public void pauseApp() {
       done = true;
       thread = null;
       resumeScreen = display.getCurrent();
    }
     public void destroyApp(boolean unconditional) {
       done = true;
       thread = null;
       if (mmsconn != null) {
          try {
             mmsconn.close();
          } catch (IOException e) {
          }
       }
    }
     public void commandAction(Command c, Displayable s) {
       try {
          if ((c == CMD_EXIT) || (c == Alert.DISMISS_COMMAND)) {
             destroyApp(false);
             notifyDestroyed();
```

```
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

Adicionalmente se debe crear una carpeta fotos dentro del paquete Vigilancia.mms, esta carpeta debe contener las imágenes: duke1.png; diablo.jpg; Gas.png; Ladron.png; fuego.png

ANEXO B
CÓDIGO PARA LA CAPTURA DE FOTO CON LA CÁMARA DE UN CELULAR

```
import java.io.IOException;
import javax.microedition.lcdui.*;
import javax.microedition.media.*;
import javax.microedition.midlet.MIDlet;
import javax.microedition.media.control.VideoControl;

public class FotoControl extends MIDlet implements CommandListener{
   private Display display;
   private Form form;
   private Command exit, back, capture, camera;
   private Player player;
   private VideoControl videoControl;
   private Video video;
   public FotoControl() {
                display = Display.getDisplay(this);
                form = new Form("Capturar Foto");
      exit = new Command("Exit", Command.EXIT, 0);
      camera = new Command("Cámara", Command.SCREEN, 1);
      back = new Command("Atrás", Command.BACK, 2);
      capture = new Command("Capturar", Command.SCREEN, 3);
      form.addCommand(camera);
      form.setCommandListener(this);
   }

   public void startApp() {
      display.setCurrent(form);
   }

   public void pauseApp() {}

   public void destroyApp(boolean unconditional){
                notifyDestroyed();
        }

   public void commandAction(Command c, Displayable s){
                String label = c.getLabel();
      if (label.equals("Exit")){
         destroyApp(true);
      } else if (label.equals("Cámara")) {
         showCamera();
      } else if (label.equals("Atrás"))
         display.setCurrent(form);
      else if (label.equals("Capturar")) {
         video = new Video(this);
         video.start();
      }
   }

   public void showCamera(){
      try{
         player = Manager.createPlayer("capture://video");
         player.realize();
         videoControl = (VideoControl)player.getControl("VideoControl");
         Canvas canvas = new VideoCanvas(this, videoControl);
```

```
            canvas.addCommand(back);
            canvas.addCommand(capture);
            canvas.setCommandListener(this);
            display.setCurrent(canvas);
            player.start();
        } catch (IOException ioe) {} catch (MediaException me) {}
    }
    class Video extends Thread {
        FotoControl midlet;
        public Video(FotoControl midlet) {
            this.midlet = midlet;
        }
        public void run() {
            captureVideo();
        }
        public void captureVideo() {
            try {
                byte[] photo = videoControl.getSnapshot(null);
                Image image = Image.createImage(photo, 0, photo.length);
                form.append(image);
                display.setCurrent(form);
                player.close();
                player = null;
                videoControl = null;
            } catch (MediaException me) { }
        }
    };
}
class VideoCanvas extends Canvas {
    private FotoControl midlet;

    public VideoCanvas(FotoControl midlet, VideoControl videoControl) {
        int width = getWidth();
        int height = getHeight();
        this.midlet = midlet;

        videoControl.initDisplayMode(VideoControl.USE_DIRECT_VIDEO, this);
        try {
            videoControl.setDisplayLocation(2, 2);
            videoControl.setDisplaySize(width - 4, height - 4);
        } catch (MediaException me) {}
        videoControl.setVisible(true);
    }

    public void paint(Graphics g) {
        int width = getWidth();
        int height = getHeight();

        g.setColor(255, 0, 0);
        g.drawRect(0, 0, width - 1, height - 1);
        g.drawRect(1, 1, width - 3, height - 3);
    }
}
```

BIBLIOGRAFÍA

[1] Jonathank Knudsen & Sing Li, Beginning j2Me - From Novice To Professional - Third Edition, 2005.

[2] Sergio Gálvez Rojas & Lucas Ortega D., Java a tope: J2ME (Java 2 Micro Edition) ,2003, Universidad de Málaga.

[3] Manuel J. Prieto, Curso de J2ME (Java 2 Microedition), 2003

[4] Mgs. Ing. Eleazar Sal y Rosas Celi, Java y Herramientas de Desarrollo para aplicativos Móviles

[5] Mgs. Ing. Eleazar Sal y Rosas Celi, Diseño de aplicaciones utilizando J2ME

[6] C. Enrique Ortiz, Article The Wireless Messaging API 2.0,October 2005 http://developers.sun.com/mobility/midp/articles/wma2/

[7] C. Enrique Ortiz, Article The Wireless Messaging API, December 2002 http://developers.sun.com/mobility/midp/articles/wma/

[8] Mobile Media API (JSR-135), http://java.sun.com/javame/reference/apis/jsr135/overview-summary.html

[9] JSR 135: Mobile Media API, http://jcp.org/en/jsr/detail?id=135

[10] Desarrollo de aplicaciones para dispositivos móviles con Java Micro Edition (JME),ftp://jano.unicauca.edu.co/cursos/Electiva_ApliMovil/presentaciones/II-2007/06-JME-Iparte.pdf

[11] Tema 6: Programación de dispositivos móviles, http://wwwdi.ujaen.es/asignaturas/progav-tema6.pdf

[12] Arquitectura y Estructura de Dispositivos y Redes http://www.construible.es/noticiasDetalle.aspx?c=50&idm=61

[13] Foro de electrónica: Uso de un celular GSM con PIC o PC http://www.forosdeelectronica.com/f24/uso-celular-gsm-pic-pc-2730/index25.html

http://www.developershome.com/sms/

[17] Dispositivos Lógicos Microprogramables: Microcontroladores

http://perso.wanadoo.es/pictob/microcr.htm

[18] Dispositivos Lógicos Microprogramables: La comunicación serie

http://perso.wanadoo.es/pictob/comserie.htm#la_comunicacion_serie

[19] Comandos AT, modo de uso

http://www.wikilearning.com/curso_gratis/que_son_y_como_funcionan_los_modems/3477-18