ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

IMPLEMENTACIÓN DE UN PROTOCOLO DE COMUNICACIÓN PARA EL CONTROL DE LOS MOVIMIENTOS DE UN BRAZO ROBOT A TRAVÉS DE LA INTERFAZ BLUETOOTH DE UN TELÉFONO CELULAR

PROYECTO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y REDES DE INFORMACIÓN

ROLDÁN MOLINA ELSA JANETH
elsyjmr24@hotmail.com

ALMEIDA PAZMIÑO MARCO ANTONIO
malmeidap@hotmail.com

DIRECTORA: MSc. Soraya Sinche
soraya.sinche@epn.edu.ec

Quito, Marzo 2011

DECLARACIÓN

Nosotros, Roldán Molina Elsa Janeth y Almeida Pazmiño Marco Antonio, declaramos bajo juramento que el trabajo aquí descrito es de nuestra autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que hemos consultado las referencias bibliográficas que se incluyen en este documento.

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

Roldán Molina Elsa Janeth

Almeida Pazmiño Marco Antonio

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por Roldán Molina Elsa Janeth y Almeida Pazmiño Marco Antonio, bajo mi supervisión.

MSc. Soraya Sinche
DIRECTORA DE PROYECTO

AGRADECIMIENTOS

A la Escuela Politécnica Nacional, Ingeniería en Electrónica y Redes de Información, benemérita Institución que nos educó.

A la MSc. Soraya Sinche, por su valiosa guía para la realización del presente Proyecto de Titilación.

A nuestros profesores que a través de sus sabias enseñanzas supieron guiarnos por el camino del bien, haciendo de nosotros personas útiles a la sociedad y al País.

A nuestros compañeros y amigos que colaboraron de una u otra manera para que se cristalice este sueño.

DEDICATORIA

A Dios que ha sido mi guía y mi apoyo a lo largo de toda mi vida. Él me ha dado la fuerza para seguir adelante.

A mis padres Milton y Elsa, por su amor, su confianza, su guía, su abnegación, y su apoyo incondicional para alcanzar esta meta.

A mis abuelitos Isabel y Pascual, Nidia y Jorge, a mi tía Gloria, a toda mi familia que me alentó en todo momento.

A mis hermanas Liliana, Andrea y Gabriela que me acompañaron, apoyaron y motivaron siempre.

A mi gran amigo Marco con quien hemos compartido todos estos años de una bonita y verdadera amistad y que gracias al apoyo y esfuerzo mutuo pudimos alcanzar esta meta.

Elsa Janeth Roldán Molina

DEDICATORIA

A Dios por enseñarme a ser valiente y esforzado.

A mi familia, quienes me prestaron sus hombros para apoyarme en los momentos difíciles.

A todos los amigos que estuvieron siempre pendientes de este proyecto.

A Janeth que convirtió a este proyecto en una experiencia inolvidable.

Marco Antonio Almeida Pazmiño

ÍNDICE GENERAL

CAPÍTULO 1

CAPÍTULO 2

CAPÍTULO 4

CAPÍTULO 5

ANEXOS

ÍNDICE DE FIGURAS

CAPÍTULO 1

CAPÍTULO 2

CAPÍTULO 3

CAPÍTULO 4

ÍNDICE DE TABLAS

CAPÍTULO 1

CAPÍTULO 2

CAPÍTULO 3

CAPÍTULO 4

ÍNDICE DE ECUACIONES

CAPÍTULO 1

CAPÍTULO 4

RESUMEN

El presente proyecto de titulación trata del diseño e implementación de un protocolo de comunicación que permita el control de los movimientos de un brazo robot utilizando el interfaz *Bluetooth* de un teléfono celular.

En el capítulo 1, se revisa la temática que envuelve el diseño de protocolos así como la herramienta *SPIN* que permite la validación de modelos para sistemas distribuidos y que fue usada en el presente proyecto para la validación y simulación del protocolo de comunicación.

En el capítulo 2, se presentan las principales características de la tecnología *Bluetooth,* y se revisa la interfaz *Bluetooth* que se encuentra incorporada en el brazo robot *Lego Mindstorms.*

En el capítulo 3, se revisa los principales *APIs* de la plataforma *J2ME* y el sistema operativo *leJOS* utilizados para el desarrollo de la aplicación para los teléfonos celulares y el brazo robot respectivamente.

En el capítulo 4, se describe el desarrollo del sistema, y pruebas realizadas; además se presenta el costo referencial del mismo.

El capítulo 5, contiene las conclusiones y recomendaciones del presente proyecto.

Finalmente, se incluyen los anexos donde se presenta el código fuente de los programas realizados para el sistema, el manual de usuario e información adicional referente el proyecto.

PRESENTACIÓN

El creciente desarrollo de aplicaciones para la transmisión de información utilizando sistemas embebidos (*PDAs*, *laptops*, teléfonos celulares, etc.) que contienen tecnologías inalámbricas que actúan sobre la misma banda de frecuencia, ha provocado que en diversas ocasiones estos sistemas se interfieran entre sí.

Las tecnologías inalámbricas *IEEE* 802.11 y *Bluetooth* que operan en las bandas de frecuencia de 2,45 GHz y 2,4 GHz respectivamente, se incluyen por lo general en la mayoría de estos sistemas embebidos de uso diario, provocando con esto, que estas tecnologías operen en el mismo ambiente y espacio de frecuencia.

La degradación del rendimiento en la transmisión de información debido a la interferencia, es un problema que el ingeniero en Electrónica y Redes de Información debe solucionar mediante el diseño de protocolos de comunicación eficientes, que permitan a los sistemas involucrados en la transmisión recuperarse de esta clase de eventos inesperados.

Por tal razón, el presente proyecto describe cómo se debe desarrollar un conjunto de reglas y procedimientos consistentes, para que un sistema embebido se desenvuelva sin problemas dentro de ambientes que presenten interferencia.

CAPÍTULO 1: CONCEPTOS PARA EL DISEÑO DE PROTOCOLOS

1.1 CONCEPTO Y ESTRUCTURA DE UN PROTOCOLO DE COMUNICACIÓN [1] [2] [3] [4] [5] [12][13]

Un protocolo de comunicación es un conjunto de reglas, formatos y procedimientos que gobiernan la interacción de procesos[1] concurrentes[2] que se ejecutan en varios equipos de comunicación de un sistema distribuido[3].

Los equipos de comunicación que conforman un sistema distribuido pueden ser distintos, y también pueden estar conectados entre sí por diferentes tipos de redes como se muestra en la Figura 1.1.

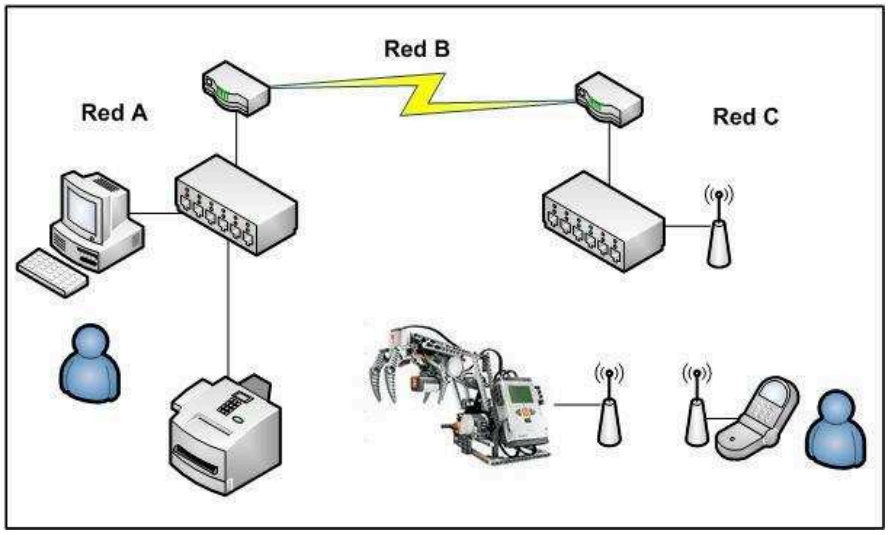


Figura 1.1 Diferentes equipos de comunicación conectados entre sí para conformar un sistema distribuido

[1] Proceso: Es un programa en ejecución o preparado para la ejecución.

[2] Concurrencia: Se refiere al procesamiento simultáneo real o aparente.

[3] Sistema distribuido: Es una colección de microprocesadores independientes y posiblemente heterogéneos conectados entre sí por medio de una red de interconexión.

Debido a dichas características, los objetivos para el diseño de un protocolo de comunicación son los siguientes:

- Establecer acuerdos para el uso de recursos compartidos entre los equipos de comunicación.

- Formalizar la interacción entre los equipos de comunicación estandarizando el uso del canal de comunicación.

Para cumplir con estos objetivos, el protocolo de comunicación debe ser especificado tomando en cuenta los siguientes aspectos:

- Los servicios que va a ofrecer el protocolo.
- La hipótesis acerca del medio donde se ejecutará el protocolo.
- Los tipos de mensajes usados para implementar el protocolo (semántica).
- El formato de cada mensaje (sintaxis).
- Las reglas y procedimientos para garantizar la consistencia en el intercambio de datos (gramática).

Debido a que un protocolo de comunicación considera aspectos como la semántica, sintaxis y gramática para el intercambio de información, se puede decir que su definición es similar a la de un lenguaje.

1.1.1. ESPECIFICACIÓN DE UN PROTOCOLO DE COMUNICACIÓN

1.1.1.1. Servicios de un protocolo de comunicación

Existen dos tipos de servicios genéricos usados en la transmisión de información que son:

- **Servicios orientados a conexión:** “este servicio se concibió con base en el sistema telefónico. El usuario para usar este tipo de servicio primero debe establecer una conexión, usarla, y luego liberarla. El aspecto esencial de una conexión es que funciona como un tubo: el emisor empuja objetos (bits) en un extremo y el receptor los toma en el otro extremo”. [5]

- **Servicios no orientados a conexión:** “este servicio se concibió con base en el sistema postal. Cada mensaje lleva completa la dirección de destino y cada uno se enruta a través del sistema, independientemente de los demás”. [5]

Otros tipos de servicios que un protocolo de comunicación puede brindar son:

- **Preservación de la secuencia:** esta característica permite al protocolo de comunicación entregar los mensajes en el mismo orden en que fueron enviados.

- **Sincronización de las unidades de datos:** este servicio envía al receptor cada mensaje creado por el usuario como una unidad. A este servicio también se lo conoce como servicio orientado a bloques o mensajes.

  Una alternativa común es el servicio orientado a *streams*, el mismo que trabaja de manera similar al servicio orientado a bloques, pero con la particular característica de que los mensajes pueden ser de distintos tamaños.

- **Servicio para el establecimiento y liberación de la conexión:** este servicio permite configurar un canal físico o lógico para permitir que el transmisor y el receptor lleguen a un mútuo acuerdo respecto a ciertos parámetros que serán enviados con los mensajes.

Estos parámetros también intervienen cuando la comunicación entre los dos equipos se interrumpe y se libera el canal. A esta fase se le conoce como liberación de la conexión.

- **Detección de errores:** este servicio permite conocer al receptor si un mensaje llegó con error.

  Uno de los tipos de error más críticos es el error de ráfaga. El error de ráfaga se presenta cuando un grupo de bits de un mensaje se han alterado. Este tipo de error es muy frecuente en las transmisiones de tipo serial.

  "La detección de errores usa el concepto de redundancia, que significa añadir bits extras al mensaje original para detectar en el destino los errores". [4]

  La técnica más habitual para la detección de errores de ráfaga es la comprobación *CRC* (*Cyclic Redundancy Check*).

  La Figura 1.2 muestra el proceso para la obtención de la *FCS* (*Frame Check Sequence*), que corresponde a la información redundante generada por la comprobación *CRC*. La *FCS* será transmitida junto al mensaje original para permitir al receptor la detección de errores.

  La Figura 1.3 muestra el proceso que sigue tanto el transmisor como el receptor para la detección de errores en los mensajes generados por el usuario.

  Para la correcta elección del polinomio divisor se deben tomar en cuenta tres factores importantes que son: El orden del polinomio *CRC*, el tamaño del mensaje y la distancia de *Hamming*[4].

[4] Distancia de *Hamming*: Se define como el número de errores, de todos los posibles que puede contener un mensaje, que son indetectables por un polinomio *CRC*.

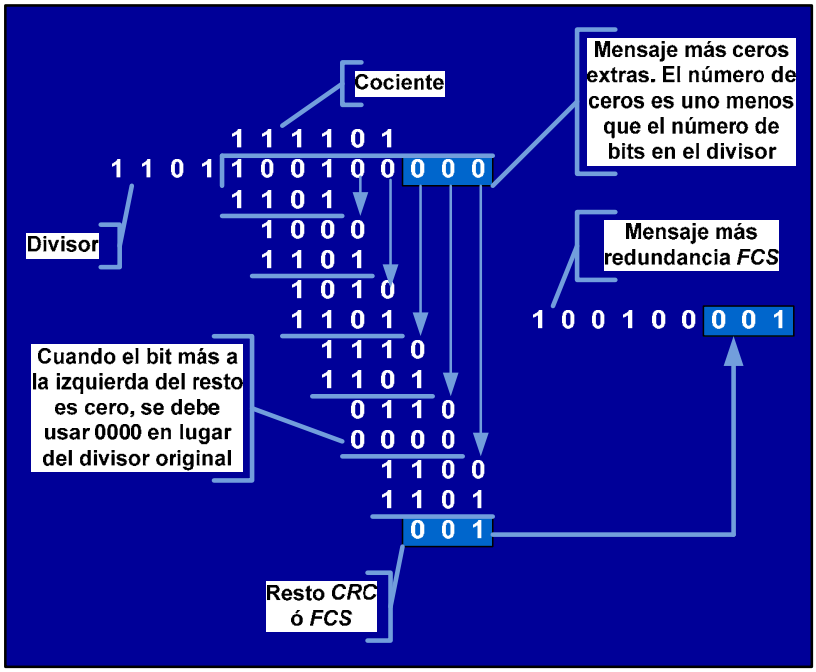


Figura 1.2 Ejemplo de cálculo de la *FCS* a partir de un polinomio *CRC* de orden 3 [4]

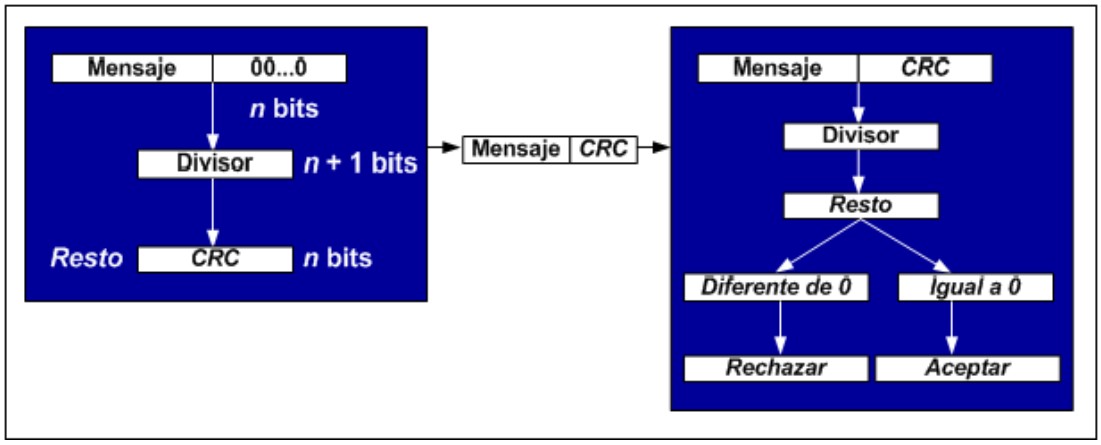


Figura 1.3 Proceso que sigue el transmisor y el receptor para la detección de errores [4]

- **Control de flujo:** en este servicio se asegura que los mensajes no sean entregados más rápido de lo que el equipo receptor pueda procesarlos.

Existen dos métodos para controlar el flujo de datos: *stop and wait* y ventana deslizante.

En el método *stop and wait* que se muestra en la Figura 1.4, el emisor envía un mensaje y espera un acuse de recibo positivo denominado *ACK* (*Acknowledgment*) antes de enviar el siguiente mensaje.

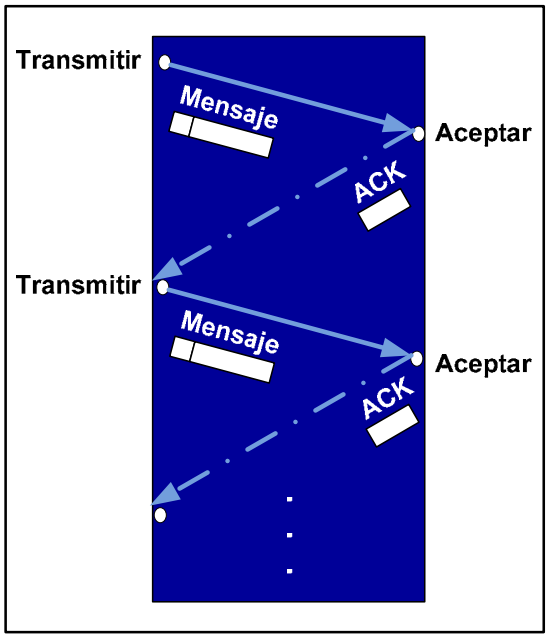


Figura 1.4 Método *stop and wait* [1]

El problema de saturación del receptor desaparece con este método, sin embargo la pérdida de un *ACK* o de un mensaje provocaría el bloqueo total del sistema como se muestra en la Figura 1.5, por tal razón se emplea un temporizador para limitar el tiempo que el transmisor debe esperar por un acuse de recibo.

“En el método de control de flujo de ventana deslizante, se puede transmitir varios mensajes antes de necesitar un acuse de recibo. Los mensajes se pueden enviar uno detrás de otro, lo que significa que el enlace puede

transportar varios mensajes de una vez y que su capacidad se puede usar de forma más eficiente. El receptor envía un *ACK* para algunas de las tramas para confirmar la recepción de múltiples mensajes". [4]

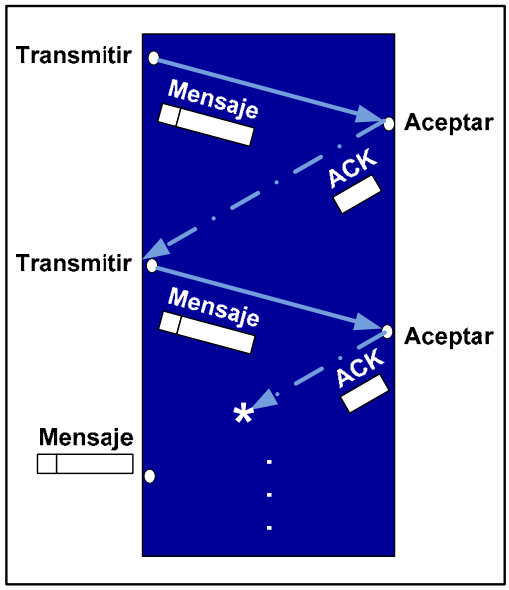


Figura 1.5 Pérdida de un *ACK* en el método *stop and wait* [1]

Las Figuras 1.6 y 1.7 muestran cómo funciona el método de ventana deslizante tanto para el transmisor como para el receptor.

- **Control de errores:** este servicio permite entregar los mensajes al equipo receptor tal y como fueron enviados por el equipo transmisor.

  Generalmente este servicio implementa la detección de errores para la retransmisión de los mensajes.

  Existen dos métodos que se utilizan para la corrección de errores: *FEC* (*Forward Error Correction*) y *BEC* (*Backward Error Correction*).

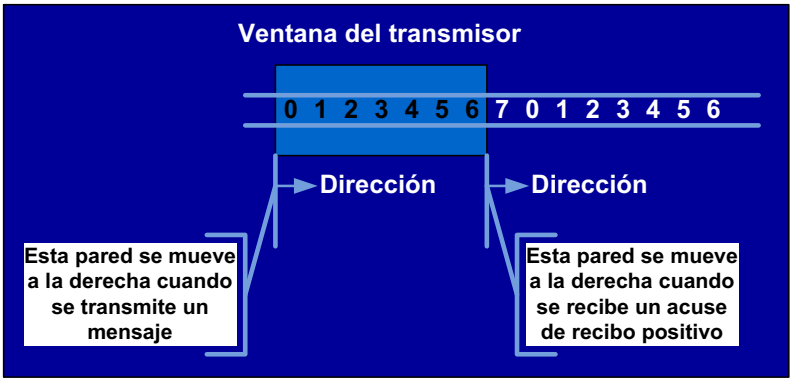


Figura 1.6 Ventana deslizante del transmisor [4]

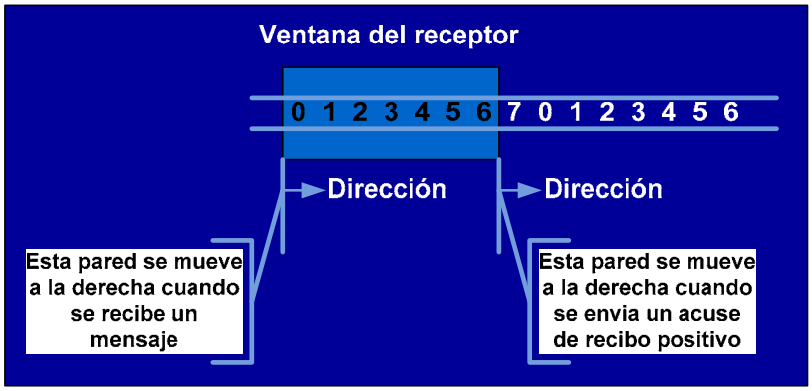


Figura 1.7 Ventana deslizante del receptor [4]

El método *FEC* permite a un equipo receptor la detección y corrección de errores en los mensajes sin necesidad de solicitar la retransmisión de esos mensajes al equipo transmisor. Se utilizan los códigos de *Hamming*[5] para lograr este objetivo.

Por otra parte, el método *BEC* permite al equipo receptor la detección de errores en los mensajes, y el uso de acuses de recibo negativos *NACK* (*Negative Acknowledgment*) para solicitar la retransmisión de esos mensajes al

[5] Código de *Hamming*: Técnica que permite a un equipo transmisor añadir o intercalar bits de redundancia a los mensajes de datos que se van a transmitir, para permitir al equipo receptor la detección y corrección de errores de dichos mensajes en caso de ser necesario.

equipo transmisor. La técnica *ARQ* (*Automatic Repeat Request*) se basa en el método *BEC*.

El método *stop and wait* es un tipo de *ARQ* como se muestra en la Figura 1.8 y su funcionamiento se describe a continuación:

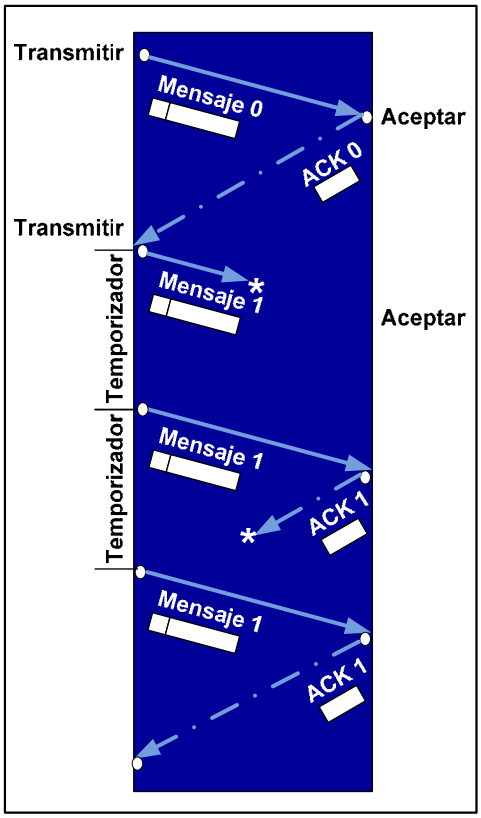


Figura 1.8 Método de control de flujo *stop and wait* con control de errores [1]

- Cuando el equipo transmisor envía un mensaje hacia el equipo receptor, este deberá mantener una copia de ese mensaje hasta que se reciba un *ACK* o un *NACK.* Si se recibe un *ACK* el transmisor podrá enviar el próximo mensaje, caso contrario retransmite el mensaje almacenado.

- El problema del bloqueo del sistema del método *stop and wait* se resuelve mediante la implementación de un temporizador en el transmisor.

  Este temporizador tiene como finalidad el establecer un intervalo de tiempo, durante el cual, debe llegar el acuse de recibo del mensaje previamente enviado. Si no llegó dicho acuse de recibo durante ese lapso de tiempo, el transmisor tendrá que retransmitir el mensaje.

  El valor del intervalo de tiempo se puede deducir como sigue:

  - Si el equipo transmisor envía un mensaje denominado *Mensaje 0* hacia el equipo receptor, entonces este mensaje se demorará un tiempo de propagación *tp* para llegar. Una vez recibido el *Mensaje 0*, el equipo receptor se toma un tiempo de procesamiento *tc* para verificar si el mensaje contiene errores.

  - Si el mensaje no contiene errores, entonces enviará un *ACK* 0 hacia el equipo transmisor, el mismo que se demorará en viajar un tiempo de propagación igual que el del *Mensaje 0*.

  - Además se debe considerar el tiempo de transmisión del *Mensaje 0* (*tx*) y el tiempo de transmisión del *ACK 0 (txAck)*.

  Entonces el tiempo mínimo que hay que esperar antes de retransmitir un mensaje deberá ser mayor que el tiempo que tardaría en recibirse el *ACK* del mensaje que inicia el temporizador. El valor del temporizador se expresa utilizando la Ecuación 1.1:

$$Temporizador > 2*tp + tc + tx + txAck$$

Ecuación 1.1 Valor del temporizador para *stop and wait*

1.1.1.2. Hipótesis del medio

En la hipótesis del medio se describen las características más relevantes respecto al medio de transmisión por donde viajará la información, el modo de operación del canal de comunicación, la tecnología que se utilizará para acceder al mismo, la capacidad de detección de errores de dicha tecnología, etc. También se enumeran todos los factores externos que pueden intervenir en la ejecución del protocolo.

1.1.1.3. Tipos de mensajes

Aquí se definen todos los tipos de mensajes que intervendrán en la comunicación. Se toma como ejemplo al método *stop and wait.*

Este método tiene los siguientes mensajes: *MENSAJE* que representa la información que se va a transmitir, *ACK* para los acuses de recibo positivos, y *NACK* para los acuses de recibo negativos.

Estos mensajes pueden ser expresados como un conjunto finito de la siguiente manera:

$$V = \{MENSAJE, ACK, NACK\}$$

1.1.1.4. Formato de encapsulación de los tipos de mensajes [6]

La estructura de un mensaje clásico consta de tres campos como se muestra en la Figura 1.9, y se describen a continuación:

- **Cabecera:** Es un conjunto de campos de longitud fija que contienen información de control denominada *PCI* (*Protocol Control Information*). La *PCI* por lo general contiene bits para definir la función del mensaje, el número de secuencia del mismo e información acerca del estado de los equipos y de la conexión.

- **Información:** Este campo contiene los datos de un protocolo de mayor jerarquía.

- **Cola:** Es un conjunto de campos de longitud fija que contienen información de control al final del mensaje y por lo general corresponde a un *FCS* para la detección de errores. Este campo depende de la *PCI* y de los datos presentes en el campo información.

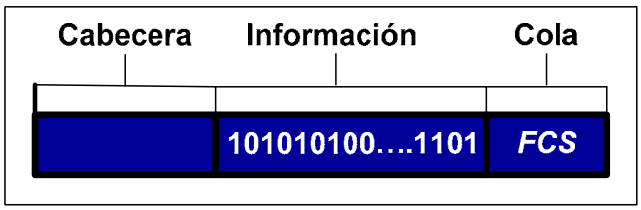


Figura 1.9 Formato de mensaje utilizando cabeceras y colas [1]

1.1.1.5. Reglas y procedimientos para el intercambio de datos

Las reglas y procedimientos para el intercambio de datos permiten al diseñador realizar una descripción parcial del protocolo. Se deberán validar todas estas reglas y procedimientos antes de continuar con su implementación.

La especificación de las reglas y procedimientos para el protocolo de comunicación a diseñarse se encuentra en el ítem ***4.1.2.3.1. Planeamiento e implementación del modelo del protocolo de comunicación*** del Capítulo 4.

1.1.2. MODELOS Y ARQUITECTURAS PARA EL DISEÑO DE PROTOCOLOS DE COMUNICACIÓN

Los modelos para el diseño de protocolos de comunicación proporcionan un marco teórico y tecnológico para diseñar, implementar y administrar dichos protocolos.

Típicamente se basan en una estructura por capas, lo que permite dividir las distintas tareas que realizará el protocolo de comunicación en módulos. Cada módulo tendrá la capacidad de realizar una sub tarea y de interactuar con otros módulos.

1.1.2.1. El Modelo de referencia *OSI* [6] [7] [8]

"Creado en 1947, la Organización Internacional de Estandarización (*ISO, International Standards Organization*) es un organismo multinacional dedicado a establecer acuerdos mundiales sobre estándares internacionales. Un estándar *ISO* que cubre todos los aspectos de la redes de comunicación es el modelo de Interconexión de Sistemas Abiertos (*OSI, Open System Interconnection*). Un sistema abierto es un modelo que permite que dos sistemas diferentes se puedan comunicar independientemente de la arquitectura subyacente. Los protocolos específicos de cada vendedor no permiten la comunicación entre dispositivos no relacionados. El objetivo del modelo *OSI* es permitir la comunicación entre sistemas distintos sin que sea necesario cambiar la lógica del *hardware* o el *software* subyacente" [4].

Este modelo divide a todo el proceso de comunicación en varias funciones, las mismas que se encuentran distribuidas en siete capas como se observa en la Figura 1.10.

En cada capa, un proceso que se encuentra en una máquina se comunica con su proceso par en otra máquina como se muestra en la Figura 1.11. Según la terminología del modelo *OSI*, los procesos que se ejecutan en una capa *n* se les denominan entidades de capa *n*. El intercambio de información entre entidades se realiza utilizando *PDU*s (*Protocol Data Units*). Cada *PDU* está compuesta por una cabecera que contiene la información de control y por la información de usuario ó *SDU* (*Service Data Unit*). El comportamiento de las entidades de capa *n* es administrado por un protocolo de capa *n*.

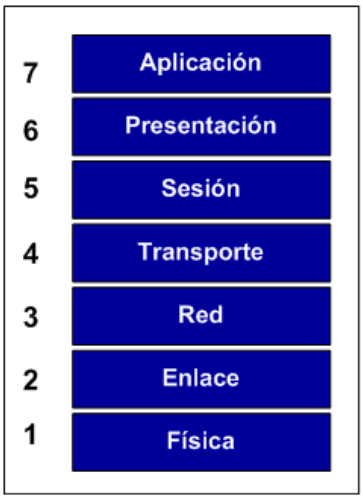


Figura 1.10 El modelo de referencia *OSI* [5]

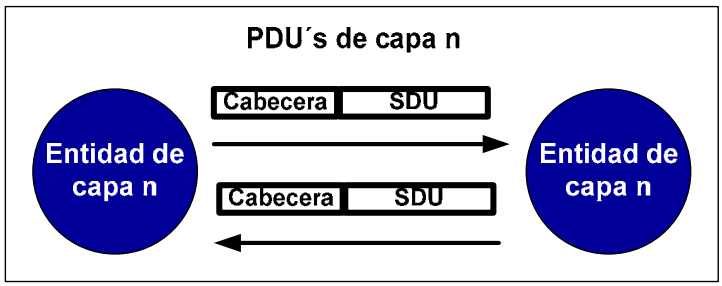


Figura 1.11 Comunicación entre dos entidades pares de capa *n* [6]

La comunicación entre procesos pares es virtual en el sentido de que no existe un enlace físico entre ellos. Por ejemplo, para que la comunicación tenga lugar, la entidad de la capa *n* + *1* hace uso de los servicios ofrecidos por la capa *n*. La forma en que una capa solicita un servicio a otra capa es a través de primitivas como se muestra en la Figura 1.12.

Una primitiva especifica una operación o una acción que va a ocurrir. Puede ser una solicitud de un determinado servicio, o una indicación de que una determinada acción o evento, ha sucedido. Existen cuatro primitivas que son [46]:

- ***Request***: “una entidad desea que el servicio realice un trabajo.

- ***Indication*:** una entidad es informada acerca de un evento.
- ***Response*:** una entidad desea responder a un evento.
- ***Confirm*:** una entidad va a ser informado acerca de su solicitud".

Para que una capa *n + 1* solicite un servicio, deberá enviar una primitiva del tipo *request* hacia la capa *n*. La capa *n* entonces, le responderá a la capa *n + 1* utilizando la primitiva del tipo *confirm*.

La Figura 1.12 (a) muestra como una entidad de capa *n + 1* solicita un servicio a otra entidad de capa *n*, mientras que la Figura 1.12 (b) muestra como una entidad de capa *n + 1* solicita un servicio a otra entidad de capa *n + 1*.

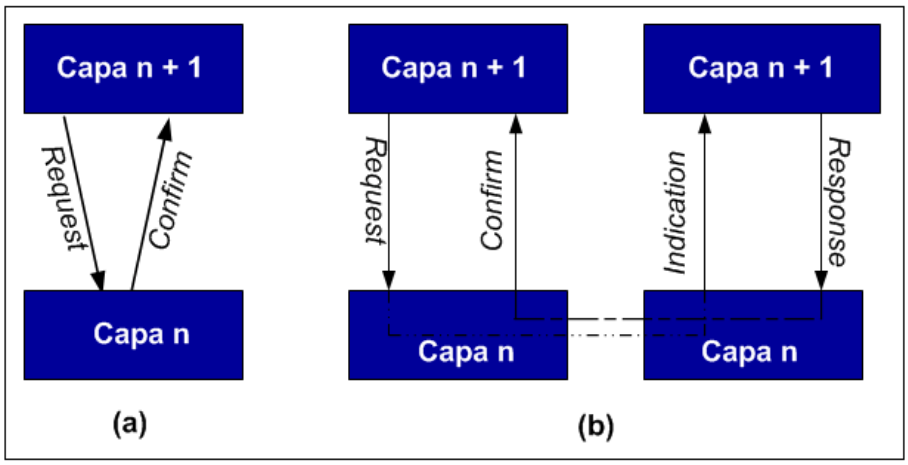


Figura 1.12 Comunicación entre capas usando primitivas [6]

La transmisión de la *PDU* de la capa *n + 1* se realiza a través de un punto que pertenece a la capa *n* denominado *SAP* (*Service Access Point*) y se muestra a en la Figura 1.13.

La entidad de la capa *n* añade su cabecera a la *PDU* de la capa *n + 1* para formar la *PDU* de la capa *n*.

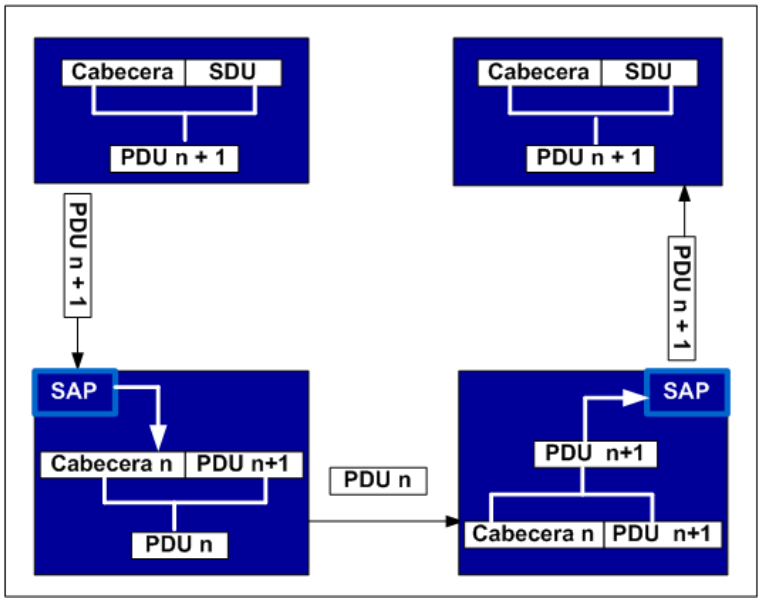


Figura 1.13 Comunicación entre entidades pares [6]

Tras la recepción de la *PDU* de la capa *n*, el proceso par usa la cabecera para ejecutar el protocolo de la capa *n* y, si es apropiado, entregar la *SDU* a la entidad de la capa *n + 1* correspondiente.

A continuación se describen de manera breve las diferentes tareas que realizan cada una de las capas del Modelo de Referencia *OSI* [47]:

- **Capa física:** "Se encarga de la transmisión de cadenas de bits no estructurados sobre el medio físico; está relacionada con las características mecánicas, eléctricas, funcionales y de procedimiento para acceder al medio físico.

- **Capa de enlace de datos:** Proporciona un servicio de transferencia de datos confiable a través del enlace físico; envía bloques de datos (tramas) llevando a cabo la sincronización, el control de errores y el flujo.

- **Capa de red:** proporciona independencia a los niveles superiores respecto a las técnicas de conmutación y de transmisión utilizadas para conectar los sistemas; es responsable del establecimiento, mantenimiento y cierre de las conexiones.

- **Capa transporte:** proporciona una transferencia transparente y confiable de datos entre los puntos finales; además, proporciona procedimientos de recuperación de errores y control de flujo entre el origen y el destino.

- **Capa sesión:** proporciona el control de la comunicación entre las aplicaciones; establece, gestiona y cierra las conexiones (sesiones) entre las aplicaciones operadoras.

- **Capa presentación:** proporciona a los procesos de la aplicación independencia respecto a las diferencias en la representación de datos.

- **Capa aplicación:** proporciona el acceso al entorno *OSI* para los usuarios y, también, proporciona servicios de información distribuida".

1.1.2.2. La arquitectura *TCP*/*IP*

La arquitectura *TCP*/*IP* (*Transmission Control Protocol, Internet Protocol*) no se originó en base al modelo *OSI,* por tal razón no existe un acuerdo universal para describir las funciones de las capas que conforman esta arquitectura en base a dicho modelo.

TCP/*IP* omite algunas características que se encuentran bajo el modelo *OSI* y combina las funciones de algunas capas.

Esta arquitectura está conformada por cuatro capas. Cuando se envían datos, cada capa trata a la información que recibe de la capa superior como datos, añade su

cabecera y después la pasa a la capa inferior. Cuando se reciben datos, el procedimiento inverso se lleva a cabo.

Las cuatro capas que conforman al modelo *TCP*/*IP*, y sus respectivas funciones son las siguientes:

- **Capa Aplicación:** esta capa agrupa las funciones de las capas aplicación, presentación y sesión correspondientes al modelo *OSI.* En esta capa se utilizan *sockets* y puertos para permitir la comunicación entre aplicaciones. Generalmente las aplicaciones están asociadas con uno o más puertos.

- **Capa Transporte:** en la arquitectura *TCP*/*IP*, existen dos protocolos a nivel de la capa transporte. El protocolo *TCP* que garantiza la transmisión de la información, mientras que el protocolo *UDP* (*User Datagram Protocol*) transporta datagramas de un extremo hacia el otro sin verificar la fiabilidad de los mismos. Ambos protocolos son usados para diferentes tipos de aplicaciones.

- **Capa Red:** "el protocolo internet (*IP, Internet Protocol*) se utiliza en esta capa para ofrecer el servicio de encaminamiento a través de varias redes. Este protocolo se implementa tanto en los sistemas finales como en los *routers* intermedios". [47]

- **Capa de acceso a la red:** las funciones de las capas enlace de datos y física son agrupadas en esta capa. Existe documentación donde se describe cómo *IP* puede utilizar protocolos de capa enlace de datos ya existentes como *Ethernet.*

La Figura 1.14 muestra las capas que conforman la arquitectura *TCP*/*IP*.

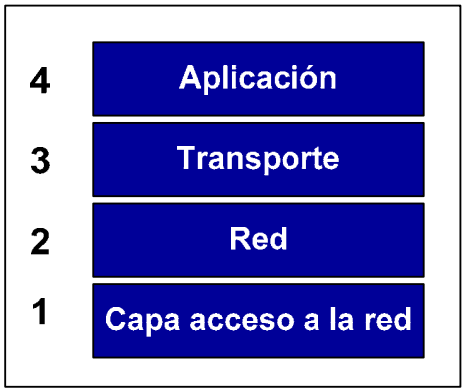


Figura 1.14 La arquitectura *TCP/IP* [5]

1.1.3. MÉTODOS FORMALES PARA LA VERIFICACIÓN DE PROTOCOLOS DE COMUNICACIÓN [3][9][10] [11] [14] [15] [16] [17] [74]

En el año de 1970, el desarrollo en el campo de la computación, y específicamente de los microprocesadores provocó que el costo de estos *chips* se reduzca de manera considerable.

La utilización masiva de estos chips en diversos sistemas, provocó una serie de investigaciones, las cuales tenían por objetivo crear métodos formales que permitan verificar estos sistemas.

Los métodos que nacieron a partir de esas investigaciones son:

- **Álgebra de procesos:** "este método utiliza un enfoque algebraico para el estudio de procesos concurrentes. Sus herramientas son lenguajes algebraicos para la especificación de los procesos y la formulación de declaraciones acerca de ellos, junto con los cálculos para la verificación de estas declaraciones". [73]

- **Validación de modelos:** este método se basa en la creación de modelos, que son generalizaciones que ignoran detalles acerca de la implementación de un

sistema (*hardware* ó *software*), con el propósito de controlar la complejidad intelectual de ese sistema.

El método de validación de modelos ofrece en la actualidad diferentes herramientas de *software,* que están generalmente basadas en lenguajes de programación conocidos.

Estas herramientas minimizan el tiempo de verificación del comportamiento de los procesos concurrentes en un sistema, y también la fácil localización de errores en el mismo.

1.1.3.1. Herramientas para la validación de modelos de sistemas distribuidos

Las herramientas para la validación de modelos de sistemas distribuidos se basan en algoritmos que utilizan la lógica temporal para su análisis.

La lógica temporal se deriva de la lógica proposicional, por lo cual contiene un conjunto de operadores temporales. Esta lógica permite estudiar el comportamiento de un sistema concurrente mediante el uso de sistemas de transición de estados[6].

Existe una gran variedad de herramientas para la validación de modelos de sistemas distribuidos, sin embargo dos de ellas son populares para este propósito:

- ***SPIN* (*Simple PROMELA Model Checker*):** herramienta utilizada para analizar sistemas asincrónicos, concurrentes y distribuidos. Es capaz de validar únicamente los modelos que se encuentran escritos bajo el lenguaje de programación *PROMELA* (*Process Meta Language*), el mismo que tiene una sintaxis similar a la del lenguaje C.

[6] Sistemas de transición de estados: Máquina abstracta que se utiliza para el estudio de sistemas de computación. La máquina está compuesta por un conjunto de estados y de las transiciones entre los mismos.

- ***SMV* (*Symbolic Model Verifier*):** herramienta utilizada para analizar sistemas sincrónicos, asincrónicos y procesos no determinísticos. Utiliza un lenguaje de programación basado en símbolos denominado *SMV*.

La Tabla 1.1 resume las características más importantes de ambas herramientas, pudiendo concluir que *SPIN* brinda muchos beneficios que permiten el desarrollo rápido y eficiente de sistemas distribuidos debido a que:

- Su interfaz gráfica combinada con la generación de contra ejemplos, permiten el análisis de *software* crítico, el mismo que se ejecuta en un ambiente donde actúan procesos concurrentes.
- Permite describir sistemas distribuidos de gran escala a través modelos, los mismos que *SPIN* utiliza para generar todos sus posibles estados facilitando así, la detección de errores en ellos.
- Simula sistemas no determínisticos, como es el caso de los sistemas de comunicaciones.

Característica	***SPIN***	***SMV***
Tipo de lógica que utiliza el algoritmo	Temporal	Temporal
Lenguaje de programación	*PROMELA*	*SMV*
Sintaxis del lenguaje	Similar a C	Simbólico
Generación de contra ejemplos	Si	No
Interfaz gráfica	Si	No
Sistemas operativos	*Win/Unix*	*Win/Unix/Mac*
Premios otorgados	*ACM System Software Award*	Ninguno
Tipos de sistemas que simula	Asincrónicos, distribuidos, no determinísticos	Asincrónicos, sincrónicos y no determinísticos

Tabla 1.1 Resumen de las principales características de las herramientas *SPIN* y *SMV* [9] [16].

Característica	*SPIN*	*SMV*
Análisis de estados de un programa	Si	No
Facilidad para la detección de errores	Si	No

Tabla 1.1 Resumen de las principales características de las herramientas *SPIN* y *SMV* [9] [16]. (Continuación)

1.1.3.2. Introducción a la herramienta *SPIN* [18]

"SPIN es una herramienta *software* que soporta el análisis y verificación de sistemas asíncronos, concurrentes y distribuidos. Para la creación del modelo del sistema se utiliza los fundamentos de la notación matemática *CSP*[7] (*Communicating Sequential Processes*) de *Hoare* y para su descripción un lenguaje de alto nivel llamado *PROMELA*. La sintaxis de este lenguaje es derivada del lenguaje C, con extensiones de los comandos de guarda[8] de *Dijkstra*.

Un sistema en *PROMELA* está compuesto por los siguientes componentes:

- Procesos.
- Canales sincrónicos[9] y asincrónicos[10].
- Variables.

La concurrencia de los procesos es asíncrona (no se toma en cuenta el tiempo de ejecución para cada proceso) y modelada mediante el intercalado de instrucciones.

[7] *CSP*: Introduce el concepto de proceso como una abstracción matemática que describe la interacción entre los sistemas y su medio ambiente.
[8] Comando de guarda: Conjunto de instrucciones que pueden bloquear la ejecución de un proceso debido al valor de sus variables o al contenido de los canales.
[9] Canal sincrónico: Canal que no almacena los mensajes en el *buffer*.
[10] Canal asincrónico: Canal que almacena los mensajes en el *buffer*.

Dado un modelo escrito en *PROMELA* como entrada, *SPIN* genera un programa en C que realiza la verificación del sistema mediante la enumeración de su espacio de estados, también conocido como estados de un programa. En caso de que el modelo presente errores, *SPIN* genera los denominados contra ejemplos, que permiten al programador saber cómo y dónde el modelo falla al momento de satisfacer una propiedad específica.” [14]

La Figura 1.15 muestra el proceso que sigue la herramienta *SPIN* para generar los reportes al usuario.

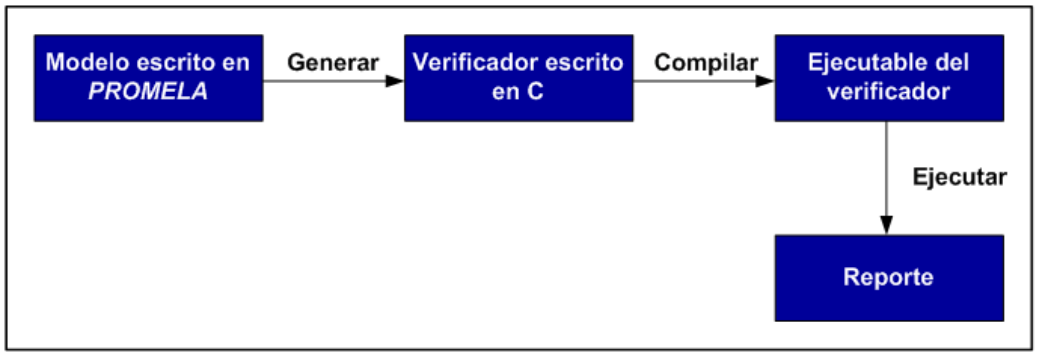


Figura 1.15 Proceso que sigue la herramienta *SPIN* para la verificación de un modelo escrito en *PROMELA* [9]

Los procesos por definición son objetos globales. Los canales y las variables representan información que puede ser global o local para un proceso.

Se pueden definir constantes y variables. Las constantes se declaran a partir de la palabra reservada *define,* mientras que las variables mediante varios tipos de datos como son: *bit*, *bool*, *byte*, *short*, *int, mtype* y *unsigned*. Se pueden crear también arreglos a partir de estas variables.

Las sentencias de control para selección y repetición son *if* y *do* respectivamente, sin embargo se puede añadir sentencias como las que se encuentran en otros lenguajes

de programación por medio de macros. Un ejemplo de esas sentencias es la sentencia *for*.

La Figura 1.16 muestra un ejemplo que utiliza varios de los recursos antes mencionados.

```
1.      mtype = {red,yellow,green};
2.      mtype light = green;
3.
3.      proctype P() {
4.        do
5.        :: if
6.              :: light == red -> light = green;
7.              :: light == yellow -> light = red;
8.              :: light == green -> light = yellow;
9.           fi;
10.             printf("The light is now %e\n",light);
11.          od
12.     }
13.
14.   init { run P()}
```

Figura 1.16 Ejemplo de un programa escrito en el lenguaje de programación *PROMELA* [9]

La instrucción *printf* algunos tipos de datos y operadores lógicos fueron heredados del lenguaje C.

La Tabla 1.2 resume los operadores del lenguaje de programación *PROMELA.*

Las ventajas de *SPIN* son las siguientes:

- Puede ser usado para describir una acción ó una secuencia de acciones.
- Cada proceso puede bloquearse hasta que se cumplan determinadas condiciones (comandos de guarda).
- Permite el uso de etiquetas para realizar saltos.

- Se puede definir macros para representar acciones.
- Rápida detección de errores de los modelos con simulación guiada.

Las desventajas de *SPIN* son:

- Si el modelo no es diseñado de una manera adecuada ó presenta demasiadas características, *SPIN* podría generar un espacio de estados demasiado grande, con lo cual aumentaría el tiempo de procesamiento. A este problema se le conoce como explosión de estados.

- La capacidad de memoria de un computador puede ser insuficiente para determinados modelos, ya que *SPIN* almacena cada estado en ella.

Operador	**Nombre**
()	paréntesis
[ ]	arreglo
.	selección
!	negación lógica
~	complemento
++,--	incremento, decremento
*,/,%	multiplicación, división y módulo
+,-	suma, resta
<< , >>	desplazamiento de bits
<, <=, >,>=	operadores relacionales aritméticos
==, !=	igualdad, desigualdad
&	operación and
^	operación or exclusivo
|	operación or inclusivo
&&	and lógico
||	or lógico
(>:)	expresión condicional
=	asignación

Tabla 1.2 Operadores lógicos del lenguaje de programación *PROMELA* [9]

1.1.3.3. Programación concurrente y distribuida en *SPIN*

Los dispositivos electrónicos de hoy pueden ejecutar varios procesos que son intercalados en tiempo por su procesador, para simular que se están realizando varias tareas simultáneamente. La velocidad del reloj de la *CPU* (C*entral Processing Unit*) de algunos de estos dispositivos están aproximadamente en el orden de magnitud de un *gigahertz*, es decir, que en cada nanosegundo, la *CPU* realiza un determinado proceso.

Estos procesos dependen también de un gestor de procesos, el mismo que decide qué proceso será el siguiente en ejecutarse en la *CPU* dependiendo de su prioridad.

A la programación en un sistema concurrente, como el antes mencionado, se la denomina programación concurrente.

Para simular el comportamiento de un sistema concurrente, *SPIN* considera los siguientes aspectos: [9]

- "Cuando un proceso ejecuta un programa, un registro denominado puntero de instrucción almacena la dirección de la siguiente instrucción que puede ser ejecutada. La dirección de esa instrucción se denomina punto de control.

- Un estado de un programa en ejecución, viene dado por los valores de las variables junto con la instrucción que corresponde al valor almacenado en el puntero de instrucción y el nombre del proceso que contiene a dicha instrucción.

- Los estados que se originan cuando se ejecuta un programa, pueden ser mostrados en una tabla. Un estado es almacenado en una fila de la tabla. La primera fila corresponde al estado inicial.

- Si se ejecuta un solo proceso P_i, una posible secuencia de estados podría ser $(S_i^0, S_i^1, S_i^2, \ldots)$, donde el estado S_i^j sigue al estado S_i^{j-1} si y solo si S_i^j se obtiene al ejecutar la instrucción del puntero de instrucción de P_i en S_i^{j-1}.

- Si se ejecutan varios procesos, una posible secuencia de estados podría ser $(S^0, S^1, S^2, \ldots)$, donde el estado S^j sigue al estado S^{j-1} si y solo si S^j se obtiene al ejecutar la instrucción del puntero de instrucción de un determinado proceso en S^{j-1}".

Para ejecutar un proceso es necesario que se defina su nombre, contenido y se cree una instancia del mismo. Todos los procesos que pueden ser instanciados deben estar definidos por la palabra reservada *proctype*, como se muestra en el programa de la Figura 1.17.

```
1.    byte   n = 0;
2.
3.    proctype P() {
4.       n = 1;
5.       printf("Process P, n = %d\n", n);
6.    }
7.
8.    proctype Q() {
9.       n = 2;
10.      printf("Process Q, n = %d\n", n);
11.   }
12.
13.   init { run P();run Q() }
```

Figura 1.17 Programa escrito en *PROMELA* que contiene dos procesos concurrentes [9]

En este programa se declaran dos procesos denominados *P* y *Q*, como también una variable global del tipo de dato *byte*, compartida por ambos procesos. Debido a que *proctype* solo declara el comportamiento de un proceso, es necesario otro tipo de proceso denominado *init* para inicializarlo.

El proceso *init* es comúnmente comparado con la función *main*() de un programa escrito en el lenguaje de programación *C*. Este proceso puede inicializar variables globales, procesos y crear canales.

El operador *run* permite instanciar un determinado número de copias de un proceso. Este operador puede estar en cualquier lugar dentro de un proceso, no solo dentro del proceso *init*.

Se debe tomar en cuenta también que las instrucciones en *PROMELA* son atómicas, es decir, las instrucciones que se almacenan en los punteros de instrucción de un conjunto de procesos son ejecutadas por completo.

Para el programa de la Figura 1.17 un estado vendría conformado por el valor de la variable *n* y los punteros de instrucción de los procesos *P* y *Q:*

(n, Puntero P, Puntero Q)

Para obtener los estados del programa de la Figura 1.17 se realiza el siguiente procedimiento:

- Se debe obtener todos los posibles estados de los procesos *P* y *Q* por separado como se muestra en la Tabla 1.3:

Procesos	**Estados**		
	1	**2**	**3**
P	(0,4,-)	(1,5,-)	(1,6,-)
Q	(0,-,9)	(2,-,10)	(2,-,11)

Tabla 1.3 Posibles estados para los procesos *P* y *Q* [9]

Por ejemplo, para el estado 1 del proceso *P* se tiene:

- 0 corresponde al valor inicial de *n.*
- 4 corresponde al número de línea de código de la Figura 1.17, y representa al puntero de instrucción de *P*.
- - indica que no existe un puntero de instrucción *Q* debido a que se están considerando los procesos *P* y *Q* por separado.

- El siguiente paso es intercalar los estados 1 de los procesos *P* y *Q* de la Tabla 1.3. Luego *SPIN* escogerá aleatoriamente el puntero de instrucción del proceso *P* o *Q* que se ejecutará. Una posible combinación se muestra en la Tabla 1.4.

Estados del programa	**Puntero de instrucción a ejecutar**	**Instrucción**	**Proceso**
(0,4,9)	4	*n = 1*	*P*
(1,5,9)	9	*n = 2*	*Q*
(2,5,10)	10	*printf(Q)*	*Q*
(2,5,11)	5	*printf(P)*	*P*
(2,6,11)	Fin del programa	Fin del programa	Fin del programa

Tabla 1.4 Estados generados después de la ejecución del programa de la Figura 1.17 [9]

Existe un *IDE* (*Integrated Development Environment,* Ambiente de Desarrollo Integrado) denominado *jSpin*, el mismo que fue escrito en el lenguaje de programación *Java* para facilitar el manejo de *SPIN*.

La Figura 1.18 muestra la salida de pantallas después de haber ejecutado el programa de la Figura 1.17 con esta herramienta.

La programación que se utiliza para interconectar los diferentes sistemas concurrentes de un sistema distribuido se denomina programación distribuida.

“Para modelar un sistema distribuido en *SPIN* se abstraen todos los detalles de la red y sus protocolos, y se modela al sistema distribuido como un conjunto de procesos concurrentes y a las redes de comunicación como canales, que son objetos sobre los cuales se puede enviar y recibir mensajes.” [9]

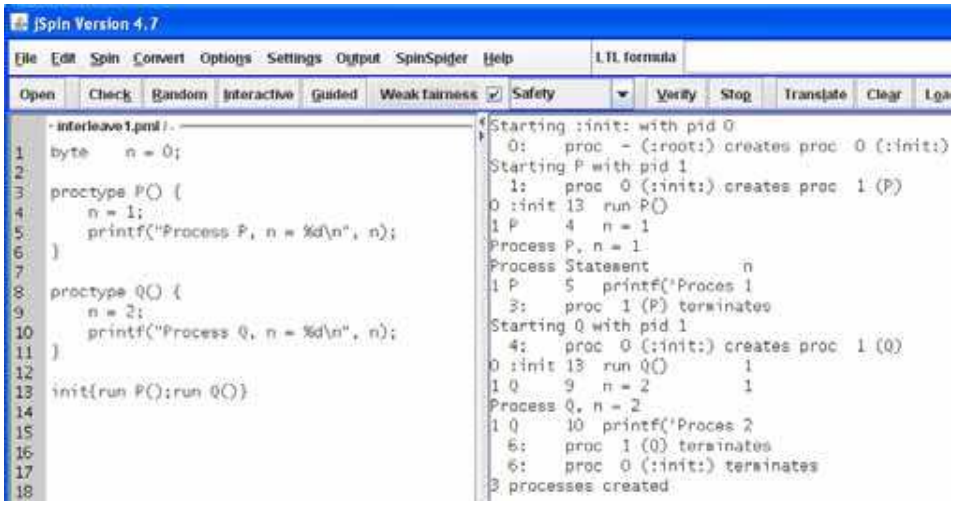


Figura 1.18 Tabla de estados generada después de la ejecución del programa de la Figura 1.17 en *jSpin*

Un canal en *PROMELA* es un tipo de dato que puede enviar y recibir mensajes, y es declarado utilizando un inicializador, el nombre del canal, su capacidad y los tipos de datos que enviará y recibirá como sigue:

chan *ch = [capacidad] of {tipo de dato,.....,tipo de dato};*

Existe un canal especial denominado *Rendezvous,* el mismo que permite el envío de mensajes de manera sincrónica entre procesos. La operación se ejecuta como una operación atómica simple, es decir, no existe ningún estado intermedio durante el proceso de enviar y recibir.

La declaración de este tipo de canal es similar a la de cualquier canal asincrónico, con la única diferencia de que su capacidad es cero.

Si en el puntero de instrucción de un proceso se encuentra la instrucción de envío (ver proceso *Tx* de la Figura 1.19 línea 4), se dice que el proceso desea participar de un encuentro. Si en el puntero de instrucción de otro proceso se encuentra en la instrucción de recepción (ver proceso *Rx* de la Figura 1.19 línea 10), el encuentro puede ser aceptado y se produce el intercambio de mensajes como se muestra en la Figura 1.20.

```
1.    chan ch = [0] of { byte};
2.
3.    proctype Tx() {
4.        ch ! 20;
5.      printf("Mensaje enviado\n")
6.    }
7.
8.    proctype Rx() {
9.        byte n;
10.     ch ? n;
11.       printf("Mensaje recibido, n = %d\n", n);
12.   }
13.
14.  init { run Tx();run Rx() }
```

Figura 1.19 Ejemplo de utilización de un canal *Rendezvous* en *PROMELA* [9]

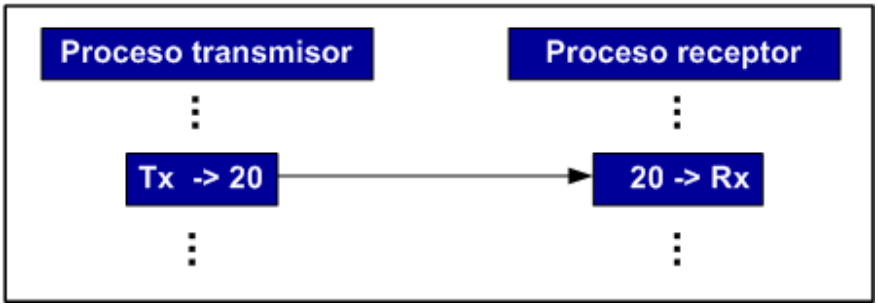


Figura 1.20 Funcionamiento del canal *Rendezvous* [9]

El problema típico de los sistemas distribuidos se denomina problema de la exclusión mutua o mejor conocido como el problema de la sección crítica. Este problema aparece cuando dos o más procesos tienen la capacidad de leer y escribir el valor de una o más variables globales.

La especificación del problema es:

- Cada uno de los procesos de los *N* existentes contiene una secuencia finita de instrucciones. Estas secuencias deberán dividirse en dos sub secuencias: crítica y no crítica. En la secuencia crítica deberán estar las variables compartidas por los procesos, mientras que en la sección no crítica todas las instrucciones que no involucren dichas variables.

Los supuestos que se deben resolver son:

- **Exclusión mutua:** Las secuencias de instrucciones de las secciones críticas de dos o más procesos no se deben intercalar.

- **Ausencia de *deadlocks* (estancamientos):** Si algunos procesos están tratando de entrar en su sección crítica, entonces al menos uno de ellos debe tener éxito.

- **Libertad de iniciación:** Si cualquier proceso intenta entrar en su sección crítica, entonces ese proceso debe tener éxito.

Para comprender mejor el problema se considera el programa de la Figura 1.21. Los dos procesos (*A* y *B*) contienen comandos de guarda (líneas 4 y 9). Ambos procesos desean sobrescribir la variable estado.

Al momento de ejecutar este programa, pueden presentarse dos casos:

- Solo un proceso pasa la condición inicial, por tanto tendrá la opción de cambiar la variable estado, dejando al otro bloqueado para siempre.

- Ambos procesos pasan la condición inicial y sobrescriben el valor de la variable.

```
1.    byte estado = 1;
2.
3.    proctype A() {
4.       (estado == 1);
5.      estado = estado + 1
6.    }
7.
8.    proctype B() {
9.       (estado == 1);
10.      estado = estado - 1
11.   }
12.
13.  init { run A();run B() }
```

Figura 1.21 Programa que está compuesto por dos procesos que comparten una variable global [1]

Debido a estos casos, el valor de la variable estado es impredecible y puede ser también no deseado. Los valores que la variable estado podría adoptar son: cero, uno y dos.

Para solucionar este problema, *PROMELA* introduce el concepto de secuencias atómicas. Una secuencia de instrucciones encerradas entre llaves precedidas de la palabra reservada *atomic,* indica que dichas instrucciones deben ser ejecutadas como si se trataran de una sola instrucción atómica, es decir, no pueden ser intercaladas con otras instrucciones.

En el programa de la Figura 1.22, se puede predecir que el valor de la variable estado puede ser cero o dos, dependiendo de cuál de los dos procesos se ejecute primero. El otro proceso quedará bloqueado para siempre.

Otra instrucción muy útil es *assert*, la cual permite explorar los estados de un programa. Junto a esta instrucción se pueden incluir condiciones que en el caso de ser falsas al momento de ejecutarse producen un error. Se utiliza principalmente dentro de las secciones críticas de un programa para verificar si los valores de las variables compartidas siguen la lógica que espera el programador.

```
1.    byte estado = 1;
2.
3.    proctype A() {
4.      atomic{
5.           (estado == 1);
6.         estado = estado + 1
7.      }
8.    }
9.
10.   proctype B() {
11.        atomic{
12.          (estado == 1);
13.          estado = estado – 1
14.     }
15.   }
16.
17.  init { run A();run B() }
```

Figura 1.22 Ejemplo de sentencias atómicas [1]

El programa de la Figura 1.23 muestra un ejemplo de uso de esta instrucción; el valor de la variable estado activará una de las dos instrucciones *assert* que contienen los procesos *A* y *B*, por tal razón el programador podrá determinar con seguridad cual de los dos procesos se ejecutó en primer lugar.

En los sistemas de comunicaciones, es frecuente encontrar procesos que reciban arbitrariamente diferentes tipos mensajes. Esta característica convierte a un sistema de comunicación en un sistema no determinístico.

```
1.      byte estado = 1;
2.
3.      proctype A() {
4.        assert(estado == 1);
5.        atomic{
6.             (estado == 1);
7.           estado = estado + 1
8.        }
9.      }
10.
11.     proctype B() {
12.          assert(estado == 1);
13.          atomic{
14.            (estado == 1);
15.            estado = estado – 1
16.       }
17.     }
18.
19.    init { run A();run B() }
```

Figura 1.23 Ejemplo de uso de la instrucción *assert* [1]

La Figura 1.24 muestra un ejemplo en el cual un proceso *TX* envía un mensaje denominado *msg* hacia un proceso *RX*. Una vez recibido el mensaje por parte de *RX*, este proceso devuelve otro mensaje al proceso *TX*. Este mensaje podría ser un *ack*, *nack* o *error* dependiendo de la selección arbitraria que realice *SPIN*.

El no determinismo en *SPIN* es usado para modelar valores arbitrarios de datos. Cuando se solicita un valor, *SPIN* realiza una selección no determinística de un valor de entre todos los posibles.

```
 1.      mtype = {msg,ack,nack,error};
 2.      chan tx2rx = [0] of {mtype};
 3.      chan rx2tx = [0] of {mtype};
 4.
 5.      proctype TX() {
 6.          mtype recep;
 7.            tx2rx!msg;
 8.            rx2tx?recep;
 9.            printf("%e",recep);
10.      }
11.
12.      proctype RX() {
13.            mtype recep;
14.            tx2rx?recep;
15.            if
16.            ::rx2tx!ack;
17.            ::rx2tx!nack;
18.           ::rx2tx!error;
19.             fi;
20.      }
21.
22.     init { run TX();run RX() }
```

Figura 1.24 Ejemplo de uso del no determinismo en *SPIN*

CAPÍTULO 2: TECNOLOGÍA *BLUETOOTH*

2.1. CARACTERÍSTICAS GENERALES DE LA TECNOLOGÍA *BLUETOOTH* [19]

La tecnología inalámbrica *Bluetooth* es un sistema de comunicación de corto alcance destinado a sustituir los cables de conexión entre dispositivos electrónicos portables o fijos. Algunas de las características de esta tecnología son:

- Al ser una tecnología de corto alcance los dispositivos pueden comunicarse a través del aire mediante ondas de radio a una distancia de 10 metros. Aumentando la potencia de transmisión la distancia puede variar hasta llegar aproximadamente a 100 metros.

- Es una tecnología de bajo consumo de energía y de bajo costo lo que es beneficioso para los dispositivos portables.

- Soporta voz y datos, permitiendo a los dispositivos transmitir cualquier tipo de contenido.

- Puede trabajar en cualquier parte del mundo ya que esta tecnología opera en la banda *ISM* (*Industrial*, *Scientific* and *Medical*) de 2.4 GHz, que no requiere de licencia para su uso y se encuentra disponible a nivel mundial.

2.2. PROTOCOLOS DEFINIDOS PARA LA TECNOLOGÍA *BLUETOOTH* [29]

La pila de protocolos *Bluetooth* se basa en el modelo de referencia *OSI*, y utiliza una arquitectura de protocolos que divide las diversas funciones de red en un sistema de

niveles. En conjunto permiten el intercambio transparente de información entre aplicaciones diseñadas de acuerdo con dicha especificación, y fomentan la interoperabilidad entre productos de diferentes fabricantes. [28]

En la Figura 2.1 se muestra la pila de protocolos *Bluetooth* y su comparación con el modelo *OSI*.

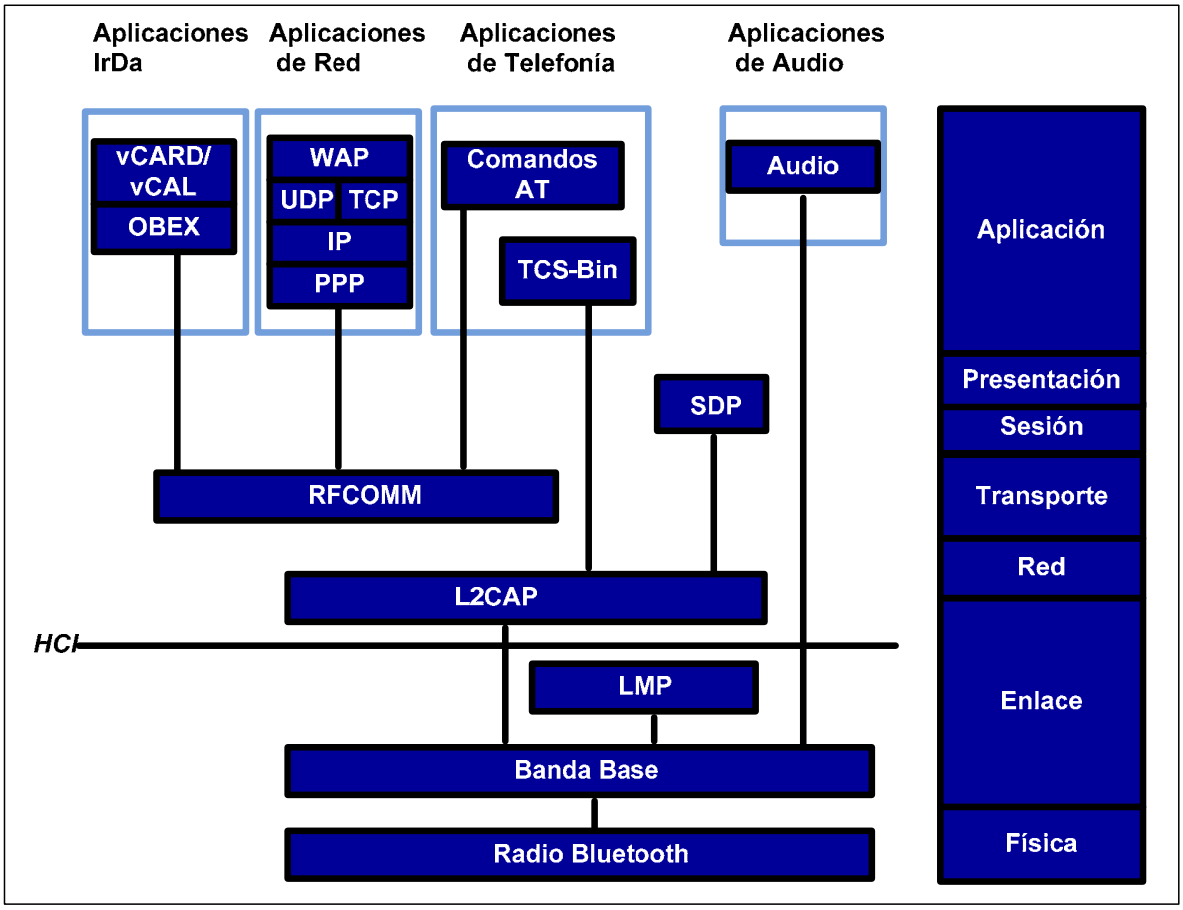


Figura 2.1 Pila de protocolos *Bluetooth* y su comparación con el modelo de referencia *OSI* [24]

Los protocolos pueden ser agrupados de la siguiente forma: [28]

- **Protocolos del núcleo *Bluetooth*:** Banda Base, Protocolo *LMP* (*Link Manager Protocol*), Protocolo *L2CAP* (*Logical Link Control and Adaptation Protocol*). Protocolo *SDP* (*Service Discovery Protocol*).

- **Protocolos de sustitución de cable:** *RFCOMM* (*Radio Frecuency Communication*).

- **Protocolos de control de telefonía:** *TCS*-Binario (*Telephony Control Protocol*), Comandos *AT*.

- **Protocolos adaptados:** *PPP* (*Point-to-Point Protocol*), *UDP* (*User Datagram Protocol*), *TCP* (*Transmission Control Protocol*), *IP* (*Internet Protocol*), *OBEX* (*OBject EXchange*), *WAP*, (*Wireless Application Protocol*).

Dado que los dispositivos dentro del sistema a implementar utilizan la versión *Bluetooth* 2.0 *EDR (Enhanced Data Rate)*, en los siguientes apartados se describirá los protocolos que forman parte del núcleo de *Bluetooth* siguiendo esta especificación. Adicionalmente se describirá el protocolo de sustitución de cables *RFCOMM* utilizado también por los dispositivos dentro del sistema.

2.2.1. PROTOCOLOS DEL NÚCLEO DE *BLUETOOTH*

2.2.1.1. Radio *Bluetooth*

2.2.1.1.1. Bandas de Frecuencia

Bluetooth opera en la banda *ISM* de 2.4 GHz, usando 79 canales de radio frecuencia con un ancho de banda de 1 MHz cada uno. En la Tabla 2.1 se muestra los rangos de frecuencia para diferentes regiones del mundo.

Región	Rango de Frecuencia (MHz)	Canales RF (MHz)
Europa y Estados Unidos	2400-2483.5	f = 2402+k k=0,….,78
Francia	2446.5-2483.5	f = 2454+k k=0,….,22
España	2445-2475	f = 2449+k k=0,….,22

Tabla 2.1 Bandas de Frecuencia y Canales *RF* [21]

Dado que la banda *ISM* está abierta, el sistema de radio *Bluetooth* se encuentra expuesto a múltiples interferencias por lo que utiliza la técnica *FHSS* (*Frequency-Hopping Spread Spectrum*).[11]

2.2.1.1.2. Velocidad

Se tienen dos modos de velocidad:

- **Modo Básico:** en este modo se emplea la modulación *GFSK* (*Gaussian Frequency Shift Keying).* Se tiene una velocidad de 1 Mbps.

- **Modo *EDR* (*Enhanced Data Rate*):** en este modo se emplean dos tipos de modulaciones. La modulación π/4-*DQPSK* (*Differential Quadrature Phase Shift Keying*) que permite una velocidad de 2 Mbps y la modulación 8-*DPSK* (*Diffrential Phase Shift Keying*) que permite una velocidad de 3 Mbps.

Para todos los esquemas de modulación se tiene una velocidad de símbolo de 1 Ms/s.

[11] *FHSS*: Esta técnica utiliza una señal portadora que cambia de frecuencia durante la transmisión. Tanto el transmisor como el receptor deben usar la misma secuencia de frecuencias de portadora, el orden de los saltos sigue una secuencia pseudoaleatoria. Con esto se minimiza la posibilidad de que dos emisores usen la misma frecuencia simultáneamente evitando así la interferencia.[70]

2.2.1.1.3. Potencia [21] [28]

Bluetooth define tres clases de dispositivos de acuerdo a la potencia de transmisión como se muestra en la Tabla 2.2

El equipo receptor debe poseer una sensibilidad de al menos -70 dBm y la tasa de error admisible (*BER*, *Bit Error Rate*) debe ser menor o igual a 0.1 % en modo de velocidad básico y menor o igual a 0.01 % en modo de velocidad *EDR*. Estos valores se encuentran establecidos en la especificación *Bluetooth* versión 2.0 con *EDR*. [29]

Clase	Potencia de salida máxima	Potencia de salida mínima	Alcance
1	100 mW (20 dBm)	1 mW (0dBm)	100 m
2	2.5 mW (4 dBm)	0.25 mW (-6 dBm)	10 m
3	1 mW (0 dBm)	----------	1 m

Tabla 2.2 Niveles de Potencia en *Bluetooth* [29]

2.2.1.2. Banda Base [28]

Aquí se encuentra el controlador del enlace. Entre las tareas que realiza esta capa se tiene: sincronización, transmisión de la información, detección y corrección de errores, división lógica de canales, control del enlace, direccionamiento y formato de paquetes.

Además esta capa define la red básica de *Bluetooth*: *piconet* y *scatternet,* que se muestran en la Figura 2.2. Una *piconet* es una colección de dos o más dispositivos *Bluetooth* compartiendo el mismo canal físico (están sincronizados a un reloj común y usan una misma secuencia de salto de frecuencia). Está conformada por un

dispositivo maestro y puede llegar a tener hasta siete dispositivos esclavos activos, además de dispositivos esclavos en modo *park*[12].

La topología *Bluetooth* permite la interconexión de varias *piconets* formando una *scatternet*. Un dispositivo puede pertenecer a diferentes *piconets* pero solo estar activo en una de ellas a la vez. En ocasiones el dispositivo que es el maestro de una *piconet* puede ser esclavo de otra *piconet*.

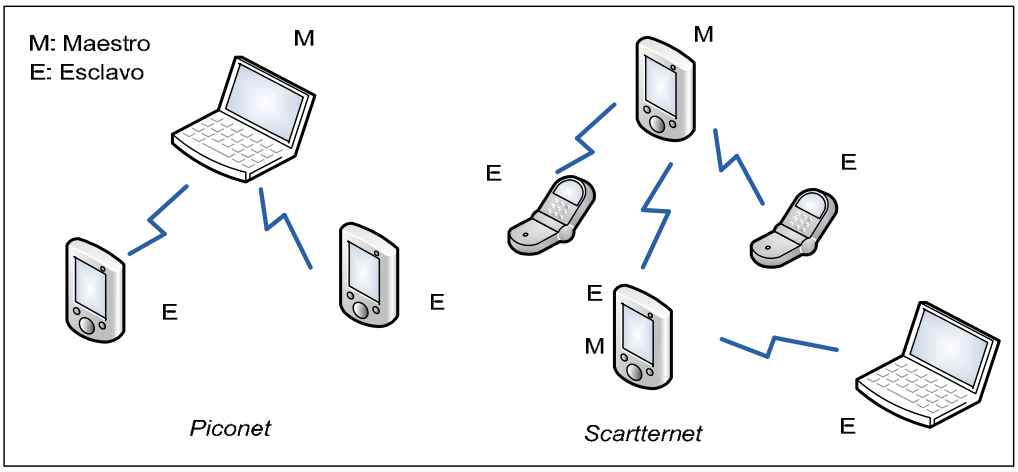


Figura 2.2 *Piconet y Scarttnet* [24]

2.2.1.2.1. Direccionamiento Bluetooth

A cada dispositivo *Bluetooth* se le asigna una dirección *BD_ADDR* (*Bluetooth Device Address*) única de 48 bits, la misma que es obtenida de la autoridad de registro de la IEEE[13]. Esta dirección es dividida en los siguientes campos:

- *LAP* (*Lower Address Part*): tiene 24 bits y es la parte baja de la dirección.
- *UAP* (*Upper Address Part*): tiene 8 bits y es la parte alta de la dirección.

[12] Modo *Park*: Este modo se lo emplea cuando un esclavo no necesita participar en el canal de la *piconet* transmitiendo datos pero aún desea permanecer sincronizado con el dispositivo maestro. [29]
[13] *IEEE (Institute of Electrical and Electronics Engineers)*

- *NAP* (*Non-significant Address Part*): tiene 16 bits y es la parte no significativa de la dirección.

En la Figura 2.3 se muestra un ejemplo de la dirección *BD_ADDR.*

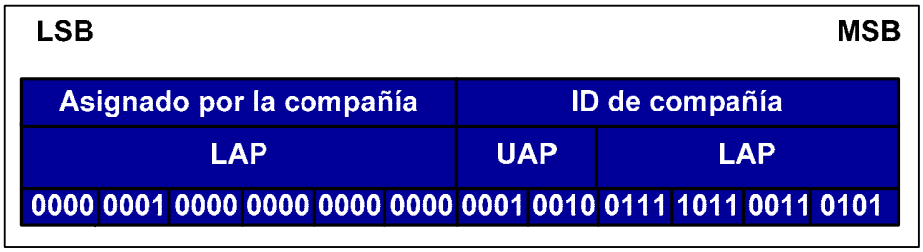


Figura 2.3 Formato de la dirección *BD_ADDR* [29]

2.2.1.2.2. Canales Físicos [23]

Están definidos por una secuencia pseudoaleatoria de salto escogida de entre los 79 canales (para las regiones de Francia y España se tiene 23 canales disposnibles) de radiofrecuencia disponibles en la banda *ISM* de 2.4 GHz usando la técnica *FHSS*. Tanto el dispositivo transmisor como el receptor deben usar la misma secuencia de frecuencias de portadora.

La secuencia de salto es única para cada *piconet*. Todos los dispositivos conectados a la *piconet* están sincronizados con el canal en salto y tiempo. La velocidad de salto es de 1600 saltos/segundo en el estado de conexión y de 3200 saltos/segundo en los estados *Inquiry*[14] y *Page*[15].

Para emular una trasmisión *full duplex* los canales físicos utilizan el esquema *TDD* (*Time Division Duplex*), en el que cada canal está dividido en ranuras de tiempo o *slots,* cada *slot* corresponde a una frecuencia de salto y tiene una duración de 625 us. Los dispositivos *Bluetooth* alternan entre la transmisión y recepción de datos

[14] *Inquiry*: Es utilizado para descubrir nuevos dispositivos dentro del rango de cobertura. [28]
[15] *Page*: Es utilizado para la búsqueda de los dispositivos con los cuales se quiere establecer una conexión. [28]

de un *slot* a otro. El rango de numeración va de 0 a 2^{27}-1 en forma cíclica con un ciclo de duración 2^{27} .El maestro comienza su transmisión en los *slots* pares, mientras que los esclavos lo hacen en los *slots* impares para evitar fallas en la trasmisión. En la Figura 2.4 se muestra la transmisión entre un dispositivo maestro y un esclavo.

Los dispositivos *Bluetooth* usan paquetes para la trasmisión de sus datos, cada paquete corresponde a un *slot* pero para permitir comunicaciones más eficientes se usa también paquetes *multislot* que pueden extenderse hasta una duración de 5 *slots*.

Los canales físicos son identificados por el código de acceso que se encuentra en el paquete, junto con el reloj y la dirección del dispositivo *Bluetooth* maestro.

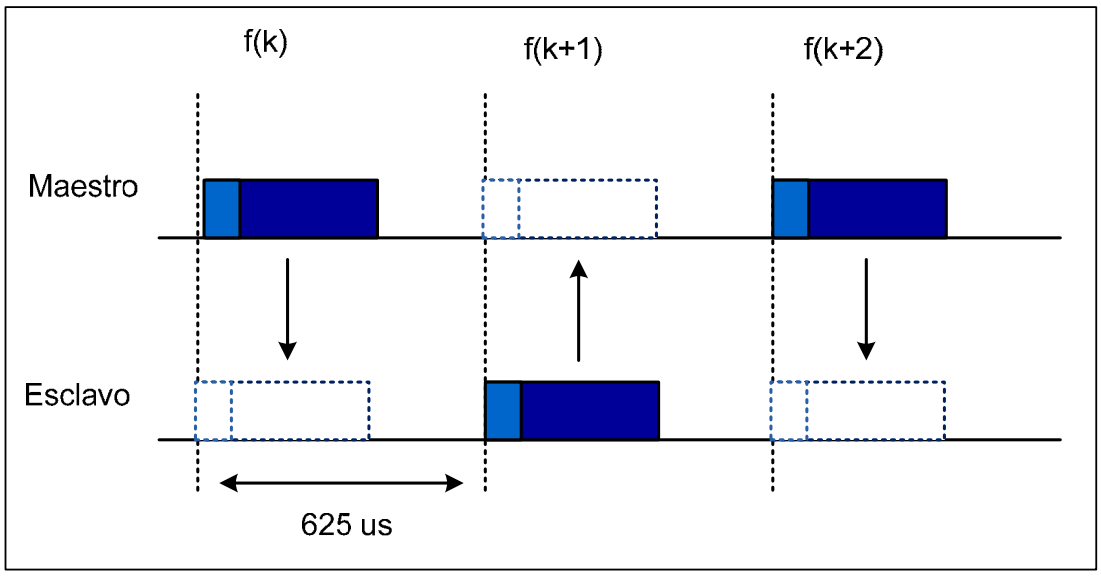


Figure 2.4 Transmisión entre un dispositivo maestro y un esclavo. [29]

Bluetooth define cuatro canales físicos: [23]

- ***Piconet:*** Este canal es usado para la comunicación entre los dispositivos conectados.

- ***Adapted Piconet*:** Este canal ofrece la posibilidad de utilizar un subconjunto de los 79 canales de radio frecuencia (mínimo 20) para evitar las frecuencias en las que la interferencia es demasiado alta y facilitar la coexistencia con otros sistemas es la misma banda de frecuencias. Además el dispositivo esclavo usa la misma frecuencia que el dispositivo maestro en su transmisión anterior.

- ***Page-Scan*:** A través de este canal se envían las solicitudes de *Page* hacia los dispositivos con los cuales se quiere establecer una conexión. Los dispositivos que estén preparados para aceptar conexiones escuchan a través de este canal dichas solicitudes y envían sus respectivas respuestas.

- ***Inquiry-Scan*:** Es para que los dispositivos que quieran descubrir otros dispositivos envíen sus solicitudes de *Inquiry*. Los dispositivos que esperan ser descubiertos escuchan a través de este canal a la espera de las solicitudes de *Inquiry* y envían sus respectivas respuestas.

2.2.1.2.3. Enlaces Físicos [29]

Dentro del sistema *Bluetooth* un enlace físico es un concepto virtual y representa una conexión de Banda Base entre dispositivos *Bluetooth*. Es un enlace punto a punto entre el dispositivo maestro y un esclavo y siempre se encuentra presente cuando el esclavo está sincronizado dentro de la *piconet*.
Los enlaces físicos tienen propiedades que son manejadas mediante el protocolo *LMP* (*Link Manger Protocol*) entre las que se tiene:

- Control de potencia.
- Supervisión del enlace.
- Encripción.
- Manejo de la calidad del canal por cambio de velocidad.
- Control de paquetes *multi-slot.*

Hay dos tipos de enlaces físicos:

- **Activo**: Un enlace físico entre un dispositivo maestro y un esclavo es activo si existe un transporte lógico *ACL* entre los dispositivos.

- **Aparcado**: Existe un enlace físico de este tipo entre un dispositivo *Bluetooth* maestro y un esclavo cuando el esclavo no necesita participar en el canal de la *piconet* transmitiendo datos pero aún desea permanece sincronizado con el canal.

2.2.1.2.4. Transportes Lógicos

Se tienen definidos 5 tipos de transportes lógicos que se pueden establecer entre el dispositivo maestro y los esclavos:

- *SCO* (*Sysnchronous Connection-Oriented*).
- *eSCO* (*Extended Sysnchronous Connection-Oriented*).
- *ACL* (*Asynchronous Connection-Oriented*).
- *ASB* (*Active Slave Broadcast*).
- *PSB* (*Parked Slave Broadcast*).

a. SCO (Synchronous Connection-Oriented)

Es un transporte simétrico, punto a punto entre el maestro y un esclavo específico. Para establecerlo se reservan dos *slots* consecutivos en intervalos regulares. La reserva de los *slots* la realiza el dispositivo maestro cuando se establece la conexión con el dispositivo esclavo. Puede ser considerado como una conexión de conmutación de circuitos. Este tipo de enlace no requiere asegurar la entrega de paquetes y es utilizado principalmente para la trasmisión de voz, soportando una tasa de transferencia de 64 Kbps; los paquetes *SCO* nunca son retransmitidos.

El maestro puede soportar hasta tres enlaces *SCO* a un mismo esclavo o diferentes esclavos, mientras que los esclavos pueden soportar tres enlaces *SCO* de un mismo maestro, o dos enlaces *SCO* si éstos provienen de diferentes maestros.

Los enlaces *SCO* tienen prioridad sobre los enlaces *ACL*, por lo que los enlaces *ACL* usan los *slot* que no han sido reservados.

b. eSCO (Extended Sysnchronous Connection-Oriented)

Es un transporte punto a punto que puede ser simétrico o asimétrico entre el dispositivo maestro y un esclavo específico. También realiza la reserva de *slots,* pero adicionalmente brinda la posibilidad de realizar un número limitado de retransmisiones. Si las retransmisiones son requeridas estas pueden llevarse a cabo en los *slots* que siguen a los *slots* reservados.

c. ACL (Asynchronous Connection-Oriented)

Es un transporte punto-multipunto entre el maestro y uno o más esclavos activos dentro de la *piconet,* puede ser simétrico o asimétrico. No hay reserva del canal, usa *slots* por demanda, los mismos que pueden ser 1, 3 o 5 *slots* consecutivos. Es considerado como una conexión de conmutación de paquetes. Todo dispositivo esclavo activo dentro de la *piconet* tiene un trasporte lógico *ACL* hacia el dispositivo maestro de la *piconet* conocido como *ACL,* por defecto el mismo que es creado cuando un dispositivo se une a la *piconet*. Solamente puede existir un enlace *ACL* entre cada par maestro-esclavo.

Este trasporte es usado para intercambiar información de control y datos de usuario por lo que se aplica la retransmisión de los paquetes errados para asegurar la integridad de la información.

Los paquetes *ACL* no direccionados a un esclavo específico son considerados paquetes de *broadcast* y son leídos por todos los esclavos.

d. ASB (Active Slave Broadcast)

Es usado para transportar tráfico de usuario *L2CAP* a todos los dispositivos que se encuentren actualmente conectados al canal físico que es usado por el *ASB*. El tráfico es unidireccional del dispositivo maestro de la *piconet* hacia los esclavos. No es confiable ya que no se realizan retransmisiones.

e. PSB (Parked Slave Broadcast)

Es usado para la comunicación entre el dispositivo maestro y los esclavos que se encuentran en modo *park,* al igual que el *ASB* no es confiable. Lleva tráfico de control *LMP* y de usuario *L2CAP*.

2.2.1.2.5. Enlaces Lógicos

Se han definido 5 enlaces lógicos:

- ***LC* (*Link Control*):** lleva información de control de enlace de bajo nivel como *ARQ*, control de flujo, tipo de paquete *Bluetooth*. Esta información va en la cabecera del paquete.

- ***ACL-C* (*ACL Control*):** es usado para llevar señalización *LMP* entre los dispositivos en la *piconet* en paquetes *DM1* sobre el transporte lógico *ACL*.

- ***ACL-U* (*User Asynchronous/Isochronous*):** lleva datos de usuario *L2CAP* que pueden ser asincrónicos o isocrónicos[16] sobre el transporte lógico *ACL*. Estos mensajes dependiendo de su tamaño deberán ser fragmentados y pueden ser transmitidos en uno o varios paquetes de Banda Base.

- ***SCO-S* (*UserSynchronous*):** lleva datos sincrónicos de usuario sobre el transporte lógico *SCO*.

- ***eSCO-S* (*User Extended Synchronous*):** lleva datos sincrónicos de usuario sobre el transporte lógico *eSCO*.

2.2.1.2.6. Formato de paquete Bluetooth

En *Bluetooth* los datos son enviados en paquetes y siguen el formato *Little Endian,* en el que el bit menos significativo *LSB* es el primer bit en ser enviado por el aire.

El formato general del paquete para el modo de velocidad básica se muestra en la Figura 2.5.

Cada paquete contiene tres entidades: el código de acceso, la cabecera y la carga útil.

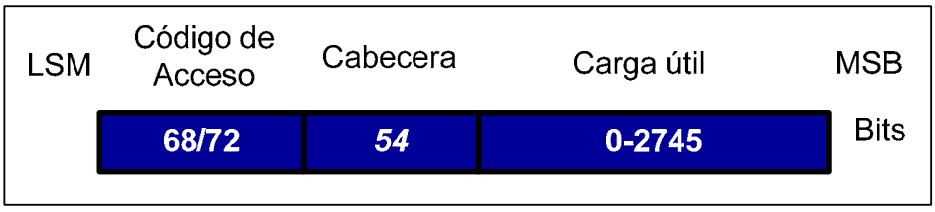


Figura 2.5 Formato general de un paquete de Banda Base para el modo Básico [29]

[16] Datos Isocrónicos: Se caracterizan porque la información de temporización está incluida en la cadena de datos, Los datos isócronos necesitan temporización de forma precisa como es el caso de enviar audio comprimido sobre un enlace ACL [28]

Para el modo de velocidad *EDR* se tiene el formato de paquete mostrado en la Figura 2.6.

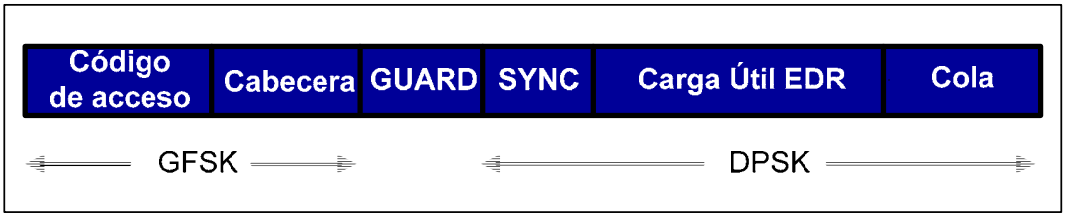


Figura 2.6 Formato general de un paquete de Banda Base para el modo *EDR*[29]

Cada paquete *EDR* está formado por 6 entidades: el código de acceso, la cabecera, un periodo de guarda (*GUARD*), una secuencia de sincronización (*SYNC*), la carga útil *EDR* y una cola.

El código de acceso y la cabecera son idénticos en formato y modulación que los paquetes de modo básico, mientras que la secuencia de sincronización, la carga útil *EDR* y la cola usan el esquema de modulación *EDR*. El tiempo de guarda (campo *GUARD* del paquete de Banda Base para el modo *EDR*) permite la transición entre los esquemas de modulación y tiene una duración entre 4.75 us y 5.25 us. La secuencia de sincronización consta de 11 símbolos *DPSK* generados a partir de una secuencia de bits preestablecidos y la cola consta de 2 símbolos cuyos bits deben ser todos cero.

A continuación se detalla cada uno de los campos del paquete de Banda Base en modo Básico:

a. Código de acceso [21][30]

El código de acceso identifica todos los paquetes intercambiados sobre el canal físico.

Existen tres tipos de código de acceso:

- ***CAC* (*Channel Access Code*):** identifica una *piconet* y es incluido en todos los paquetes transmitidos en la *piconet*.

- ***DAC* (*Device Access Code*):** se utiliza en el procedimiento de establecimiento de conexión.

- ***IAC* (*Inquiry Access Code*):** se utiliza en el procedimiento de búsqueda de dispositivos.
  Puede ser de tipo general (*GIAC*) y de tipo dedicado (*DIAC*). El código de acceso de tipo general es usado para detectar otros dispositivos *Bluetooth* dentro del alcance, y el de tipo dedicado cuando se requiere descubrir dispositivos *Bluetooth* específicos que comparten características en común.

b. Cabecera

Contiene información de control del enlace y está formada por 6 campos como se muestra en la Figura 2.7.

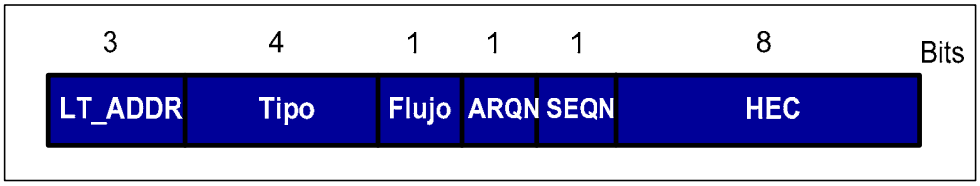


Figura 2.7 Formato de la cabecera del paquete de Banda Base [29]

- **Dirección *LT_ADDR***: a cada dispositivo esclavo activo dentro de la *piconet* se le asigna una dirección *LT_ADDR* para distinguirlos individualmente. Indica el dispositivo esclavo destino del paquete en una transmisión maestro-esclavo e indica el dispositivo esclavo fuente en una

transmisión esclavo-maestro. La dirección 000 está reservada para paquetes de *broadcast*.

- **TIPO:** especifica qué tipo de paquete es usado y depende del tipo de transporte lógico (*ACL*, *SCO* y *eSCO*) asociado con el paquete.

- **Flujo:** empleado para el control de flujo. Este bit en 0 indica una señal de parada y el bit en 1 una señal para continuar la transmisión. Esta señal solo concierne a los paquetes *ACLs.*

- ***ARQN***: es usado para informar una transferencia exitosa de la carga útil con *CRC* y puede ser un *ACK* con el bit *ARQN* = 1 o un *NAK* con el bit *ARQN* = 0 con el que se está solicitando una retransmisión. El éxito de la recepción es comprobado mediante el cálculo de *CRC*.

- ***SEQN***: proporciona una secuencia numérica para ordenar el flujo de paquetes de datos. Por cada nuevo paquete de datos con *CRC* transmitido el bit *SEQN* es invertido.

- ***HEC***: para la verificar la integridad de la cabecera. Se usa el polinomio generador $x^8+x^7+x^5+x^2+x+1$.

La cabecera consta de 18 bits y es codificada con *FEC* de tasa 1/3 que se basa en repetir 3 veces cada bit dando como resultado una cabecera de 54 bits de longitud.

c. Carga Útil [30]

Transporta la información de las capas superiores y se puede distinguir dos campos: el campo de voz (síncrono) y el campo de datos (asíncrono) como se muestra en la Figura 2.8. Los paquetes *ACL* solo tiene el campo de datos y los

paquetes *SCO* y *eSCO* solo tienen el campo de voz con excepción de los paquetes DV que tienen los dos campos.

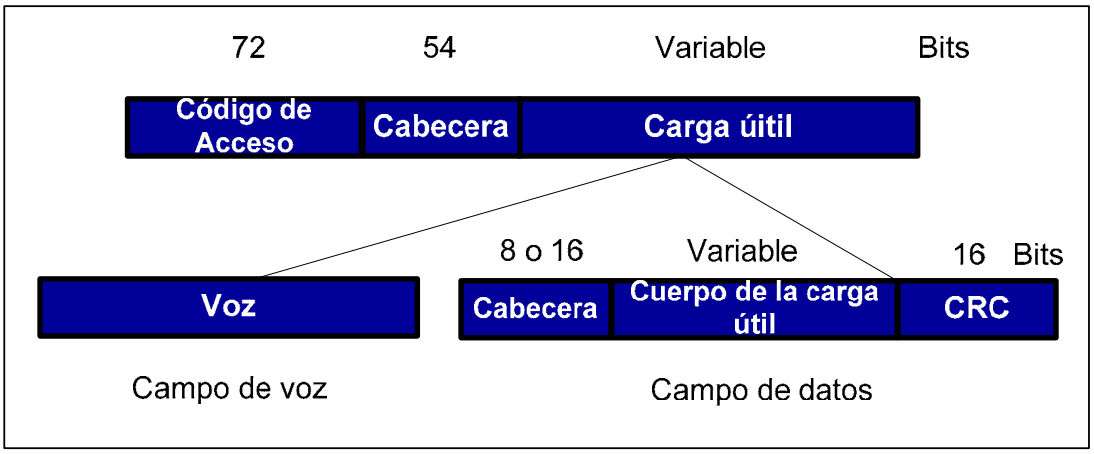


Figura 2.8 Formato del campo Carga Útil del paquete de Banda Base en modo de velocidad Básico [30]

El campo de datos se encuentra formado por una cabecera que lleva información de control, un cuerpo de la carga útil que contiene los datos propiamente dichos que pueden ser información *LMP* o *L2CAP* y un *CRC* de 16 bits para la verificación de errores en la carga útil. Para su cálculo se emplea el polinomio generador $x^{16}+x^{12}+x^{5}+1$.

2.2.1.2.7. Tipos de Paquetes [30]

Los paquetes de Banda Base se han dividido en cuatro grupos. En el grupo 1 están los paquetes de control los mismos que ocupan un solo *slot* y son comunes a todos los tipos de transporte lógico. En los grupos 2, 3 y 4 están los paquetes que ocupan 1, 3, y 5 *slots* respectivamente.

El tipo de paquete se puede identificar mediante el campo Tipo de la cabecera del paquete *Bluetooth*.

La Tabla 2.3 muestra la clasificación especificando para cada tipo de transporte lógico los paquetes correspondientes.

Grupo	Tipo de paquete	Número de *Slots*	Transporte Lógico *SCO* (1 Mbps)	Transporte Lógico *eSCO* (1 Mbps)	Transporte Lógico *eSCO* (2-3 Mbps)	Transporte Lógico *ACL* (1 Mbps)	Transporte Lógico *ACL* (2-3 Mbps)
1	0000	1	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*
	0001	1	*POLL*	*POLL*	*POLL*	*POLL*	*POLL*
	0010	1	*FHS*	reservado	reservado	*FHS*	*FHS*
	0011	1	*DM1*	reservado	reservado	*DM1*	*DM1*
2	0100	1	--	--	--	*DH1*	*2-DH1*
	0101	1	*HV1*	--	--	--	--
	0110	1	*HV2*	--	*2-EV3*	--	--
	0111	1	*HV3*	*EV3*	*3-EV3*	--	--
	1000	1	*DV*	--	--		*3-DH1*
	1001	1	--	--	--	*AUX1*	*AUX1*
3	1010	3	--	--	--	*DM3*	*2-DH3*
	1011	3	--	--	--	*DH3*	*3-DH3*
	1100	3	--	*EV4*	*2-EV5*	--	--
	1101	3	--	*EV5*	*3-EV5*	--	--
4	1110	5	--	--	--	*DM5*	*2-DH5*
	1111	5	--	--	--	*DH5*	*3-DH5*

Tabla 2.3 Tipos de paquetes de Banda base [29]

a. Paquetes de control [30]

Hay 5 paquetes de control. Adicional a los paquetes listados en el grupo 1 de la Tabla 2.3, se tiene el paquete *ID* que no se lo incluye en ésta debido a que este paquete no tiene cabecera.

- **Paquete *ID*:** es un paquete de identificación**,** consta únicamente del código de acceso (*DAC* o *IAC*).

- **Paquete *NULL*:** está formado por el código de acceso al canal (*CAC*) y la cabecera. Sirve para enviar información de control relacionada al éxito de una transmisión previa o al estado del *buffer*.

- **Paquete *POLL*:** al igual que el paquete *NULL*, no contiene carga útil y es usado por el maestro para forzar al esclavo a responder.

- **Paquete *FHS*:** se usa para la sincronización, intercambiar información de identidad y de reloj.

- **Paquete *DM1*:** se incluye en el primer grupo ya que a más de transportar datos de usuario se utiliza para el envío de información de control.

b. Paquetes *ACL*

Los paquetes *ACL* son para transmisiones sobre el transporte lógico *ACL*. La información que llevan pueden ser datos de usuario o de control. En la Tabla 2.4 se muestran los paquetes *ACL* con sus respectivas velocidades.

Para el modo de velocidad básico se han definido los paquetes: *DM1*, *DH1*, *DM3*, *DH3*, *DM5*. *DH5* y *AUX1*. Adicionalmente para el modo de velocidad *EDR* se han definido los paquetes: 2-*DH1*, 3-*DH1*, 2-*DH3* y 3-*DH3*.

c. Paquetes *SCO*

Los paquetes *SCO* son para transmisiones sobre el transporte lógico *SCO*. Se tiene los paquetes *HV1*, *HV2*, *HV3* y *DV*. Son usados para transmisiones de voz a una velocidad de 64 Kbps.

Los paquetes *HV* no incluyen un campo *CRC* y no deben ser retransmitidos.

Los paquetes *DV* llevan voz y datos, por lo que en la carga útil se encuentra dividida en un campo de voz de 80 bits y un campo de datos de hasta 150 bits. La voz y los datos deben ser tratados independientemente.

d. Paquetes *eSCO*

Los paquetes *eSCO* son para transmisiones sobre el transporte lógico *eSCO*. Se los emplea para transmitir voz.

Para el modo de velocidad básico se han definido los paquetes *EV3*, *EV4* y *EV5* y para el modo de velocidad *EDR* los paquetes: *2-EV3*, *3-EV5*, *3-EV3 y 3-EV5*.

En la Tabla 2.5 se muestran los paquetes *SCO* y *eSCO* con sus respectivas velocidades

Tipo paquete	Carga útil (bytes)	*FEC*	*CRC*	Velocidad simétrica máxima (Kbps)	Velocidad asimétrica máxima (Kbps)	
					Uplink	*Downlink*
DM1[17]	0-17	2/3	si	108,8	108,8	108,8
DH1[18]	0-27	no	si	172,8	172,8	172,8
DM3	0-121	2/3	si	258,1	387,2	54,4
DH3	0-183	no	si	390,4	585,6	86,4
DM5	0-224	2/3	si	286,7	477,8	36,3
DH5	0-339	no	si	433,9	723,2	57,6
AUX1	0-29	no	no	185,6	185,6	185,6
2-DH1	0-54	no	si	343,6	345,6	345,6
2-DH3	0-367	no	si	782,9	1174,4	172,8
2-DH5	0-679	no	si	869,7	1448,5	115,2
3-DH1	0-83	no	si	531,2	531,2	531,2
3-DH3	0-552	no	si	1177,6	1766,4	235,6
3-DH5	0-1021	no	si	1306,9	2178,1	177,1

Tabla 2.4 Paquetes *ACL* [29]

17 *DM* (*Data Medium Rate*)

18 *DH* (*Data High Rate*)

Tipo paquete	Carga útil (bytes)	*FEC*	*CRC*	Velocidad simétrica máxima (Kbps)
HV1	10	1/3	no	64
HV2	20	02-mar	no	64
HV3	30	no	no	64
DV	10 (Voz) , 0-9 (Datos)	2/3(Datos)	si (Datos)	64 (Voz), 57,6 (Datos)
EV3	1-30	no	si	96
EV4	1-120	2/3	si	192
EV5	1-180	no	si	288
2-EV3	1-60	no	si	192
2-EV5	1-360	no	si	576
3-EV3	1-90	no	si	288
3-EV5	1-540	no	si	864

Tabla 2.5 Paquetes *SCO* y *eSCO* [29]

2.2.1.2.8. Chequeo de Errores

Se puede comprobar errores en los paquetes o la entrega incorrecta usando el código de acceso al canal *CAC,* el *HEC* en la cabecera, y el *CRC* en la carga útil.

Al recibir el paquete primero se chequea el código de acceso al canal para prevenir que el receptor acepte paquetes que pertenecen a otra *piconet*.

2.2.1.2.9. Corrección de Errores

Hay tres esquemas de corrección de errores definidos para *Bluetooth*:

- *FEC* a la tasa de 1/3.
- *FEC* a la tasa de 2/3.
- Esquema *ARQ.*

El propósito del esquema *FEC* en la carga útil de datos es reducir el número de retransmisiones. Sin embargo, en un ambiente tolerante libre de errores, *FEC* incorpora sobrecarga innecesaria que reduce el rendimiento. Por lo que existe flexibilidad en usar o no el esquema *FEC* en la carga útil de acuerdo al tipo de paquete.

La cabecera de paquete siempre es protegida por el esquema *FEC* de tasa 1/3, ya que ésta contiene información valiosa del enlace y debe ser capaz de soportar una mayor cantidad de bits erróneos.

a. *FEC* a la tasa de 1/3

Se lo implementa repitiendo tres veces cada bit como se muestra en la Figura 2.9. Se lo utiliza para toda la cabecera así como para el campo de voz en el paquete *HV1*.

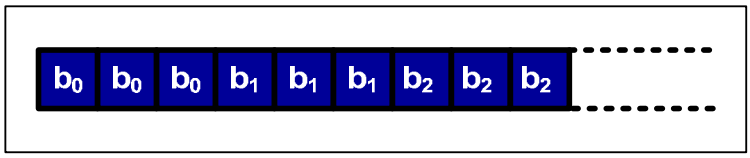


Figura 2.9 Esquema de codificación con bit de repetición [29]

b. *FEC* a la tasa de 2/3

Este esquema *FEC* usa un código *Hamming* (15,10) en el que a partir de 10 bits de información se obtiene una palabra código de 15 bits al añadir 5 bits de redundancia.

Este código puede corregir todos los errores simples y detectar todos los errores dobles en cada palabra de código.

El esquema *FEC* 2/3 es usado en los paquetes *DM*, *FHS*, *HV2*, *EV4* y en el campo de datos del paquete *DV.*

c. Esquema *ARQ*

Bluetooth emplea un esquema *ARQ* basado en un sistema *Stop and Wait* con un periodo de espera de un *slot.* Es decir el éxito o fracaso de la transmisión se indica en el campo *ARQN* del paquete de vuelta.

Mediante este esquema los paquetes *DM*, *DH*, *EV* y el campo de datos del paquete *DV* son transmitidos y retransmitidos hasta que sea devuelto por el destino un acuse de recibo de una recepción exitosa.

Las retransmisiones son controladas por el temporizador *flush timeout*[19]. Este temporizador es fijado durante los procedimientos de conexión y puede tener un valor entre 625 us y 1.28 s. Por defecto este valor está fijado en 0xFFFF (infinito) para tener un canal fiable, es decir que las retrasmisiones se llevarán a cabo hasta que ocurra una pérdida del enlace físico.

2.2.1.2.10. Establecimiento de una conexión [31]

Para el establecimiento de una conexión en *Bluetooth* los dispositivos pueden estar en ciertos estados como son: [28]

- ***Inquiry*:** Este estado es utilizado para descubrir otros dispositivos dentro del área de cobertura. En este estado el dispositivo adquiere información de los dispositivos como su dirección *BD_ADDR.*

[19] Temporizador *Flush Timeout*: Indica durante cuánto tiempo se puede llevar a cabo la retransmisión del paquete. Después de este tiempo se detiene las retransmisiones forzando al controlador del enlace a tomar los siguientes datos [29].

- ***Scan:*** Cuando un dispositivo *Bluetooth* está en modo *STANDBY* (dormido), periódicamente escucha el canal, esperando a ser descubiertos por otros dispositivos.

- ***Page:*** Es un estado de búsqueda que es utilizado, generalmente, luego del estado de *Inquiry* para establecer las conexiones.

2.2.1.2.11. Modos de ahorro de energía [29]

Los modos de ahorro de energía permiten optimizar su uso de acuerdo a los requerimientos y son: [28]

- ***Hold*****:** "el maestro puede ordenar al esclavo quedarse en modo *Hold*. Durante este periodo no hay comunicación posible entre esclavo y maestro. Cuando el periodo expira el esclavo vuelve al canal y permanece sincronizado.

- ***Park*****:** el esclavo también puede ser puesto en modo *Park*. En este caso el esclavo entra a un ciclo de trabajo en donde los intervalos de escucha del maestro son más largos.

- ***Sniff*****:** el esclavo no escucha todas las ranuras de tiempo, sino que solo escucha algunas. Para entrar al modo *Sniff*, el esclavo y maestro deben acordar en qué ranuras el esclavo pondrá atención al canal. "

2.2.1.3. Protocolo *LMP* (*Link Manager Protocol*)

En cada dispositivo *Bluetooth* se tiene un administrador del enlace, los mismos que se comunican a través del protocolo *LMP* para la configuración y control del enlace.

El protocolo *LMP* se encarga del control de operación de los dispositivos dentro de la *piconet*, de la seguridad (encriptación y autenticación), negociación de los paquetes de Banda Base, del control de potencia entre otras cosas.

2.2.1.4. Protocolo *L2CAP* (*Logical Link Control and Adaptation Protocol*) [25][23]

Se encarga de adaptar los protocolos de capas superiores al protocolo de Banda Base. Brinda servicios de datos orientados y no orientados a conexión a las capas superiores. *L2CAP* permite a las capas superiores enviar y recibir paquetes de datos de hasta 64 Kbytes de longitud y está definido únicamente para enlaces *ACL*.

L2CAP sigue un modelo de comunicación basado en canales lógicos que permite el flujo de datos entre entidades *L2CAP* en dispositivos remotos.

Los canales *L2CAP* pueden ser:

- Canales de señalización bidireccionales que transportan comandos. Estos canales son usados para crear y establecer canales orientados a conexión y para negociar cambios en las características de estos canales.

- Canales orientados a conexión para conexiones punto a punto bidireccionales.

- Canales unidireccionales no orientados a conexión que soportan conexiones punto-multipunto, permitiendo que una entidad local *L2CAP* sea conectada a un grupo de dispositivos remotos.

Sus funciones principales son:

- Multiplexación de protocolos de capas superiores.

L2CAP multiplexa los protocolos de capas superiores con el fin de enviar varios protocolos sobre un canal Banda Base. Entre los diferentes protocolos que se pueden tener en capas superiores están el protocolo *SDP, RFCOMM* y *TCS.*

- Segmentación y Reensamblado

  Los paquetes de datos definidos por el protocolo Banda Base están limitados en tamaño. Los paquetes grandes *L2CAP* deben ser segmentados en varios paquetes más pequeños antes de transmitirse.

  En el receptor los paquetes pequeños recibidos de Banda Base son reensamblados en paquetes *L2CAP*.

2.2.1.5. Protocolo *SDP (Service Discovery Protocol)*

El protocolo *SDP* proporciona a las aplicaciones un mecanismo para buscar y encontrar servicios disponibles en dispositivos *Bluetooth* y determinar los atributos específicos de éstos.
SDP define una topología Cliente/Servidor. Cualquier dispositivo puede asumir cualquier rol en un tiempo dado, actuando a veces como Cliente del servicio y a veces como un proveedor de servicio (Servidor).

El proveedor de servicio necesita mantener una lista que describa los servicios que proporciona; esta lista es el registro de servicio (*service registry*), el cual consiste de una colección de atributos de servicio. Estos atributos contienen información como la clase de servicio, información sobre los protocolos necesarios para interactuar con el servicio.

El Cliente y el Servidor necesitan un formato estándar para representar el servicio de interés. Para este propósito, *SDP* define los identificadores universales únicos *UUIDs* (*Universally Unique Identifiers*). Un *UUID* está compuesto por un valor de 128 bits.

2.2.2. PROTOCOLO *RFCOMM* [59]

El protocolo *RFCOMM* se basa en un subconjunto del estándar *ETSI*[20] TS 07.10[21] y emula señales de control y datos *RS-232* ofreciendo capacidades de transporte a servicios de capas superiores que usan una línea serial como mecanismo de transporte.

RFCOMM soporta múltiples conexiones simultáneas entre dos dispositivos *Bluetooth* sin embargo el número de conexiones simultáneas es específico de la implementación. En la Figura 2.10 se muestra el esquema de emulación para múltiples puertos seriales.

RFCOMM no lleva a cabo ningún mecanismo para el control de errores sobre los datos, se basa en Banda Base y *L2CAP* para proporcionar una entrega confiable de los datos.

Cada conexión entre una aplicación cliente y una servidora es identificada mediante un identificador de conexión de enlace de datos *DLCI* (*Data Link Connection Identifier*), que contiene el número de canal del servidor que puede tener un valor entre 1 y 30.

[20] *ETSI* (*European Telecommunications Standards Institute*)

[21] El estándar TS 07.10 es usado por los teléfonos celulares *GSM* para multiplexar varios *streams* de datos sobre un cable serial físico. [60]

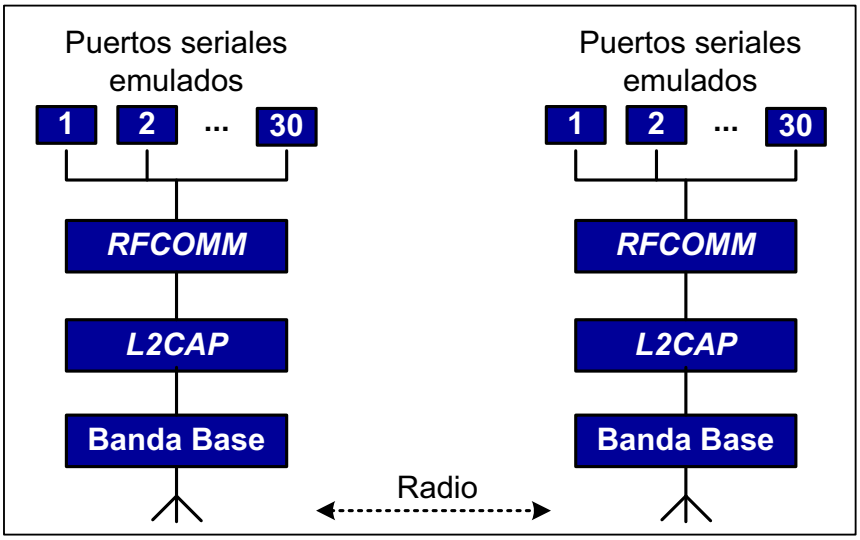


Figura 2.10 Esquema para múltiples puertos seriales emulados mediante *RFCOMM*[29]

Los dispositivos *Bluetooth* que ofrecen servicios soportados por *RFCOMM* deben tener una entrada en su base de datos de servicios, la cual proporciona información sobre cómo conectarse a través de *RFCOMM*.

La información mínima necesaria para conectarse a un servicio sobre *RFCOMM* es el nombre del servicio y el número de canal sobre el cual se va a realizar la transferencia de datos.

2.2.3. INTERFAZ *HCI* (*HOST CONTROLLER INTERFACE*)

La interfaz *HCI* es la línea divisoria entre el *software* y *hardware*. *L2CAP* y las capas sobre ella son implementadas actualmente en *software,* mientras que *LMP* y las capas inferiores son implementadas en *hardware*. *HCI* es la interfaz para el bus físico que conecta estos dos componentes. Permite disponer de una capa de acceso homogénea para todos los módulos *Bluetooth* de Banda Base, aunque sean de distintos fabricantes. [24] [66]

2.3. PERFILES *BLUETOOTH* [21][27][26]

Para garantizar la interoperabilidad entre productos y aplicaciones *Bluetooth* de diferentes fabricantes la especificación *Bluetooth* define un conjunto de perfiles.

Cada perfil representa un posible escenario de uso, en el que dos o más dispositivos con tecnología *Bluetooth* deben interactuar para proporcionar al usuario un determinado servicio. En cada perfil se definen los protocolos a utilizar y los procedimientos a seguir en distintos escenarios de aplicación.

Todos los dispositivos *Bluetooth* deben soportar el perfil *GAP* (*Generic Access Profile*), a partir de éste se derivan los demás perfiles.

El perfil utilizado por los dispositivos del sistema es el perfil *SPP* (*Serail Port Profile*) que permite establecer una conexión serial emulada entre dispositivos *Bluetooth*.

2.3.1. PERFIL *GAP* (*GENERIC ACCESS PROFILE*) [23]

En este perfil se involucrará principalmente el uso de las capas más bajas de la pila de protocolos de *Bluetooth* (Banda Base, *LMP* y *L2CAP*).

Adicionalmente este perfil especifica los términos generales que pueden ser usados sobre el nivel de interfaz de usuario:

- **Dirección del dispositivo *Bluetooth* (*BD_ADDR*):** Es una dirección única definida en el nivel de Banda Base. Es recibida por el dispositivo remoto durante el procedimiento de descubrimiento (*Inquiry*). En el nivel de interfaz de usuario la dirección *Bluetooth* debe ser representada por 12 caracteres hexadecimales los mismo que pueden estar subdivididos o no por el símbolo ":" como por ejemplo: 000C3E3A4B69 o 00:0C:3E:3A:4B:69.

- **Nombre del dispositivo *Bluetooth (friendly-name)* :** El perfil *GAP* permite que los dispositivos *Bluetooth* tengan nombres descriptivos de hasta 248 bytes de longitud, sin embargo su longitud se ve limitada también por la capacidad de visualización del dispositivo *Bluetooth*.

- ***PIN* (*Personal Identification Number*) *Bluetooth*:** Es usado para la autenticación de dos dispositivos *Bluetooth* que no hayan intercambiado previamente claves de enlace para establecer una relación de confianza. Se lo utiliza en el procedimiento de emparejamiento[22].

- **Clase de dispositivo:** Este parámetro es transmitido desde el dispositivo remoto durante el procedimiento de descubrimiento de dispositivos. Indica que tipo de dispositivo es y qué servicios soporta.

2.3.2. PERFIL *SPP* (*SERIAL PORT PROFILE*) [30][33]

Este perfil define los protocolos y procedimientos que se emplea para establecer una conexión serial emulada entre dispositivos *Bluetooth*.

En la Figura 2.11 se muestran los protocolos y entidades usados en este perfil.

El protocolo de transporte *RFCOMM* es utilizado para la emulación de puerto serie. La capa de emulación de puerto provee un *API* (Programa de Aplicación de Interfaz) para las aplicaciones que corren por encima. Las aplicaciones *A* y *B* representan las aplicaciones en diferentes dispositivos que se van a comunicar a través del puerto serie emulado.

[22] Emparejamiento: La primera vez que dos dispositivos intentan comunicarse, tiene lugar el procedimiento denominado emparejamiento que permite crear una clave de enlace común que será usada para posterior autenticación. Este procedimiento requiere que el usuario de cada dispositivo introduzca el *PIN* que debe ser el mismo en los dos casos.[23]

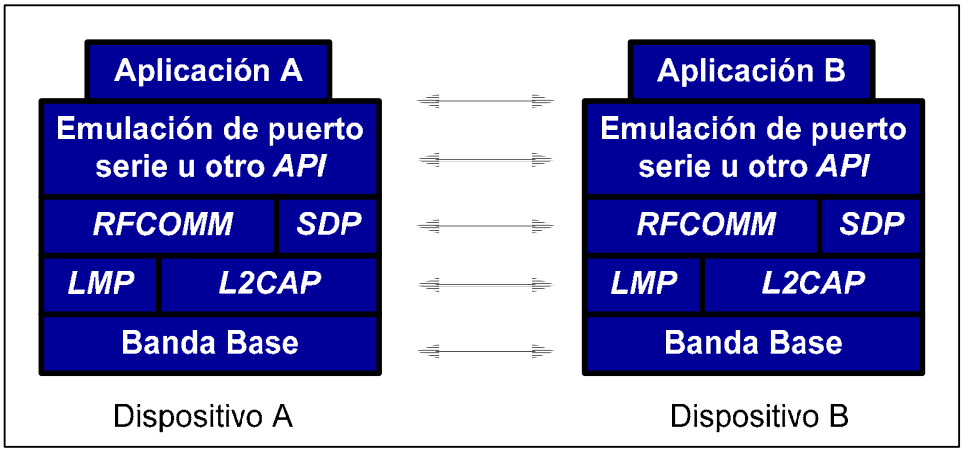


Figura 2.11 Protocolos y entidades del perfil *SPP* [33]

En este perfil un dispositivo puede desempeñar dos roles:

- **Iniciador:** Es el dispositivo que toma la iniciativa de establecer una conexión.

- **Aceptador:** Es el dispositivo que espera que otro dispositivo inicie el establecimiento de la conexión.

Una vez que el dispositivo iniciador comienza el establecimiento del enlace, se realizan procedimientos de búsqueda de servicios para establecer la conexión de cable serie emulado.

Algunas consideraciones que se deben tener encuentra en este perfil son:

- Este perfil requiere el soporte para paquetes de un solo slot y garantiza velocidades de datos de hasta 128 Kbps, el soporte para velocidades superiores es opcional.

- A nivel de *L2CAP* este perfil solamente emplea canales orientados a la conexión. Los datos son llevados sobre canales confiables lo que implica que

el temporizador de banda base *flush timeout* debe estar en su valor por defecto (infinito).

2.4. INTERFAZ *BLUETOOTH* DEL *NXT BRICK*

El brazo robot a ser utilizado en el presente proyecto es un robot *Lego Mindstorms NXT* que se muestra en la Figura 2.12. Los elementos de *hardware* que lo componen se pueden agrupar en cuatro categorías principales:

- Unidad central de control: *NXT Brick.*
- Dispositivos de salida: motores.
- Dispositivos de entrada: sensores.
- Medio de comunicación: *Bluetooth.*

A continuación se detallará la funcionalidad *Bluetooth* dentro de *NXT Brick* ,en el capítulo 3 se detallarán los demás componentes del brazo robot.

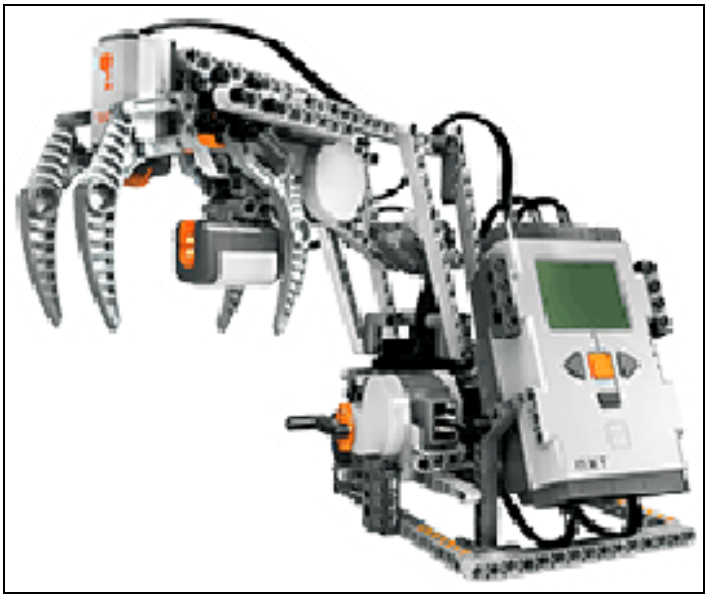


Figura 2.12 Brazo robot *Lego Mindstorms NXT*

2.4.1. FUNCIONALIDAD *BLUETOOTH* DENTRO DEL *NXT BRICK* [34][61]

El *NXT brick* soporta comunicación inalámbrica *Bluetooth* mediante el *chip CSR BlueCore* 4 *EXT*. Este *chip* implementa la especificación *Bluetooth* versión 2.0 con *EDR*, trabaja como un dispositivo *Bluetooth* de clase 2, lo que significa que puede comunicarse hasta una distancia máxima de 10 metros aproximadamente y emplea el perfil *SPP*.

La comunicación se la realiza mediante canales, donde un *NXT* dentro de la red debe funcionar como dispositivo maestro y los otros *NXT* se comunican a través de él como se muestra en la Figura 2.13.

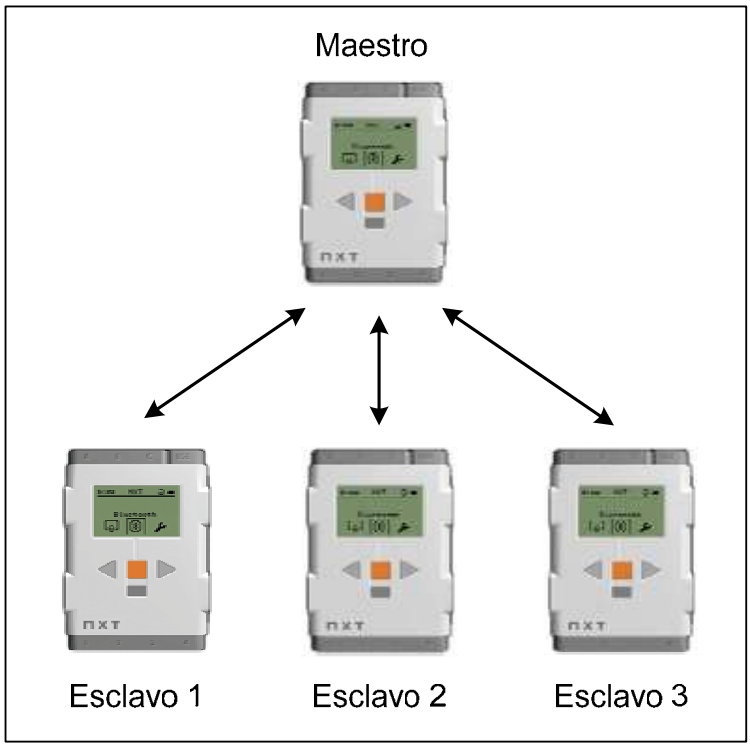


Figura 2.13 Comunicación entre 4 *NXTs* usando *Bluetooth* [34]

El *NXT brick* puede conectarse inalámbricamente con otros tres dispositivos al mismo tiempo pero solamente comunicarse con un dispositivo a la vez.

El *NXT* tiene cuatro canales para la comunicación *Bluetooth*. El canal 0 es siempre utilizado por un *NXT* esclavo para comunicarse con el *NXT* maestro y los canales 1, 2 y 3 son para la comunicación del dispositivo maestro hacia los diferentes esclavos.

Un *NXT* no es capaz de funcionar como maestro y esclavo al mismo tiempo debido a que ésto puede causar la pérdida de datos entre los dispositivos *NXT*. Esta funcionalidad ha sido deshabilitada dentro del *firmware* estándar *Lego Mindstorms NXT*.

CAPÍTULO 3: LA PLATAFORMA *J2ME* Y EL SISTEMA OPERATIVO *leJOS*

Tomando en cuenta los recursos con los que se cuenta en los dispositivos que van a formar parte del sistema dentro del proyecto, que son los teléfonos celulares *Nokia* 5220 y *Sony Ericsson W580,* y el Brazo Robot *Lego Mindstorms NXT*, se considera que *Java* es el lenguaje más adecuado para llevar a cabo el proyecto ya que todos los dispositivos dentro del sistema lo soportan y se ajusta a las necesidades y requerimientos. En este capítulo se detallará la plataforma *J2ME*[23] y el sistema operativo *leJOS*[24] utilizados para el desarrollo del sistema.

3.1. *J2ME* [39] [40]

J2ME (*Java 2 Micro Edition*) es una plataforma basada en el lenguaje *Java* que creó *Sun Microsystems,* orientada al desarrollo de aplicaciones para dispositivos con capacidades restringidas tanto en pantalla gráfica, como en procesamiento y memoria como son electrodomésticos inteligentes, teléfonos móviles, *PDA's*, etc.

J2ME mantiene las cualidades de la tecnología *Java*, algunas de sus características son: [45]

- **Portabilidad:** permite el desarrollo de aplicaciones que pueden correrse en distintos dispositivos con características similares. Proporciona independencia de plataforma respecto al *hardware* y al sistema operativo del dispositivo.

- **Accesibilidad:** provee de bases para desarrollar aplicaciones inteligentes y dinámicas en red.

[23] *J2ME* (*Java 2 Micro Edition*)
[24] *leJOS* (*Lego Java Operating System*)

- **Seguridad:** la máquina virtual de *Java*, al ejecutar el código *Java*, realiza comprobaciones de seguridad. Además el propio lenguaje carece de características inseguras, como por ejemplo los punteros.

- **Multihilos:** *Java* soporta la sincronización de múltiples hilos (*threads*) de ejecución a nivel de lenguaje, que son útiles en la creación de aplicaciones de red distribuidas.

3.1.1. COMPONENTES *J2ME* [37] [39]

J2ME está diseñada de forma modular y escalable considerando las diferencias en funcionamiento, modo de uso, configuración de *hardware* entre otras cosas que puede existir entre los dispositivos hacia los cuales está orientada.

Los elementos que la conforman se muestran en la Figura 3.1

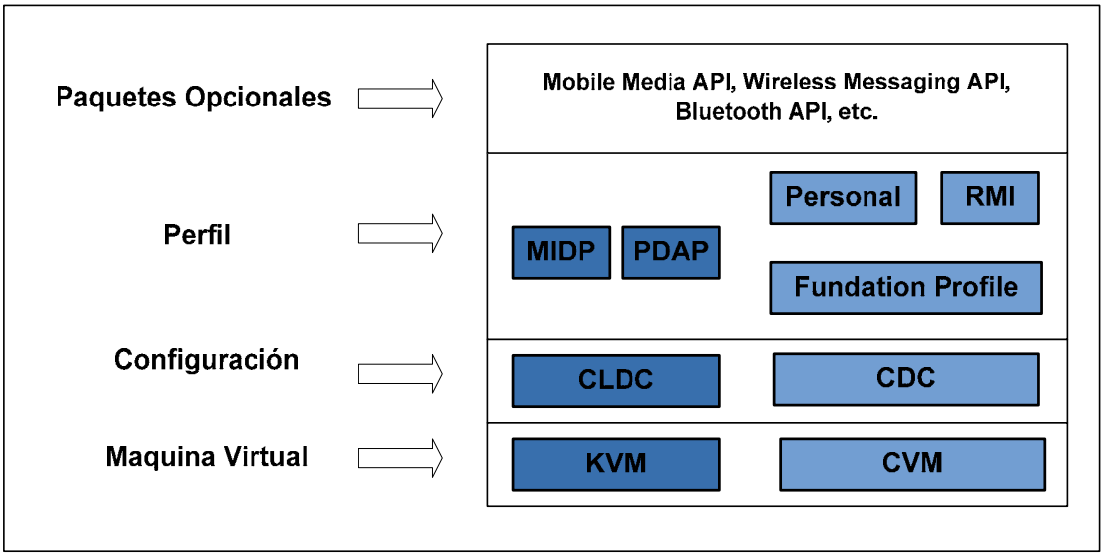


Figura 3.1 Componentes de *J2ME* [37]

3.1.1.1. Maquina Virtual

La máquina virtual de *Java* (*JVM*) es el programa encargado de interpretar código intermedio (*bytecode*) de los programas *Java* a un código de máquina ejecutable por la plataforma; además de efectuar las llamadas pertinentes al sistema operativo y verificar las reglas de seguridad y corrección de código definidas para el lenguaje *Java*. [37]

J2ME define diferentes *JVMs* dependiendo de las capacidades de los dispositivos sobre los cuales se va a implementar. Se tiene las siguientes:

- ***KVM (Kilo Virtual Machine):*** Es una implementación de maquina virtual reducida. Tiene una baja ocupación de memoria, entre 40 KBytes y 80 KBytes y está orientada a dispositivos con bajas capacidades computacionales y de memoria.

- ***CVM (Compact Virtual Machine)*:** Está orientada a dispositivos electrónicos con procesadores de 32 bits de gama alta. Tiene una ocupación de 2 MBytes o más de memoria *RAM*.

3.1.1.2. Configuraciones [43]

Una Configuración *J2ME* define una plataforma mínima para un grupo específico de dispositivos con similares características de memoria y procesamiento. Una configuración específica:

- Las características del lenguaje *Java* soportadas por el dispositivo.
- Las características soportadas por la máquina virtual *Java*.
- Las librerías *Java* soportadas.

Hay dos tipos de configuraciones *J2ME*: *CLDC* (*Connected Limited Device Configuration*) diseñada para dispositivos con restricciones de procesamiento y memoria considerables y *CDC* (*Connected Device Configuration*) para dispositivos con menos limitaciones.

3.1.1.2.1. CDC (Connected Device Configuration)

Definida en la especificación *JSR*[25]-36 versión *CDC* 1.0 y mejorada en la especificación *JSR*-218, versión *CDC* 1.1.

Esta configuración está orientada a dispositivos con cierta capacidad computacional y de memoria. Ejemplos de estos dispositivos son: consolas de juego, decodificadores de televisión digital, electrodomésticos, sistemas de navegación en automóviles, reproductores de audio digital, impresoras entre otros. Usa la máquina virtual *CDC*.

Se incluyen varios paquetes *Java* de la edición estándar (*J2SE*)[26] en *CDC* que se muestran en la Tabla 3.1.

Paquetes *CDC*	**Descripción**
java.io	Clases e interfaces estándar de *E/S.*
java.lang	Clases esenciales del lenguaje Java. El compilador lo importa automáticamente hacia todos los programas.
java.lang.ref	Clases que permiten la interacción entre un programa y el recolector de basura.
java.lang.reflect	Clases e interfaces de reflexión[27].
java.math	Paquete de matemáticas.

Tabla 3.1 Paquetes *Java* de la configuración *CDC* 1.1 [37]

[25] *JSR:* Son documentos oficiales que describen las especificaciones y tecnologías propuestas para añadirse en la plataforma *Java* [36]

[26] *J2SE (Java 2 Estándar Edition):* Orientada a equipos personales y estaciones de trabajo. Permite el desarrollo de *applets* (aplicaciones que se ejecutan en un navegador web) y aplicaciones independientes (*stand-alone*). [36]

[27] Reflexión: Es el mecanismo por el cual los objetos pueden obtener información de otros objetos en tiempo de ejecución como, por ejemplo, los archivos de clases cargados o sus campos y métodos. [37]

Paquetes *CDC*	Descripción
java.net	Clases e interfaces de red.
java.security	Clases e interfaces de seguridad.
java.security.cert	Clases de certificado de seguridad.
java.text	Paquete que permite dar formato especializado a fechas, números y mensajes.
java.util	Clases de utilidades estándar (como manipulaciones de fecha y hora, almacenamiento y procesamiento de colecciones de datos entre otras).
java.util.jar	Clases y utilidades para archivos *JAR.*
java.util.zp	Clases y utilidades para archivos *ZIP* y comprimidos.
javax.microedition.io	Clases e interfaces para conexión genérica en *CDC.*

Tabla 3.1 Paquetes *Java* de la configuración *CDC* 1.1 [37]. (Continuación)

3.1.1.2.2. CLDC (Connected Limited Device Configuration)[51][54]

Definida por la especificación *JSR*-30, versión *CLDC* 1.0 y mejorada por la especificación *JSR*-139, versión *CLDC* 1.1.

Esta configuración está orientada a dispositivos pequeños que cuentan con limitaciones en cuanto a capacidad gráfica, cómputo y memoria. Dentro de este grupo de dispositivos están los teléfonos celulares, buscapersonas (*pagers*), *PDA's* y organizadores personales.

Los dispositivos que emplean esta configuración tienen las siguientes características generales: [51]

- Procesador de 16 o 32 bits con al menos 25 MHz de velocidad.

- Disponer de al menos 192 KBytes de memoria total para la plataforma *Java*. Al menos 160 KBytes de memoria no volátil para la máquina virtual *Java* y las librerías *CLDC*, y al menos 32 KBytes de memoria volátil para la máquina virtual en tiempo de ejecución.

- Ofrecer bajo consumo, debido a que estos dispositivos trabajan con suministro de energía limitado, generalmente baterías.

- Tener conexión de red, normalmente inalámbrica, con conexión intermitente y ancho de banda limitado.

Esta configuración usa la máquina virtual *KVM*.

El *API (Application Programming Interface)*[28] *CLDC* es un subconjunto de los paquetes de *J2SE*: *java.lang*, *java.io* y *java.util*, al que se incorpora el paquete específico *javax.microeditio.io*.

El paquete *javax.microedition.io* contiene las clases e interfaces de conexión genérica en *CLDC* dan soporte para el trabajo en red y comunicaciones en las aplicaciones.

3.1.1.3. Perfiles

Un perfil es un conjunto de *APIs* más específico que extiende las capacidades de una configuración para completar su funcionalidad y el uso de los elementos que proporciona un dispositivo. [43]

Los *APIs* del perfil están orientados a un ámbito de aplicación determinado y permiten la administración de la aplicación, de la interfaz gráfica de usuario, de las redes y de la transmisión de datos.

Un perfil siempre se construye sobre una configuración por lo que se tienen perfiles sobre: la configuración *CDC* y la configuración *CLDC* como se puede ver en la Figura 3.1

[28] *API* de Java: Provee de un conjunto de clases utilitarias para efectuar toda clase de tareas necesarias dentro de un programa. El *API* se encuentra organizado en paquetes lógicos. [69]

El perfil utilizado para los teléfonos celulares es el perfil *MIDP* que se detallará a continuación.

3.1.1.3.1. Perfil MIDP (Mobile Information Device Profile) [52]

Definido por la especificación *JSR*-37, versión *MIDP* 1.0 y mejorado en la especificación *JSR*-118, versión *MIDP* 2.0.

Este perfil se construye sobre la configuración *CLDC* y define un entorno *Java* para teléfonos celulares, buscapersonas y dispositivos similares con recursos limitados.
Los dispositivos que implementan este perfil deben cumplir los siguientes requerimientos mínimos: [52]

- Poseer un *display* de al menos 96x54 pixeles.
- Contar con una o más entradas de datos ya sea un teclado o una pantalla táctil.
- Tener acceso a una red inalámbrica con un ancho de banda mínimo de 9600 bps.
- Se debe tener al menos 256 KBytes de memoria no volátil para la implementación *MIDP* a más de la memoria requerida por la configuración *CLDC*.
- Tener 8 KBytes de memoria no volátil para que las aplicaciones puedan guardar datos.
- Tener 128 KBytes de memoria volátil para la ejecución de la aplicación.
- Tener capacidad de reproducir tonos ya sea vía *hardware* o con un algoritmo *software*.

Las aplicaciones que se realizan utilizando este perfil toman el nombre de *MIDlets*, se los detallará en el apartado 3.1.3.

En la Tabla 3.2 se muestran los paquetes que forman parte del perfil *MIDP*

Paquetes *MIDP*	Descripción
javax.microedition.midlet	Define las aplicaciones *MIDP* (*MIDlets*) y la interacción entre la aplicación y el entorno sobre el cual se está ejecutando la aplicación.
javax.microedition.lcdui	Provee un conjunto de características para la implementación de interfaces de usuario para aplicaciones *MIDP.*
javax.microedition.lcdui.game	Proporciona un conjunto de clases para construir juegos ricos en contenidos para dispositivos inalámbricos.
javax.microedition.io	Clases e interfaces de conexión genérica, dan soporte para el trabajo en red y comunicaciones en las aplicaciones.
javax.microedition.media	Extiende la funcionalidad de *J2ME* proporcionando audio, video y otras características multimedia.
javax.microedition.media.control	Define los tipos de control específicos que pueden ser usados por un reproductor.
javax.microedition.rms	Proporciona un mecanismo para que los *MIDlets* continuamente puedan guardar datos y posteriormente puedan recuperarlos.
javax.microedition.pki	Permite el uso de certificados para autenticar la información proveniente de conexiones seguras.
java.io	Provee clases e interfaces de *E/S* básicas.
java.lang	Provee clases e interfaces básicas del lenguaje *Java.*
java.util	Provee clases e interfaces de utilidades estándar.

Tabla 3.2 Paquetes del perfil *MIDP* versión 2.0 [52]

3.1.1.4. Paquetes Opcionales

Los paquetes opcionales son *APIs* que se pueden incluir para agregarle funcionalidades a los perfiles.

Algunos ejemplos de estos paquetes opcionales son:

- ***Mobile Media API*:** es un *API* multimedia para *J2ME*. Permite el acceso y control de audio y recursos multimedia.

- ***Wireless Messaging API*:** permite el envió, recepción y gestión de los *SMS* desde los *MIDlets*.

- ***Bluetooth API*:** permite crear aplicaciones para controlar el *Bluetooth* del dispositivo móvil. Este *API* será usado en el presente proyecto para establecer la comunicación *Bluetooth* y será detallado a continuación.

3.1.2. *APIS* DE *JAVA* PARA *BLUETOOTH* , *JSR*-82 [53]

Las librerías necesarias para la utilización de *Bluetooth* en cualquier dispositivo que implemente la plataforma *J2ME* están incluidas en la especificación *JSR*-82.

El *API* de *JSR*-82 está enfocado a las siguientes áreas:

- Transmisión de datos únicamente (las comunicaciones de voz no están soportadas).
- Permite el uso de los protocolos: *L2CAP* (únicamente orientado a la conexión), *RFCOMM*, *SDP* y *OBEX*.
- Permite el uso de los perfiles: *GAP*, *SDAP*[29], *SPP* y *GOEP*[30].

Los paquetes que forman parte de esta especificación son 2: *javax.bluetooth* y *javax.obex* los mismos que dependen del paquete *CLDC javax.microedition.io.*

En el presente proyecto se utilizará el paquete *javax.bluetooth* que es el que permite establecer la comunicación *Bluetooth* entre dispositivos empleando el protocolo *RFCOMM* a través del perfil *SPP* por lo que a continuación se realizará su estudio.

[29] *SDAP (Service Discovery Application Profile)*
[30] *GOEP (Generic Object Exchange Profile)*

3.1.2.1. *API javax.bluetooth* [19] [53] [56]

Este *API* contiene las clases y métodos necesarios para establecer una comunicación *Bluetooth* entre dispositivos. Permite establecer conexiones a nivel del protocolo *L2CAP* y el protocolo *RFCOMM*.
Como se dijo anteriormente en el presente proyecto se empleará el protocolo *RFCOMM* por lo que el estudio de este *API* se centrará en las clases y métodos relacionados a éstos.

Se debe considerar que en este tipo de comunicación un dispositivo debe actuar como cliente y otro como servidor. Los dispositivos que interviene en el sistema a implementarse son un teléfono celular que actuará como cliente y un brazo robot que actuará como servidor.

Las acciones que se deben llevar a cabo en el dispositivo cliente para poder comunicarse con el servidor son:

- Búsqueda de dispositivos.
- Establecimiento de la conexión.
- Enviar y recibir datos.

Mientras que en el servidor se deben llevar a cabo las siguientes acciones:

- Crear una conexión servidora.
- Aceptar conexiones de los clientes.
- Enviar y recibir datos.

Para llevar a cabo las acciones antes mencionadas se disponen de diferentes clases que se detallarán a continuación.

3.1.2.1.1. Clases Básicas para administración de dispositivos

Se tienen clases que representan a los objetos *Bluetooth* esenciales, como son *LocalDevice* y *RemoteDevice*. Además se tiene las clases *DeviceClass* y *BluetoothStateException* que dan soporte a la clase *LocalDevice*.

- **Clase *LocalDevice*:** Proporciona acceso y control sobre el dispositivo *Bluetooth* local. Permite obtener información del mismo como por ejemplo el modo de conectividad, la dirección *Bluetooth*, y el nombre del dispositivo o *friendly-name*. Esta clase es el punto de partida para las acciones que se vayan a llevar a cabo mediante este *API*. Los métodos principales de esta clase son :

    - ***LocalDevice getLocalDevice():*** obtiene el objeto que representa al dispositivo local.

    - ***String getAddress():*** obtienen la dirección *Bluetooth* del dispositivo local.

    - ***boolean setDiscoverable(int mode):*** establece el modo de detección del dispositivo. Los modos que se pueden establecer mediante el parámetro de entrada *mode* se muestran en la Tabla 3.3.

    - ***DiscoveryAgent getDiscoveryAgent():*** obtiene el objeto *Discovery Agent* para el dispositivo local.

- **Clase *RemoteDevice*:** De forma similar, esta clase representa al dispositivo *Bluetooth* remoto y permite obtener información como el nombre, la dirección *Bluetooth* del dispositivo remoto y otros términos asociados a la conexión. Algunos de los métodos de esta clase son:

- ***RemoteDevice getRemoteDevice (javax.microedition.io. Connection con):*** obtiene el objeto que representa al dispositivo *Bluetooth* remoto asociado a la conexión correspondiente. El parámetro de entrada *conn* representa a dicha conexión.

- ***String getBluetoothAddress():*** obtiene la dirección *Bluetooth* del dispositivo remoto.

- ***String getFriendlyName():*** obtiene el nombre del dispositivo *Bluetooth* remoto.

- **Clase *DeviceClass*:** a través de esta clase se puede obtener la clase de dispositivo. Por ejemplo se conoce si el dispositivo es un teléfono, un ordenador u otra clase de dispositivo.

- **Clase *BluetoothStateException()*:** representa la excepción que se produce cuando un dispositivo no puede cumplir con una solicitud que normalmente si la soportaría debido al estado actual del sistema. Por ejemplo algunos dispositivos no permiten realizar el proceso de *inquiry* cuando el dispositivo se encuentra conectado a otro dispositivo.

Nombre de la constante	Descripción	Valor
DiscoveryAgent.GIAC	Pone a los dispositivos en un modo detectable general.	10390323
DiscoveryAgent.LIAC	Pone a los dispositivos en un modo detectable limitado. Son visibles ante otros dispositivos durante un periodo de tiempo limitado.	10390242
DiscoveryAgent. NOT_DISCOVERABLE	En este modo los dispositivos no podrán ser descubiertos por otros.	0

Tabla 3.3 Modos para establecer el tipo de búsqueda [56]

3.1.2.1.2. Búsqueda de dispositivos

La búsqueda de dispositivos es realizada por el equipo Cliente para poder detectar el equipo al que desea conectarse.
Para poder realizar la búsqueda de dispositivos es necesario crear un único objeto *DiscoveryAgent* de la clase *DiscoveryAgent*. Este objeto se lo puede obtener a partir del método *getDiscoveryAgent()* del objeto *LocalDevice* como se ve en la Figura 3.2

```
LocalDevice equipo = LocalDevice.getLocalDevice();

DiscoveryAgent discoveryAgent = equipo.getDiscoveryAgent();

DiscoveryAgent discoveryAgent =
LocalDevice.getLocalDevice().getDiscoveryAgent();
```

Figura 3.2 Ejemplo para obtener el objeto *DiscoveryAgent* [19]

Los métodos de la clase *DiscoveryAgent* que van a permitir realizar y cancelar búsquedas de dispositivos son:

- ***boolean startInquiry (int accessCode, DiscoverListener listener):*** permite iniciar el proceso de búsqueda de dispositivos (proceso de *Inquiry*). Este método requiere que se implemente la interfaz *DiscoveyListener*, esta interfaz será notificada cada vez que un nuevo dispositivo sea encontrado o el proceso de búsqueda haya terminado.

  Este método requiere dos argumentos de entrada. El primer argumento es un valor entero que representa el modo de conectividad que se ha definido para la búsqueda. Podría tomar los valores de las constantes *DiscoveryAgent.GIAC* o *DiscoveryAgent.LIAC* que se muestran en la Tabla 3.3. Generalmente se usa *DiscoveryAgent.GIAC* para realizar una búsqueda general. El segundo

argumento es un objeto que implemente la interfaz *DiscoveryListener* a través del cual se notificarán los dispositivos que se vaya descubriendo.

- ***boolean cancelInquiry (DiscoveryListener listener):*** este método cancela la búsqueda de dispositivos que se encuentre en proceso.

- ***RemoteDevice[ ] retriveDevices (int option):*** este método se usa para obtener listas de dispositivos ya conocidos. Devuelve un arreglo de objetos *RemoteDevice*. El argumento de entrada *option* puede tomar los valores que se muestran en la Tabla 3.4.

Nombre de la constante	Descripción	Valor
DiscoveryAgent.CACHED	Recupera los dispositivos que fueron descubiertos mediante un proceso de *inquiry* previo.	0
DiscoveryAgent.PREKNOWN	Recupera los dispositivos con los cuales el dispositivos local interactúa frecuentemente y que han sido agregados como pre-conocidos.	1

Tabla 3.4 Valores que puede tomar el argumento de entrada del método *retriveDevices* [56]

Adicionalmente es necesario el uso de la interfaz *DiscoveryListener.* Esta interfaz permite que el dispositivo notifique eventos a la aplicación cada vez que se descubre un dispositivo, un servicio o se finalice una búsqueda.

Los métodos de esta interfaz que son llamados en el proceso de búsqueda de dispositivos son:

- ***void deviceDiscovered(RemoteDevice equipo, DeviceClass dc):*** este método es llamado cada vez que un dispositivo ha sido encontrado y pasa dos argumentos. El primero es un objeto *RemoteDevice* que representa al

dispositivo *Bluetooth* encontrado, y el segundo es un objeto *DeviceClass* que proporciona la clase de dispositivo *Bluetooth* encontrado.

- ***void inquiryCompleted(int tipo):*** este método es llamado una vez que se ha finalizado la búsqueda de dispositivos y pasa como argumento un entero que indica el motivo de la finalización. Este entero puede tomar los valores de la Tabla 3.5.

Nombre de la constante	Descripción	Valor
DiscoveryListener.INQUIRY_COMPLETED	Indica que la búsqueda ha terminado con normalidad .	0
DiscoveryListener.INQUIRY_TERMINATED	Indica que la búsqueda ha sido cancelada manualmente.	5
DiscoveryListener.INQUIRY_ERROR	Indica que se ha producido un error en el proceso de búsqueda.	7

Tabla 3.5 Valores que puede tomar el argumento de entrada del método *inquiryCompleted()* [56]

3.1.2.1.3. Comunicación

Una vez que la aplicación *Bluetooth* haya realizado la búsqueda de dispositivos, se procede a realizar el establecimiento de la conexión con el dispositivo seleccionado, para posteriormente poder realizar la transmisión de datos.

Para establecer una conexión utilizando el protocolo *RFCOMM* mediante el perfil *SPP* se deben seguir diferentes pasos dependiendo si el dispositivo es cliente o servidor.

- Servidor *SPP*

  Para crear una conexión se hace uso de la clase *Connector* perteneciente al paquete *javax.microedition.io*. Esta clase permite abrir una conexión mediante

el método *Connector.open(),* el cual recibe un argumento de entrada del tipo *String* que contiene un *URL* (*Uniform Resourse Locator).*
Este *URL* debe contener el protocolo, la dirección *Bluetooth*, el *UUID* (identificador de servicio), y parámetros de seguridad opcionales. Así el formato para el *URL* del dispositivo servidor es el siguiente:

btspp://localhost:UUID;parametros

En la Tabla 3.6 se muestran los diferentes parámetros que se pueden usar en el URL.

Nombre	Descripción	Valores permitidos	Cliente o Servidor
master	Especifica si el dispositivo debe ser el maestro de la conexión.	*true, false*	Ambos
authenticate	Especifica si el dispositivo remoto debe ser autenticado antes de establecer la conexión.	*true, false*	Ambos
encrypt	Especifica si el enlace debe ser encriptado.	*true, false*	Ambos
authorize	Especifica si todas las conexiones hacia el dispositivo deben recibir una autorización para usar el servicio.	*true, false*	Servidor
name	Especifica el nombre de servicio.	Cualquier *String* válido	Servidor

Tabla 3.6 Parámetros que se pueden usar en el *URL* de conexión [19]

El método *Connector.open()* devuelve un objeto *StreamConnection-Notifier* (interface del paquete *javax.microedition.io*) para la conexión servidora. Este objeto es un notificador que permitirá escuchar las conexiones entrantes de los clientes a través del método *acceptAndOpen().*

El método *acceptAndOpen()* retorna un objeto *StreanConnection* (interfaz del *paquete javax.microedition.io*) que permite obtener y abrir los flujos (*streams*) de entrada y salida de datos a través de los métodos *openDataInputStream()* y

openDataOutputStream() respectivamente. Adicionalmente tiene el método *close()* que permite cerrar la conexión.

Para enviar y leer los datos se puede emplear los métodos *read()* y *write()* de la clase *DataInputStream* y *DataOutputStream* respectivamente pertenecientes al *paquete java.io*.

En la Figura 3.3 se muestra un ejemplo de la creación de una conexión para un Servidor *SPP.*

```
StreamConnectionNotifier notificador =
(StreamConnectionNotifier)Connector.open("btspp://
localhost:123456789ABCDE;name=Servidor");
StreamConnection conn = notificador.acceptAndOpen();
DataOutputStream salida = conn.openOutputStream();
DataInputStream entrada = conn.openInputStream();
//lee los datos recibidos
byte datosRecividos[ ] = new byte;
entrada.read(datosRecividos);
//Se envía datos através del flujo de salida
salida.write(dato);
// Se cierra la conexión y los flujos de entrad y salida
entrada.close();
salida.close();
con.close();
```

Figura 3.3 Ejemplo de la creación de una conexión para un Servidor *SPP* [19]

- Cliente *SPP*

Al igual que en el servidor se emplea el método *open()* de la clase *Connector* para abrir una conexión. La diferencia está en los campos del *URL*. El formato para el *URL* del dispositivo cliente es el siguiente:

btspp://direcciónBluetooth:CN;Parámetros

Donde:

- *direcciónBluetooth:* es la dirección del dispositivo con el cual se quiere conectar.
- *CN:* es el número de canal usado por el cliente para conectarse con el servidor.
- *Parámetros:* representa los parámetros adicionales que se pueden usar. Éstos se muestran en la Tabla 3.6.

Un ejemplo de un *URL* para un dispositivo cliente es:

btspp://008003DD8901:1;master=false; encrypt=false

El método *Connector.open()* devuelve el objeto *StreamConnection* que representa la conexión. Al igual que en el servidor a través de este objeto se puede obtener los flujos de datos de entrada y salida para el envío y recepción de datos como se muestra en el ejemplo de la Figura 3.4.

```
String URL = "008003DD8901:1;master=false; encrypt=false";
StreamConnection conn =
(StreanConnection)Connector.open(URL)
DataOutputStream salida = conn.openDataOutputStream();
DataInputStream entrada = conn.openDataInputStream();
```

Figura 3.4 Ejemplo de la creación de una conexión para un Cliente *SPP*[19]

3.1.3. *MIDLET* [39] [35]

Un *MIDlet* es una aplicación *Java* realizada con el perfil *MIDP* sobre la configuración *CLDC*. Extiende la clase *javax.microedition.MIDlet* contenida en el *API MIPD.*

Gracias a la filosofía de *Java* (*write one, run anywhere*) los *MIDlet* se los puede ejecutar en una amplia gama de dispositivos sin tener que realizar modificaciones. Los dispositivos sobre los cuales se va a ejecutar el *MIDlet* deben contar con un módulo *software* encargado de la gestión de los *MIDlets* (cargarlos, ejecutarlos, administrar sus estados, etc.). Este *software* es llamado *AMS (Application Management System).*

Para la programación y compilación de los *MIDlets* se cuenta con varios entornos de desarrollo *Java* como por ejemplo el *IDE (Integrated Development Environment) NetBeans*[31] que será usado en el presente proyecto. Este *IDE* permite además descargar directamente la aplicación en los teléfonos celulares *Nokia* ya que tiene una conexión con el *software Nokia PC Suit*[32].

Adicionalmente se cuenta con diferentes simuladores para teléfonos móviles que permiten realizar las pruebas de la aplicación. Un ejemplo es la plataforma *Nokia SDK (Software Development Kit)* para la serie 40 que será usado en el presente proyecto. Este simulador cuenta con un *NCF* (*Nokia Conectivity Framework*) que permite la simulación de la comunicación inalámbrica *Bluetooth*.

Todo el *software* antes mencionado es de libre distribución.

---

[31] *NetBeans* es una plataforma para el desarrollo de aplicaciones *Java*, permite a los programadores editar, compilar, depurar y ejecutar programas. [36]

[32] *Nokia PC Suit*: *Software Nokia* que permite la conexión entre el *PC* y el teléfono móvil *Nokia*, permite realizar tareas como la descarga de aplicaciones para el teléfono, transferir archivos entre teléfono y *PC*, ver archivos y carpetas del teléfono en el *PC* entre otras. [68]

3.1.3.1. Ciclo de vida de un *Midlet*

Un *MIDlet* pasa por 3 estados a lo largo de su ejecución, desde su inicio hasta su finalización. Estos estados son:

- **Estado de Espera:** Un *MIDlet* se encuentra en este estado cuando es interrumpido por el gestor de aplicaciones y también durante un breve periodo de tiempo al inicio de la aplicación.

- **Estado Activo:** Un *MIDlet* se encuentra en este estado durante su ejecución normal.

- **Estado Destruido:** Cuando un *MIDlet* termina su ejecución pasa a este estado.

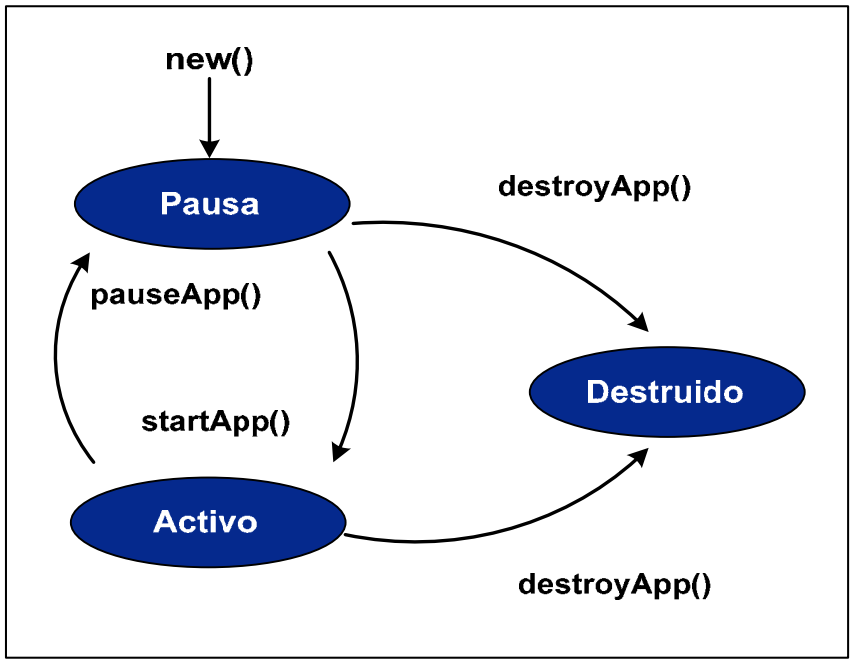


Figura 3.5 Ciclo de vida de un *MIDlet* [39]

Los cambios de estado de un *MIDlet* son realizados por el gestor de aplicaciones. Los *MIDlets* pueden solicitar un cambio de estado al gestor de aplicaciones mediante los siguientes métodos de la clase *javax.microedition.MIDlet*:

- ***starApp():*** llamada previa para pasar al estado activo.
- ***pauseApp():*** llamada previa para pasar al estado de pausa.
- ***destroyApp():*** llamada previa a la finalización de la ejecución.
- ***notifyDestroy():*** solicita al gestor de aplicaciones pasar al estado destruido.
- ***notifyPaused():*** solicita al gestor de aplicaciones pasar al estado de espera.
- ***resumeRequest():*** solicita al gestor de aplicaciones volver al estado activo.

En la Figura 3.6 se muestra la estructura de un *MIDlet*

```
import javax.microedition.midlet.*
public class ejemploMidlet extends MIDlet {
public ejemploMidlet () {
/* Constructor de clase. Se deben inicializar las
variables. */
}
public startApp(){
/* Aquíse incluye  el código que se quiere que el MIDlet
ejecute cuándo se active. */
}
public pauseApp(){
/* Aquí se incluye el código que se quiere que el MIDlet
ejecute cuándo entre en el estado de pausa (Opcional). */
}
public destroyApp(){
/* Aquí se incluye el código que se quiere que el MIDlet
ejecute cuándo sea destruido. Generalmente  aquí se liberaran
los recursos ocupados por el MIDlet como memoria, etc.
(Opcional) */
}
}
```

Figura 3.6 Estructura de un *MIDlet* [35]

3.1.4. INTERFAZ DE USUARIO [40] [39] [55]

Para la creación de interfaces de usuario en el perfil *MIDP* se tiene el paquete *javax.microedition.lcdui* (Interfaz de usuario con pantalla *LCD*). Este paquete está compuesto por un *API* de alto nivel y un *API* de bajo nivel.

El *API* de alto nivel permite crear interfaces de usuario orientadas a aplicaciones de negocio, donde el *MIDlet* actuaría como cliente de una aplicación servidora. Incluyen un conjunto de elementos genéricos (formularios, etiquetas, botones, etc.) que son implementados por cada dispositivo. Al usar estos elementos se pierde el control del aspecto de la aplicación ya que la estética de los componentes depende exclusivamente del dispositivo donde se ejecute. Pero se tiene un alto grado de portabilidad de la aplicación entre diferentes dispositivos.

El API de bajo nivel permite crear interfaces de usuario que están orientadas principalmente al desarrollo de juegos, donde se requiere un control total de lo que aparece en la pantalla y de los mecanismos de entrada de bajo nivel del dispositivo (teclas, coordenadas del cursor, etc.). En este tipo de interfaces se tiene un nivel de portabilidad más limitado.

Las clases del paquete *javax.microedition.lcdui* se muestran en la Tabla 3.7.

Clase	Descripción
Display	Representa el controlador de la pantalla y dispositivos de entrada del sistema en un dispositivo móvil.
Displayable	Representa a objetos con capacidad de ser visibles en pantalla.
Screen	Superclase de todas las clases que conforman la interfaz de usuario de alto nivel.
TextBox	Implementa una pantalla que permite introducir y editar texto.
List	Implementa una pantalla con una lista de opciones seleccionables por el usuario.

Tabla 3.7 Clases del paquete *javax.microedition.lcdui* [43].

Clase	Descripción
Alert	Representa una alerta (pantalla de aviso).
AlertType	Representa diferentes tipos de alertas usadas junto con la clases *Alert.*
Form	Pantalla que sirve como contenedor para otros componentes gráficos de usuario.
Item	Elemento genérico de una *Form.*
ChoiceGroup	Grupo de elementos seleccionables que puede ser añadido a una *Form.*
DataField	Componente editable que representa información de fecha y hora que puede ser añadido en una *Form.*
TextField	Componente que permite añadir texto editable a una *Form.*
Gauge	Componente de una *Form* con forma de diagrama de barras para representar un valor numérico.
Image	Encapsula los datos de una imagen gráfica.
ImageItem	Elemento que soporta la presentación de una imagen.
StringItem	Representa un componente que contiene una cadena del tipo *String.*
Canvas	Pantalla genérica para el *API* de bajo nivel.
Graphics	Proporciona herramientas para dibujar dentro de una pantalla del tipo *Canvas.*
Command	Representa una acción realizada sobre la interfaz por el usuario.
Ticket	Texto deslizante disponible en todas las pantallas del *API* de alto nivel.
Font	Representa un tipo de letra y su métrica.

Tabla 3.7 Clases del paquete *javax.microedition.lcdui* [43]. (Continuación)

En la Figura 3.7 se muestra la relación y la jerarquía existente entre las clases de la Tabla 3.7.

Para la aplicación en el teléfono celular se utilizará una interfaz de usuario de alto nivel, a continuación se describirá las clases e interfaces necesarias para su implementación.

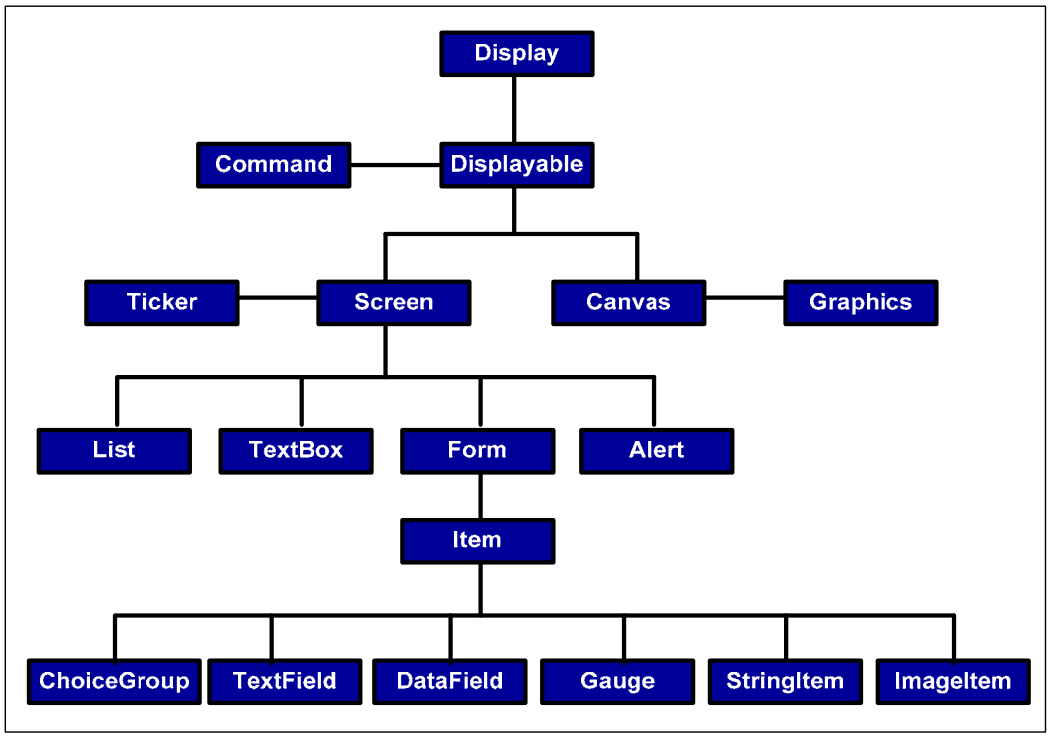


Figura 3.7 Jerarquía de clases de interfaz de usuario *MIDP* [35]

3.1.4.1. Clase *Display*

Esta clase es responsable de controlar la pantalla y la interacción con el usuario. Permite obtener información sobre las características de la pantalla del dispositivo, además de mostrar los objetos que forman parte de la interfaz.

Algunos de los métodos de esta clase se muestran en la Tabla 3.8.

Cada *MIDlet* tiene una sola instancia de la clase *Display*, se la puede obtener de la siguiente manera:

Display pantalla = Display.getDisplay(this)

Método	Descripción
Displayable getCurrent()	Devuelve la pantalla actual.
Display getDisplay(MIDLet m)	Devuelve una referencia del objeto *Display* que es único para el *MIDlet m.*
void setCurrent (Displayable d)	Establece como visible la pantalla *d.*
boolean isColor()	Devuelve *true* o *false* si la pantalla es de color o *b/n.*
int numColors()	Devuelve el número de colores aceptado por el dispositivo.

Tabla 3.8 Algunos métodos de la clase *Display* [55]

3.1.4.2. Clase *Displayable*

La clase *Displayable* representa a las pantallas de la aplicación. Cada objeto *Display* puede tener varios objetos *Displayable* pero solo un objeto *Displayable* puede estar visible a la vez. Para hacer visible un *Displayable* se usa el método *setCurrent()* de la clase *Display*.

Cada objeto *Displayble* puede tener un título, un *ticket* y un conjunto de comandos añadidos a él, a demás permite manejar eventos de pantalla. Los métodos que permiten realizar estas tareas se muestran en la Tabla 3.9

Método	Descripción
void addCommand(Command cmd)	Añade un comando.
int getHeight()	Devuelve el alto en pixeles de la pantalla.
Ticker getTicker()	Devuelve el *Ticket* asignado a la pantalla.
String getTitle()	Obtiene el título de la pantalla.
int getWidth()	Obtiene el ancho en pixeles de la pantalla.
boolean isShown()	Devuelve *true* si la pantalla está activa, visible.
void removeCommand(Command cmd)	Elimina un comando.
void setCommandListener(CommandListener l)	Establece un *listener* para la captura de eventos.
void setTcker(Ticker ticker)	Establece un *Ticker* a la pantalla.
void setTitle(String titulo)	Establece un título a la pantalla.
protected void sizeChanged(int w, int h)	Es llamado cuando el área disponible para el objeto *Displayable* ha cambiado.

Tabla 3.9 Métodos de la clase *Displayable* [55]

Se tiene dos elementos de tipo *Displayable*: *Screen* y *Canvas*. El objeto *Screen* se usa para las interfaces de alto nivel, mientras que el objeto *Canvas* para las interfaces de bajo nivel.

3.1.4.3. Clase *Command*

Un objeto de la clase *Command* contiene la información sobre un evento que se genera en la aplicación, como pulsar un botón; permite la interacción con el usuario.

El constructor de esta clase es el siguiente:

Command (String label, int commandType, int priority)

Donde:

- *label:* corresponde al rótulo visual que aparece en la pantalla del dispositivo para identificar el comando.

- *commandType:* indica el tipo de comando que se quiere crear. Los tipos que se pueden asignar se muestran en la Tabla 3.10.

- *priority:* define la importancia del comando con respecto a los demás comandos en la pantalla. A mayor número menor prioridad.

Adicionalmente se debe implementar la interfaz *CommandListener* que permitirá monitorear los eventos generados por un *Command*. Para registrar un *CommandListener* en el *Displayable* se emplea el método:

void setListener (CommandListener listener)

Adicionalmente la interfaz proporciona el método:

void commandAction(Command c, Displayable d)

Donde se indicará la acción que se quiere realizar cuando se produzca un evento en el *Command c* que se encuentra en el objeto *Displayable d*.

Comando	Descripción
BACK	Petición para volver a la pantalla anterior.
CANCEL	Petición para cancelar la acción en curso.
EXIT	Petición para salir de la aplicación.
HELP	Petición para mostrar información de ayuda.
ITEM	Petición para introducir el comando en un *item* en la pantalla.
OK	Aceptación de una acción por parte del usuario.
SCREEN	Se utiliza para comandos de propósito mas general.
STOP	Petición para parar una operación.

Tabla 3.10 Tipo de comandos [37]

3.1.4.4. Clase *Screen*

La clase *Screen* agrupa todas las clases que conforman la interfaz de usuario de alto nivel y hereda todos los métodos de la clase *Displayable*. Hay cuatro tipos de subclases de la clase *Screen*: *Alert*, *Form*, *List* y *TexBox*.

3.1.4.4.1. Clase Alert

Esta clase representa una alerta que es mostrada durante un tiempo o hasta que el usuario seleccione un comando. Se tienen los siguientes tipos de alertas: *ALARM, CONFIRMATION, ERROR, INFO, WARNING.*

El constructor de esta clase es el siguiente:

Alert (String title, String alertText, Image alertImage, AlertType alertType)

Donde:

- *title: es* el título que va a llevar la alerta.
- *alertText: es* el texto que se va a desplegar en la alerta.
- *alertImage: es* la imagen que se va a mostrar en la alerta.
- *alertType: especifica* el tipo de alerta que se va a crear.

3.1.4.4.2. Clase Form

Una *Form* es una pantalla que actúa como contenedor para una serie de elementos visuales que permiten construir interfaces más elaboradas. Todos los objetos que se pueden añadir se derivan de la clase *Item* y son:

- ***StringItem:*** representa un componente que contiene una cadena de texto de tipo *String*.
- ***ImageItem:*** elemento que soporta la presentación de una imagen.
- ***TextField****:* componente que permite añadir texto editable.
- ***DataField:*** componente editable que representa información de fecha y hora.
- ***ChoiceGroup:*** grupo de elementos seleccionables.
- ***Gauge:*** componente con forma de diagrama de barras para representar un valor numérico.

Los métodos que permiten añadir, eliminar y modificar elementos en la *Form* se muestran en la Tabla 3.11.

Los *Items* pueden producir eventos cuando los usuarios cambian su valor. Para que la aplicación pueda escuchar estos eventos se debe registra un *ItemStateListener* en la *Form* para esto se puede usar el método:

void setItemStateListener (ItemStateListener listener)

El *ItemStateListener* es una interfaz con un solo método. Este método es llamado cada vez que un *Item* ha cambiado dentro de la *Form*:

void itemStateChanged(Item item)

Método	Descripción
int append(Image imagen)	Añade una imagen.
int append(Item item)	Añade un *Item.*
int append(String texto)	Añade una cadena de texto.
void delete(int num)	Elimina el *Item* que ocupa la posición *num.*
void deleteAll()	Elimina todos los *Items.*
Item get(int num)	Devuelve el *Item* que se encuentra en la posición *num.*
void insert(int num, Item item)	Inserta un *Item* justo antes del que ocupa la posición *num.*
void set(int num, Item item)	Reemplaza el *Item* que ocupa la posición *num.*
int size()	Devuelve el número de *Items* dentro de la pantalla *Form.*

Tabla 3.11 Métodos de la clase *Form* [37]

3.1.4.4.3. Clase List

Esta clase proporciona una pantalla que contiene una lista de elementos sobre los que el usuario puede seleccionar. Se tienen los siguientes tipos de listas:

- ***EXCLUSIVE*:** permite seleccionar un solo elemento a la vez.
- ***IMPLICIT:*** lista en la que la selección de un elemento provoca un evento.
- ***MULTIPLE*:** permite seleccionar uno a más elementos a la vez.

Para crear una lista se puede usar los constructores:

List (String title, int listType)

List(String title, int tlistType, String [ ] stringElements, Image[ ] imageElements)

Donde:

- *title:* es el título que va a llevar la lista.
- *listType:* especifica el tipo de lista que se va a crear.
- *stringElements:* representa los elementos de la lista.
- *imageElements:* representa las imágenes que se van a mostrar en la lista.

Algunos de los métodos de la clase *List* que permiten manipularla se muestran en la Tabla 3.12.

Método	Descripción
int append(String texto, Image imagen)	Añade un elemento al final de la lista.
void delete (int posición)	Elimina el elemento de la posición especificada.
void deleteAll()	Elimina todas las entradas de la lista.
void insert(int pos, String texto, Image im)	Inserta un elemento en la posición especificada.
int getSelectedFlags(boolean[ ] arreglo)	Para listas de tipo MULTIPLE indica los elementos seleccionados.
int getSelectedIndex()	Obtiene el índice del elemento seleccionado en las listas de tipo EXCLUSIVE o IMPLICIT.
boolean isSelected(int posicion)	Determina si está seleccionado el elemento de la posición indicada.
int setSelectedFlags(boolean [ ] arreglo)	Permite activar elementos como seleccionados dentro de una lista tipo MULTIPLE.
int setSelectedIndex(int pos, boolean selec)	Permite activar un elemento como seleccionado en las listas de tipo EXCLUSIVE o IMPLICIT.

Tabla 3.12 Métodos de la clase *List* [37]

3.1.4.4.4. Clase TextBox

Esta clase implementa un componente de edición de texto que ocupa toda la pantalla. El constructor de esta clase es el siguiente:

TextBox (String title, String text, int maxSize, int constraints)

Donde:

- *title:* es el título que va a llevar el componente.
- *text*: es el texto que va a llevar el componente.
- *maxSize:* especifica el número máximo de caracteres que pueden ser introducidos.
- *constraints:* representa las limitaciones de acuerdo al tipo de texto que haya sido especificado.

3.2. ENTORNO *LEGO MINDSTORMS NXT* Y EL SISTEMA OPERATIVO *leJOS*

En el presente proyecto se utilizará el *kit Lego Mindstorms NXT 1.0* de la empresa *Lego*. Este *kit* permite armar y programar diferentes robots de una forma sencilla por lo que son muy utilizados en ambientes educativos.
Este *kit* cuenta con diferentes piezas de *hardware* y elementos de *software* que se detallan a continuación.

3.2.1. COMPONENTES DEL *KIT LEGO MINDSTORMS NXT*

3.2.1.1. Componentes de *Hardware* [41] [42]

Los componentes de *hardware* se muestran en la Figura 3.8 y se pueden dividir en:

- Unidad central de control: *NXT Brick.*
- Dispositivos de salida: motores.
- Dispositivos de entrada: sensores.

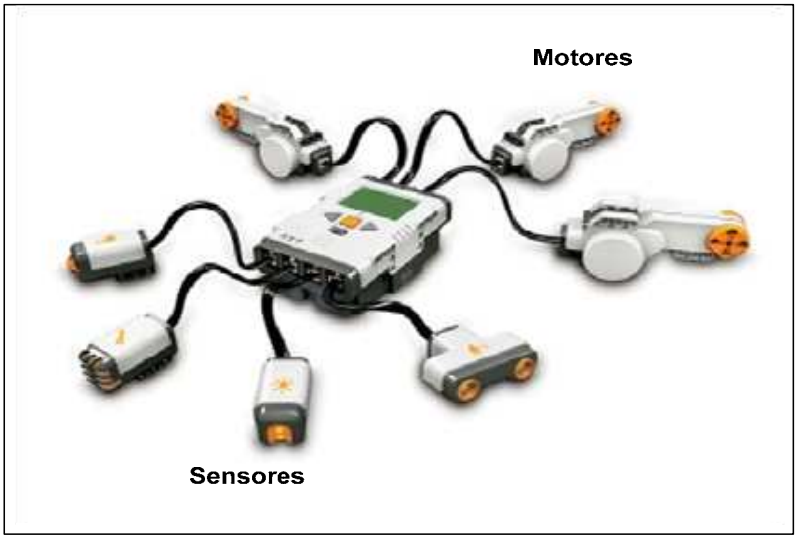


Figura 3.8 Componentes de *hardware* del sistema *Lego Mindstorms NXT* [62]

3.2.1.1.1. NXT Brick [49]

Este es el componente central del robot, actúa como cerebro y está compuesto por:

- Un procesador principal de 32 bits, *AT91SAM7S256* con 256 KBytes de memoria *flash* y 64 KBytes de memoria *RAM*. Trabaja a 48 MHz.
- Un co-procesador *Atmel* de 8 bits *AVR*, *ATmega48* con 4KBytes de memoria flash y 512 Bytes de memoria *RAM*. Trabaja a 8MHz.
- Cuatro puertos de entrada que son usados para conectar los sensores, llamados puertos 1, 2, 3 y 4.
- Tres puertos de salida que son usados para conectar los motores, llamados puertos A, B y C.
- Un puerto *USB* 2.0 y un modulo *Bluetooth CSR Bluecore 4* descrito anteriormente, compatible con la versión *Bluetooth* 2.0 para la comunicación.
- Pantalla *LCD* de 100x64 pixeles.
- Un altavoz de 8 KHz y un canal de sonido con 8 bits de resolución.
- Fuente de poder : 6 baterías AA.

El la Figura 3.9 se muestra al *NXT Brick* y en la Figura 3.10 se muestra el diagrama de bloques de los elementos de *hardware* que lo componen.

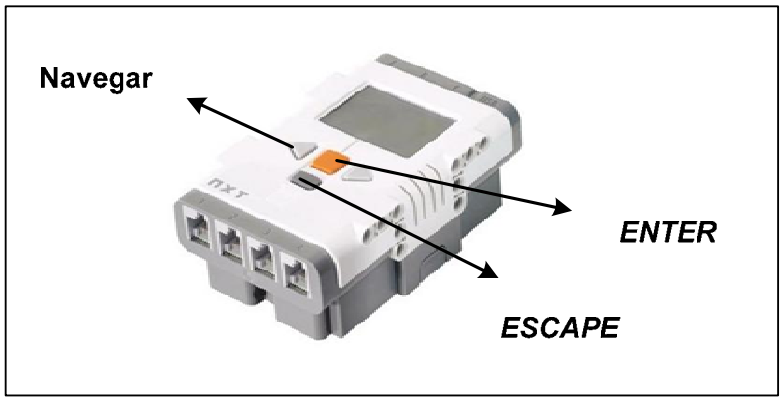


Figura 3.9 *NXT Brick*

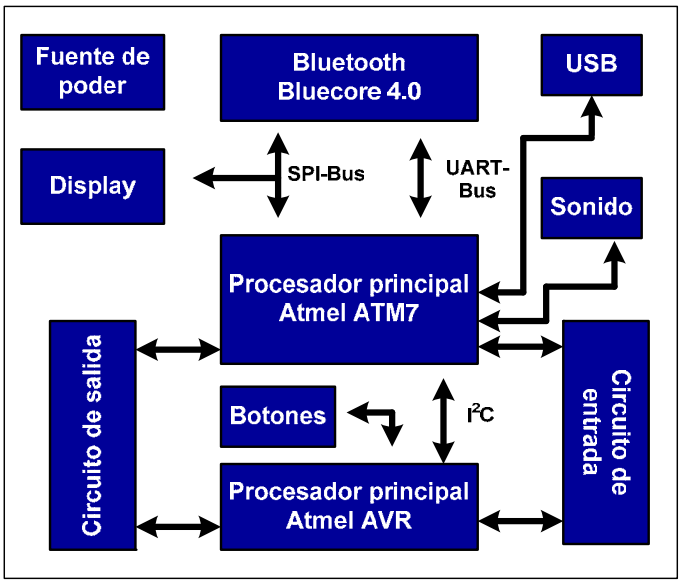


Figura 3.10 Diagrama de bloques de los elementos de hardware del *NXT Brick*[49]

3.2.1.1.2. Motores

El *kit NXT* viene con tres motores como los de la Figura 3.11 que permitirán al robot realizar tareas como desplazarse, levantar cargas, tomar objetos o cualquier otra tarea que requiera energía.

Figura 3.11 Motor *NXT* [63]

Los motores *NXT* son servos, esto significa que su posición interna y estado puede ser controlada por una unidad externa, en este caso por el *NXT Brick*. Para ello los motores cuentan con un sensor de rotación que lleva la cuenta de las rotaciones de los ejes del motor. Se puede rotar con una precisión de 1 grado.

La velocidad del motor depende del voltaje de la batería y de la carga. Sin carga, la máxima velocidad es 100 veces el voltaje.

Algunos datos sobre este tipo de motor se muestran en la Tabla 3.13.

Voltaje Máximo	Corriente mínima (sin carga)	Corriente Máxima	Velocidad Máxima (sin carga)
9 V DC	60 mA	2A	170 rpm

Tabla 3.13 Características de los motores *NXT* [42]

3.2.1.1.3. Sensores

Los sensores permiten al robot *NXT* interactuar con el mundo exterior. El *kit NXT* tiene cuatro sensores diferentes que son:

- Sensor de Luz

  Su función es medir la cantidad de luz en una dirección particular. Cuenta con un *led* que al emitir la luz en una dirección determinada permite obtener la intensidad de luz reflejada por el objeto en esa dirección. El valor obtenido es un valor relativo expresado en porcentaje.

  La cantidad de luz reflejada por una superficie depende de varios factores, principalmente del color, la textura y la distancia a la que se encuentra la superficie.

  En la Figura 3.12 se muestra el sensor de luz.

Figura 3.12 Sensor de Luz [64]

- Sensor ultrasonido [64]

  Permite la detección de objetos y determinar a qué distancia se encuentran. El rango de detección está entre 0 y 255 cm y tiene una precisión de +/- 3 cm. Para tomar las medidas este sensor utiliza señales sonoras de alta frecuencia que se reflejan en un objeto. El sensor toma la señal de vuelta para la medición.

En la Figura 3.13 se muestra el sensor ultrasonido.

Figura 3.13 Sensor Ultrasonido [65]

- Sensor de contacto

  Consiste de un pequeño botón interruptor (*pushbutton*) que permite detectar eventos de presión y liberación del mismo. Este sensor puede ser usado para detectar obstáculos muy cerca del robot.

  En la Figura 3.14 se muestra el sensor de contacto

Figura 3.14 Sensor de contacto [64]

- Sensor de sonido

  Este sensor es capaz de detectar sonidos en dos modos. El primer modo es *dbA* (*adjusted decibels*) que se adapta a la sensibilidad del oído humano, es decir no considera las frecuencias muy bajas o muy altas (considera frecuencias entre 3 y 6 KHz). El segundo modo es *db* (*decibels*) que registra el ancho de banda de frecuencias por igual. Este sensor entrega sus resultados en porcentaje en base a su volumen máximo que es 90 decibelios.

  En la Figura 3.15 se muestra el sensor de sonido.

Figura 3.15 Sensor de sonido [64]

En el proyecto se armará un brazo robot como el que se muestra en la Figura 2.12 con el *kit Lego Mindstorms NXT* versión 1.0. Se emplearán los 3 servomotores que permitirán el movimiento del brazo en las direcciones derecha- izquierda, arriba-abajo, abrir y cerrar la mano. Además se emplearán los sensores de tacto y luz.

3.2.1.2. Componentes de Software [41]

Como cualquier dispositivo programable el *NXT* requiere de elementos de *Software*. Los elementos con los que se cuenta son:

- Un sistema operativo: el *firmware.*

El *NXT Brick* tiene su propio *firmware* que puede ser considerado como su sistema operativo. Este viene almacenado en la memoria *flash* por lo que no será borrado si se apaga o se retiran las baterías. Para este *firmware* se tiene un sistema de programación gráfico llamado *NXT-G*. Este lenguaje de alto nivel funciona por medio de bloques con funcionalidades específicas, que se conectan generando rutinas que luego son transferidas al procesador del robot para su posterior ejecución.

Este *firmware* puede ser reemplazado por otros tipos de *firmware* que ofrezcan un mejor desempeño, características adicionales o soporte a algún lenguaje de programación específico. En el presente proyecto se reemplazará el *firmware* original y se utilizará el sistema operativo *leJOS* (*Lego Java Operating System*) que permite el desarrollo de programas *Java* para el robot.

El procedimiento para reemplazar el *firmware* original por el *firmware leJOS* se muestra en el ANEXO E.5.

- Un sistema de archivos

  El *NXT Brick* contiene un sistema de archivos llamado *TOC* (*Table of Contents*). Este es usado para almacenar objetos persistentes como programas y archivos de datos. Se permite un máximo de 63 *items*.

- Herramientas para la programación

  Se disponen de diferentes entornos de desarrollo *IDEs* dependiendo del *firmware* que se vaya a utilizar. Para el *firmware leJOS* que se utilizará se tienen *IDEs Java* como por ejemplo el *IDE NetBeams* que se utilizará en el presente proyecto. Este *IDE* dispone de un *plugin* (complemento) que permite

la creación de proyectos *leJOS*. En el ANEXO E.7 se muestra la forma de añadir este *plugin*. Este *IDE* también permite transferir directamente el proyecto *leJOS* creado al *NXT brick* como se indica en el ANEXO F.1.3.

3.2.2. *leJOS* (*LEGO JAVA OPERATING SYSTEM*) [44] [57] [58]

leJOS es un proyecto *open source* que permite a los desarrolladores realizar programas *Java* para los robots *Lego Mindstorms*. Su descarga es gratuita y se la puede realizar desde el *URL*: *http://lejos.sourceforge.net/*

Reemplaza al *firmware* original y emplea una máquina virtual *Java* que es la encargada de interpretar el *bytecode* para comunicarle al *NXT Brick* que es lo que debe hacer.

Algunas de las características de *leJOS* son:

- Soporte para programación orientada a objetos (*Java*).
- Es un proyecto *open source* con varios colaboradores.
- Tiene soporte multiplataforma: *Windows, Linux, MAC OS.*
- Es más rápido que el *NXT-G.*
- Tiene soporte para el uso de hilos.
- Tiene soporte para comunicaciones *Bluetooth.*
- Proporciona alta precisión en el control de los motores del robot.
- Tiene soporte para telerobótica a través de sockets *TCP/IP.*
- Soporta monitoreo remoto y seguimiento de los programas *leJOS* desde el *PC*.

leJOS cuenta con un *API* llamado *NXJ* (*Java for Lego Mindstorms*) *API* que está formado por varias clases que permiten darle funcionalidad a los programas.

Los paquetes de comunicación se muestran en la Tabla 3.14.

Paquetes de comunicación	Descripción
java.io	Soporte para entrada y salida.
javax.bluetooth	Clases estándar de *Bluetooth.*
javax.microedition.io	*J2ME I/O.*
lejos.nxt.comm	Clases de comunicación *NXT.*
lejos.nxt.socket	Soporte para *sockets* vía *PC.*
lejos.net.remote	Acceso remoto *NXT* sobre *Bluetooth.*

Tabla 3.14 Paquetes del *NXJ* para la comunicación [50]

Los paquetes que permiten la administración del *NXT Brick*, sensores y motores se muestran en la Tabla 3.15.

Paquetes de administración	Descripción
javax.microedition.lcdui	Clases *J2ME* para interfaces de usuario.
javax.microedition.location	Clases *J2ME* para soporte de servicios de localización.
lejos.gps	Soporte para *GPS.*
lejos.nxt	Proporciona acceso a los sensores, motores *NXT.*
lejos.nxt.addon	Proporciona acceso para hardware que no ha sido incluido en el *kit Lego NXT.*
lejos.nxt.debug	Depuración de las clases de hilos.
lejos.util	Utilidades.

Tabla 3.15 Paquetes del *NXJ* para la administración del *NXT Brick*, sensores y motores. [50]

Los paquetes que permiten dar soporte a aspectos de robótica como localización, navegación se muestran en la Tabla 3.16.

Además el *API* cuenta con paquetes del núcleo *Java* como son: *java.awt*, *java.lang* y *java.util*.

Paquetes de Robótica	Descripción
lejos.robotics.localization	Soporte para localización.
lejos.robotics.navigation	Clases de navegación.
lejos.robotics.math	Clases de matemáticas.
lejos.robotics.subsumtion	Soporte para la arquitectura de subsunción.

Tabla 3.16 Paquetes de robótica del *NXJ* [50]

A continuación se detallarán las clases necesarias para desarrollar el programa para el *NXT Brick* en el presente proyecto.

3.2.2.1. Clases para el uso de los sensores de luz y tacto

3.2.2.1.1. Clases para el uso del sensor de luz

Para acceder al sensor de luz se emplea la clase *LightSensor* del paquete *lejos.nxt.*

Se tiene dos constructores:

LightSensor (SensorPort puerto)
LightSensor (SensorPort puerto, boolean foco)

El parámetro de entrada *puerto* puede tomar el valor de las siguientes constantes que representan a los cuatro puertos del *NXT Brick* donde se conectan los sensores:

- *SensorPort.S1.*
- *SensorPort.S2.*
- *SensorPort.S3.*
- *SensorPort.S4.*

El parámetro *foco* permite encender o apagar el *led* del sensor dependiendo si tiene el valor de *true* o *false*.

El método que permite obtener la lectura del sensor de luz es:

int readValue()

3.2.2.1.2. Clases para el uso del sensor de tacto

Para acceder al sensor de tacto se emplea la clase *TouchSensor* del paquete *lejos.nxt.*
Su constructor es:

TouchSensor (SensorPort puerto)

Esta clase dispone del método:

boolean isPressed()

Este método chequea si el sensor está presionado, en caso afirmativo devuelve el valor de *True*.

3.2.2.2. Clases para el manejo de motores

Se tiene la clase *Motor* del paquete *lejos.nxt.* Esta clase es una abstracción de los motores *NXT* y proporciona una interfaz para cada puerto a los que se conectan los motores, estas son: *Motor.A*, *Motor.B* y *Motor.C*.

Adicionalmente se cuenta con la interfaz *Tachometer* que es una abstracción del sensor de rotación del motor *NXT* y permite monitorear su velocidad. Algunos de los métodos que permiten el control de los motores se muestran en la Tabla 3.17.

Métodos	Descripción
void backward()	Provoca que el motor rote hacia atrás.
void forward()	Provoca que el motor rote hacia adelante.
void stop()	Provoca que el motor se detenga.
void setSpeed(int velocidad)	Establece la velocidad el motor en grados por segundo. La máxima velocidad es 900 con 8 V.
void rotate(int ang)	Provoca que el motor rote en un ángulo (*ang*) específico.
int getTachoCount()	Obtiene la cuenta del tacómetro.
void resetTachoCount()	Pone la cuenta del tacómetro en cero.

Tabla 3.17 Algunos métodos para el control de motores *NXT*. [57]

3.2.2.3. Clases para el manejo de la pantalla *LCD*

Esta clase no tiene instancias ya que solo existe un *LCD* en el *NXT*. Se tiene un modo texto y un modo gráfico.

En el modo texto se debe considerar que la pantalla admite 16 caracteres en el ancho y 8 en profundidad. Y para acceder a cada posición se lo hace mediante las coordenadas (x, y). Donde *x* toma los valores de 0 a 15 y de 0 a 7.

Algunos de los métodos que se tiene se muestran en la Tabla 3.18.

Métodos	Descripción
void drawString (String str, int x, int y)	Escribe texto sobre la pantalla iniciando en la coordenada (x,y).
void drawInt(int i, int x, int y)	Escribe un entero sobre la pantalla iniciando en la coordenada (x,y).
void clear ()	Limpia la pantalla.

Tabla 3.18 Algunos métodos para el manejo de la pantalla *LCD* del *NXT Brick*[57]

En el modo gráfico se debe considerar que la pantalla tiene 100 pixeles de ancho y 64 pixeles de largo. Son accedidos mediante las coordenadas (x,y) que representan

a los pixeles. Para graficar en la pantalla se utiliza la clase *Graphics* del paquete *javax.microedition.lcdui,* esta clase permite dibujar líneas, rectángulos y arcos.

3.2.2.4. Clases para el manejo de Botones

Se tiene la clase *Button* con 4 instancias que representan los 4 botones del *NXT Brick*:

- *Button.ENTER.*
- *Button.ESCAPE.*
- *Button.LEFT.*
- *Button.RIGHT.*

Algunos de métodos de esta clase se muestran en la Tabla 3.19.

Método	Descripción
boolean isPressed()	Chequea si el botón se encuentra presionado.
void waitForPress()	Espera hasta que un botón sea presionado.
int readButtons()	Lee el estado de los botones.

Tabla 3.19 Algunos métodos de la clase *Button.* [57]

Además se debe añadir un *listener* implementando la interfaz *ButtonListener* para escuchar los eventos generados por los botones. Esto se lo realiza mediante el método:

void addButtonListener (ButtonListener listener)

La interfaz *ButtonListener* implementa los métodos:

- ***void buttonPressed (Button btn):*** es llamado cada vez que un botón ha sido presionado. Permite determinar que acción se desea realizar según el botón presionado.
- ***void buttonReleassed (Button btn):*** es llamado cuando el botón ha sido liberado.

3.2.2.5. Clases para la comunicación *Bluetooth*

Se tiene varias clases que permiten establecer la comunicación mediante *Bluetooth* con otros dispositivos.

- Búsqueda de dispositivos

  Si no se tiene conocimiento sobre los dispositivos que se encuentran alrededor se debe llevar a cabo el proceso de *Inquiry*. Para ello en la clase *Bluetooth* del paquete *lejos.nxt.comm* se tiene el siguiente método:

  Vector inquire (int maxDevices, int timeout, byte[] code)

  Donde:

  - *maxDevices*: representa el máximo número de dispositivos a descubrir.
  - *timeout* : representa el tiempo de espera (1.28 s).
  - *code: representa* la clase de dispositivo que se va a buscar.

  Este método retorna un vector con los dispositivos encontrados a partir de los cuales se obtiene los objetos *RemoteDevice*.

- Conexión con un dispositivo *Bluetooth*

Una vez que se tiene el *RemoteDevice* se puede establecer la conexión con un dispositivo específico mediante el siguiente método de la Clase *Bluetooth*:

BTConnection connect (RemoteDevice dispositivoRemoto)

Este método devuelve un objeto *BTConnection* que representa la conexión *Bluetooth* establecida. Si no se pudo llevar a cabo la conexión retorna *null*.

- Intercambio de datos entre el *NXT Brick* con otro dispositivo *Bluetooth*

  Una vez que se haya establecido la conexión a través del objeto *BTConexión* se obtiene un objeto *StreamComexión* (interfaz del paquete *javax.microedition.io*) que permite obtener y abrir los flujos (*streams*) de entrada y salida a través de los métodos *openDataInputStream()* y *openDataOutputStream()* respectivamente.

  Para enviar y leer los datos se puede emplear los métodos *read()* y *write()* de la clases *DataInputStream* y *DataOutputStream* respectivamente pertenecientes al *paquete java.io*.

- Escuchar conexiones *Bluetooth*

  Si el *NXT* se encuentra como servidor deberá escuchar las solicitudes de conexión de los clientes para ello se tiene el método de la clase *Bluetooth*:

  BTConnection waitForConnection()

  Este método retorna un objeto *BTConnection* que representa la conexión *Bluetooth* establecida .Una vez establecida la conexión se realiza el intercambio de datos.

3.2.2.6. *Threads* [48]

leJOS soporta el uso de *Java Thread* (hilos) que permiten realizar tareas concurrentes.

En la programación de robots se pueden tener varios *threads* como por ejemplo:

- Un *thread* para administrar la locomoción del sistema.
- Un *thread* para administrar la comunicación *Bluetooth.*
- Un *thread* que administre los sensores del robot.

Una forma de implementar un *thread* en *Java* es crear una clase que se derive de la clase *Thread*. Esta clase se encuentra en el paquete *java.lang*. Al hacer esto se debe sobrescribir el método *run(),* dentro de este método irá el código que se desea que se ejecute cuando inicie el *thread* como se muestra en la Figura 3.16.

En tareas concurrentes se debe sincronizar los objetos para que no sean accedidos por varios *threads* a la vez. Una forma de hacerlo es mediante la palabra clave *synchronized* como se muestra en la Figura 3.17 Esto hace que si un *thread* está haciendo uso de un objeto, un segundo no pueda hacerlo hasta que el primero lo deje.

```
public class ejemploHilo extends Thread
{

    public void run()
   {

            // Código que se desea que se ejecute cuando
            // el hilo inicie
   }

 }
```

Figura 3.16 Ejemplo para implementar un *thread* [48]

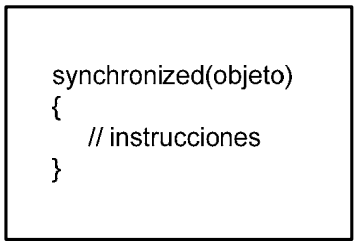


Figura 3.17 Ejemplo de uso de la palabra clave *synchronized.* [48]

El *API* de *Java* para sincronización también proporciona los métodos *wait()* y *notify()* del paquete *java.lang*

El *método wait()* espera a que una condición ocurra y es usado dentro de un bloque que ha sido sincronizado.

El método *notify()* notifica a un *thread* que se encuentra esperando que la condición ocurra.

CAPÍTULO 4: DESARROLLO DEL SISTEMA, PRUEBAS Y COSTOS REFERENCIALES

4.1. DISEÑO DE UN PROTOCOLO DE COMUNICACIÓN PARA LA TRANSMISIÓN DE COMANDOS

El protocolo de comunicación a diseñarse permitirá la transmisión de comandos a través de la tecnología inalámbrica *Bluetooth*, desde un teléfono celular hacia un brazo robot, para permitir a un usuario el control del mismo.

El brazo robot es capaz de realizar la lectura del color de una pelota utilizando un sensor de color y un pulsador. Cuando eso sucede el brazo robot compara dicha lectura con un color previamente establecido por el usuario, y en caso de que coincidan, el brazo robot capturará la pelota automáticamente y podrá ser trasladada según las órdenes del usuario.

Tres servo motores conectados a los puertos *A*, *B*, y *C* del *NXT brick* permiten el control de los movimientos horizontal, vertical y abrir y cerrar la mano del brazo robot respectivamente.

Debido a que solo dos equipos intervienen en la comunicación, se dice que el protocolo de comunicación tendrá una configuración de línea punto a punto. La Figura 4.1 muestra el ambiente donde actuará el protocolo de comunicación.

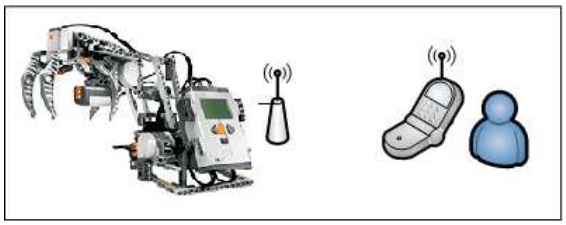


Figura 4.1 Equipos que intervienen en la comunicación

4.1.1. METODOLOGÍAS PARA DESARROLLO DE *SOFTWARE* [20]

"La metodología para el desarrollo de *software* es un modelo sistemático que permite realizar, gestionar y administrar un proyecto. Esta sistematización nos indica cómo dividiremos un gran proyecto en módulos más pequeños llamados etapas, y las acciones que corresponden en cada una de ellas, nos ayuda a definir entradas y salidas para cada una de las etapas y, sobre todo, normaliza el modo en que administraremos el proyecto. Entonces, una metodología para el desarrollo de *software* son los procesos a seguir sistemáticamente para idear, implementar y mantener un producto *software* desde que surge la necesidad del producto hasta que cumplimos el objetivo por el cual fue creado." [71]

La metodología para desarrollo de *software* ofrece las siguientes ventajas:

- Permiten desarrollar un *software* de mejor calidad.
- Proporcionan un lenguaje común para los desarrolladores.
- Proporcionan características comunes acerca del *software* a todos los desarrolladores.

4.1.1.1. Modelo con ciclo de vida en cascada [20]

Es un modelo lineal y secuencial, que es usado para describir distintas actividades relacionadas con el desarrollo de *software*. Los periodos en que estas actividades ocurren son conocidos como fases. Su simplicidad y su antigüedad dentro del mercado del *software* (aproximadamente 30 años), convierten a este modelo en uno de los más populares para el desarrollo de *software*.

El número de fases que conforman este modelo se muestra en la Figura 4.2:

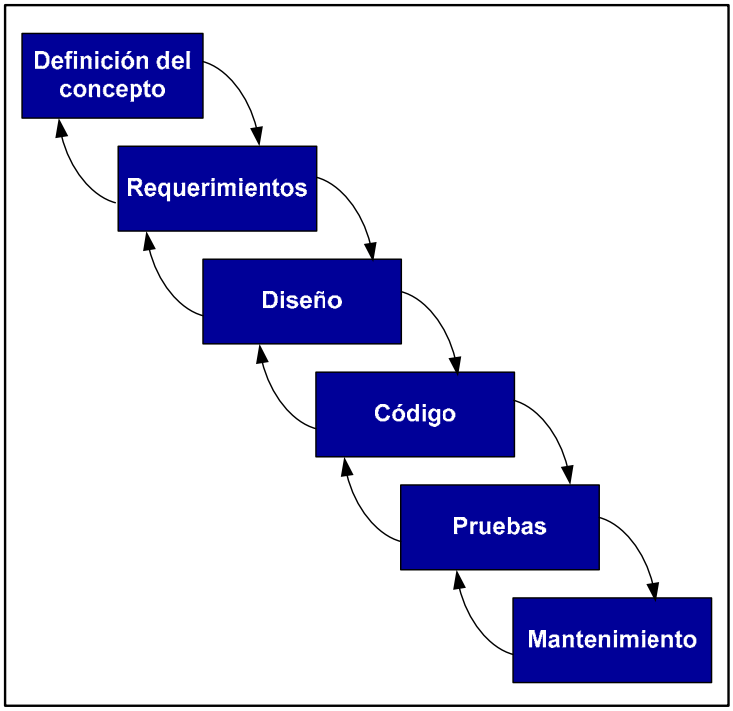


Figura 4.2 Fases que conforman el modelo en cascada [20]

La función que cumple cada etapa se describe a continuación:

- **Definición del Concepto:** en esta fase se describen los objetivos del sistema.
- **Requerimientos:** se decide que debe realizar el *software* y su alcance.
- **Diseño:** muestra cómo el *software* debe cumplir los requerimientos.
- **Código:** se procede a la construcción del sistema.
- **Pruebas:** se demuestra que el *software* cumple con los requerimientos.
- **Mantenimiento:** esta fase permite adicionar características o corregir de posibles errores del *software.*

4.1.1.2. Modelo con ciclo de vida en espiral

Este modelo corrige ciertas características del modelo en cascada que no permiten realizar una representación real del sistema. A diferencia del modelo en cascada, el modelo en espiral permite realizar una serie de prototipos estratégicos y análisis de

riesgos de cada uno de ellos a lo largo del ciclo de vida del desarrollo del *software*. Este modelo se muestra a continuación en la Figura 4.3:

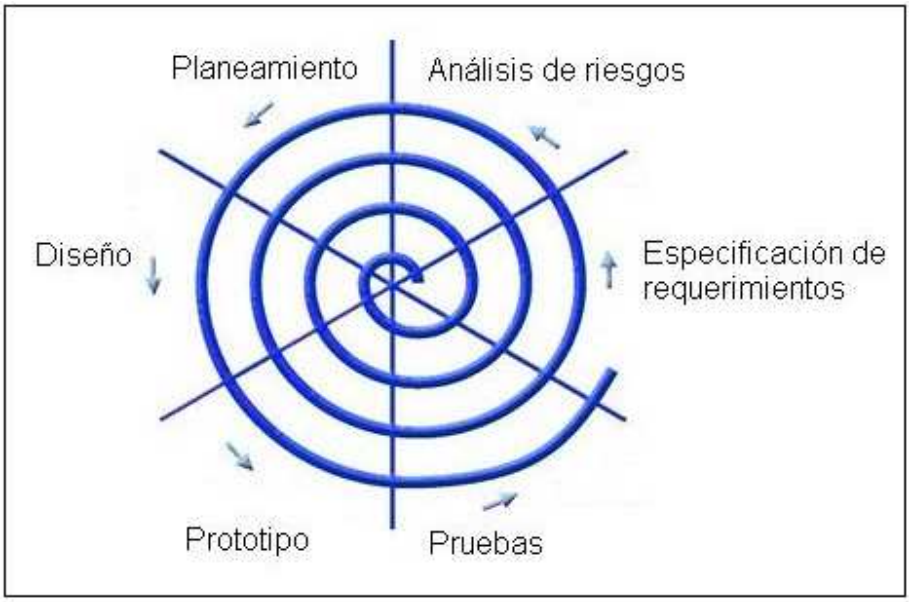


Figura 4.3 Etapas que conforman el modelo en espiral [20]

La función que cumple cada etapa se describe a continuación:

- **Especificación de Requerimientos:** en esta etapa se requiere que el desarrollador plantee los requerimientos del sistema en detalle.
- **Análisis de Riesgos:** se identifica todos los riesgos que deberá enfrentar el desarrollador para cumplir con todos los requerimientos que fueron planteados en la primera etapa.
- **Planeamiento:** el desarrollador planteará una estrategia que permita la implementación del *software*.
- **Diseño:** se procede a la escritura del programa.
- **Primer Prototipo del Sistema:** se construye el primer prototipo del sistema, el mismo que se debe aproximar a las características finales del sistema ya que es la base para los siguientes prototipos.

- **Pruebas:** el desarrollador prueba el primer prototipo y si este no cumple con ciertos requerimientos, entonces este procederá al análisis del segundo prototipo siguiendo los mismos pasos anteriores.

4.1.2. IMPLEMENTACIÓN DEL SISTEMA [19] [39] [40] [50] [58]

La implementación del sistema se realizará en base al modelo con ciclo de vida en espiral debido a los siguientes criterios:

- Ofrece un mejor enfoque cuando el responsable del desarrollo del *software* está inseguro de la eficacia de un algoritmo, de la adaptabilidad de un sistema operativo o de la forma que debería tomar la interacción humano-máquina.

- Se utiliza particularmente en sistemas embebidos por su fortaleza en el análisis de riesgos para cada prototipo que se implementa, lo que permite que el resultado sea un *software* robusto.

Se aprovechará también las características del lenguaje *Java* como son su modularidad y su reusabilidad*,* es decir*,* muchos de los módulos o clases que se implementarán dentro de este sistema deberán proporcionar la flexibilidad necesaria para utilizarlos en cualquier otra aplicación y también para modificarlos en cualquier momento.

4.1.2.1. Especificación de requerimientos

4.1.2.1.1. *Especificación de la hipótesis del medio*

Para cumplir con sus funciones, el protocolo de comunicación debe comunicarse con su ambiente, que está compuesto por los siguientes elementos:

- El usuario.
- El brazo robot *Lego Mindstorms.*
- El teléfono celular 1 ó teléfono celular 2.
- El canal de comunicación.

El teléfono celular 1 y el teléfono celular 2 corresponden a un *Sony Ericcson* W580 y a un *Nokia* 5220 respectivamente. Ambos teléfonos contienen el perfil *MIDP* 2.0 y la configuración *CLDC* 1.1.

El usuario podrá manipular tanto al brazo robot como al teléfono celular para iniciar la fase de establecimiento de la conexión, transferencia de información, y finalización de la conexión.

Los mensajes del protocolo de comunicación serán intercambiados sobre un canal de comunicación inalámbrico *full dúplex,* el mismo que será configurado utilizando el perfil *SPP* (*Serial Port Profile*) de la tecnología inalámbrica *Bluetooth* versión 2.0 con *EDR* (*Enhanced Data Rate*).

El enlace serial alcanza una velocidad efectiva *uplink* asíncrona entre 108.8 Kbps y 2.1 Mbps, dependiendo del tamaño del mensaje que se va a transmitir (Ver Tabla 2.4).

El *BER* (*Bit Error Rate*) para la tecnología *Bluetooth* versión 2.0 con *EDR* es de 0.01% y en modo básico (sin *EDR*) 0.1%.

La Tabla 4.1 muestra las características, respecto a la tecnología *Bluetooth,* de los equipos de comunicación que se emplearán.

Se debe tomar en cuenta para el diseño del protocolo de comunicación que la tecnología *Bluetooth* utiliza las siguientes características:

	Equipo		
Característica	**Brazo robot**	**Teléfono Celular 1**	**Teléfono Celular 2**
Bluetooth* v2,0 + *EDR	Si	Si	Si
Clase de dispositivo	2	2	2
Serial Port Profile	Si	Si	Si
JSR 82	Si	Si	Si

Tabla 4.1 Características de los equipos de comunicación. [34] [76]

- En capa banda base se encuentra implementado el método de control de flujo *stop and wait* y un temporizador denominado *flush timeout*[33], el mismo que por defecto está configurado para realizar un número indefinido de retransmisiones hasta recibir un acuse de recibo positivo del paquete enviado. *JSR*-82 no permite modificar ese parámetro [19].

- *Bluetooth* utiliza polinomios *CRC* de orden 16 para todos los tamaños de mensajes de la Tabla 2.5. Esto ocasiona que algunos mensajes errados no sean detectados por dicho polinomio debido a la distancia de *Hamming* y pasen a capas superiores [75].

Por otra parte el brazo robot contiene un sensor de luz, el mismo que depende de la intensidad de luz del ambiente donde se trabaje. El programa para el brazo robot se lo realizará en un ambiente con poca luz ambiental.

[33] Temporizador *Flush Timeout*: Indica durante cuánto tiempo se puede llevar a cabo la retransmisión del paquete. Después de este tiempo se detiene las retransmisiones forzando al controlador del enlace a tomar los siguientes datos. [29]

4.1.2.1.2. Especificación de los servicios del protocolo de comunicación

El protocolo de comunicación será orientado a la conexión para la transmisión de la información y ofrecerá los siguientes servicios:

- Preservación de secuencia.
- Sincronización de unidades de datos orientado a bloques.
- Detección y control de errores.
- Control de flujo.

El protocolo de comunicación deberá ser capaz de enviar un comando a la vez, controlar posibles errores que provengan de las capas inferiores debido al polinomio *CRC*-16 que utilizan, y de limitar el número de retransmisiones del temporizador *flush timeout.* Por lo tanto se utilizará para el control de flujo y control de errores el método *stop and wait*.

Los diagramas de flujo del método *stop and wait* para el transmisor y receptor se muestran en el ANEXO A.1 y A.2 respectivamente.

4.1.2.2. Análisis de riesgos del sistema

Los riesgos para implementar los requerimientos anteriores además de ciertas consideraciones son:

- El protocolo de comunicación debe realizar varias tareas simultáneamente, por lo cual, se debe recurrir a la programación concurrente para cumplir con ese objetivo.

Java utiliza procesos ligeros para permitir la creación de sistemas multitarea, por tal razón, se debe tener cuidado ya que estos comparten recursos entre sí de una manera no determinística.

- Todos los hilos de ejecución que se produzcan debido al protocolo de comunicación deben ejecutarse una sola vez durante el ciclo de vida del protocolo de comunicación.

- La capacidad de procesamiento del *NXT brick* y su capacidad de memoria es baja. Estas características limitan el uso excesivo de hilos de ejecución dentro de este equipo.

- El protocolo de comunicación será implementado basado en el modelo de referencia *OSI,* el mismo que permite el uso de procesos pares, es decir, los módulos que se implementen en el teléfono celular podrían re utilizarse en el brazo robot bajo pequeñas diferencias. Para lograr este objetivo, se debe observar que ambos equipos trabajen con librerías de *Java* comunes.

- El *NXT brick* no dispone de las bases de datos de servicios que ofrece *J2ME* para los teléfonos celulares, por tal razón se debe proporcionar un método especial para el establecimiento de la conexión.

- El acceso a los servicios del *SAP* debe ser inmediato e independiente de la clase para facilitar la implementación del sistema.

4.1.2.3. Planeamiento para la implementación del sistema

La implementación del sistema estará dividida en tres fases que corresponden al ambiente donde se desempeñará el protocolo de comunicación:

- El protocolo de comunicación.
- La interfaz gráfica de usuario.
- El programa para el *NXT brick.*

4.1.2.3.1. Planeamiento e implementación del modelo de validación del protocolo de comunicación [1] [9] [10] [11]

Para la creación del modelo de validación del protocolo de comunicación se debe considerar los siguientes criterios:

- Especificación de los tipos de mensajes que se utilizarán en el modelo. Estos mensajes son una abstracción de los mensajes que se utilizarán en la implementación, por lo tanto, deben ocultar detalles acerca de los formatos de los mismos.

- Especificación de las reglas y procedimientos para el intercambio de datos.

- Escritura del programa en base a las condiciones anteriores en lenguaje *PROMELA* para verificar su validez mediante la herramienta *SPIN,* la misma que permite simular el comportamiento del sistema antes de su implementación.

El protocolo de comunicación intercambiará mensajes con el usuario y con el brazo robot a través de canales internos sincrónicos, los mismos que simulan las llamadas a métodos de *Java*.

Dentro del modelo, el usuario podrá enviar únicamente cuatro tipos de mensajes hacia el protocolo de comunicación que son:

- **pCon:** permitirá al usuario realizar una petición de conexión al protocolo de comunicación desde el teléfono celular o desde el brazo robot.

- **pTx:** permitirá al usuario realizar una petición de transmisión de comandos al protocolo de comunicación desde el teléfono celular.

- **pFinCon:** permitirá al usuario realizar una petición de finalización de la conexión desde el teléfono celular o desde el brazo robot.

- **msg:** corresponde al mensaje que desea enviar el usuario. Este mensaje representa un mensaje de ejecución de comando.

El brazo robot será capaz de comunicarse con el protocolo de comunicación utilizando los siguientes mensajes:

- **pTx:** permitirá al brazo robot realizar una petición de transmisión de acuses de recibo al protocolo de comunicación.

- **ack:** corresponde a un *ACK* que desea enviar el brazo robot luego de haber recibido un msg correcto.

- **nack:** corresponde a un *NACK* que desea enviar el brazo robot luego de haber recibido un msg incorrecto.

- **error:** corresponde a un *ACK* o un *NACK* que sufrió un error debido al canal de comunicación inalámbrico.

- **rojo:** cuando el brazo robot realiza la lectura de una pelota de color rojo, se envía este mensaje para informar al usuario del acontecimiento

- **azul:** cuando el brazo robot realiza la lectura de una pelota de color azul, se envía este mensaje para informar al usuario del acontecimiento

Los mensajes que utilizará el protocolo de comunicación para comunicarse con el usuario son los siguientes:

- **conf:** permite al protocolo de comunicación enviar al usuario una confirmación positiva acerca de una petición de transmisión o de finalización de conexión que realizó previamente.

- **nconf:** permite al protocolo de comunicación enviar al usuario una confirmación negativa acerca de una petición de transmisión o de finalización de conexión que realizó previamente.

- **con:** permite al protocolo de comunicación enviar al usuario una confirmación positiva en caso de que la conexión se estableció correctamente. Este mensaje también será utilizado en la comunicación entre procesos al momento de ejecutarse la fase de establecimiento de la conexión.

- **ncon:** permite al protocolo de comunicación enviar al usuario una confirmación negativa en caso de que la conexión no se estableció correctamente. Este mensaje también será utilizado en la comunicación entre procesos al momento de ejecutarse la fase de establecimiento de la conexión.

Las reglas y procedimientos para el protocolo de comunicación son:

- El usuario debe iniciar el servidor (brazo robot), el mismo que será capaz de aceptar una sola conexión debido a que el protocolo de comunicación tiene una configuración de línea punto a punto.

- El usuario desde el cliente (teléfono celular) solicitará el establecimiento de la conexión al servidor.

- Una vez establecida o no la conexión, el protocolo de comunicación informará al usuario del acontecimiento.

- Si se establece la conexión, el usuario podrá transmitir mensajes hacia el brazo robot, caso contrario, el protocolo de comunicación finalizará e informará al usuario del acontecimiento.

- Si el usuario transmite un mensaje desde el teléfono celular y llega correctamente al brazo robot, este deberá enviar un *ACK*, caso contrario enviará un *NACK.*

- Si llega un *ACK*, el usuario podrá enviar el siguiente mensaje, caso contrario, si llega un *NACK* o un acuse de recibo con error el teléfono celular no permitirá al usuario enviar otro mensaje y realizará una retransmisión. Por defecto se realizará dos retransmisiones para que el brazo robot, en presencia de errores, reaccione a tiempo durante la comunicación.

- Si no se recibe un *ACK* luego de haber realizado todas las retransmisiones posibles del mensaje, el protocolo de comunicación informará al usuario del acontecimiento y finalizará la conexión.

- El brazo robot podrá enviar mensajes de color cuando su pulsador toque una pelota y realice la lectura del color de la misma mediante su sensor de luz. Una vez realizada la lectura, este procederá a tomar la pelota dependiendo del color que se configuró previamente por el usuario. El color por defecto será rojo.

- El usuario puede configurar el color de la pelota que el brazo robot deberá capturar cuando lo desee mediante el teléfono celular.

- El temporizador se activará cuando se envíen mensajes de ejecución de comando. Si llega un *ACK,* este temporizador se detendrá inmediatamente, caso contrario si llega un *NACK,* un acuse de recibo con errores o no llega ningún acuse de recibo se realizará las retransmisiones correspondientes.
  Se debe tomar en cuenta que también se pueden recibir mensajes de color rojo o azul, los mismos que el temporizador podría confundir con un mensaje que contenga errores, por lo tanto, se debe clasificar los mensajes recibidos para que el temporizador no realice retransmisiones innecesarias.

- El usuario podrá finalizar la conexión en cualquier instante, ya sea desde el teléfono celular o desde el brazo robot. En cualquiera de los casos, el protocolo de comunicación informará al usuario del acontecimiento.

- Si se pierde la conexión, el protocolo de comunicación informará del acontecimiento al usuario por medio del teléfono celular.

Los procesos que intervienen en el modelo de validación del protocolo de comunicación para describir las reglas y procedimientos del mismo son los siguientes:

- ***USR***

  Este proceso realiza las siguientes tareas *(Ver ANEXO B.1)* :

  - Inicia el servidor del brazo robot. Si no se provocó ningún error, este proceso desplegará en pantalla *st1*, de lo contrario desplegará *err1*.

  - Si se obtuvo *st1* en el paso anterior, procede a iniciar el cliente del teléfono celular. Similar al paso anterior desplegará en pantalla *st2* ó *err2* si se inició o no el cliente respectivamente.

- En caso de que los dos equipos se iniciaron con éxito, procede a realizar la petición de establecimiento de conexión. Si ambos equipos se conectaron con éxito se desplegará en pantalla *st3* y *st4*, caso contrario *err3* y *err4*.

- Si los equipos se conectaron con éxito, este proceso realiza una petición de transmisión, caso contrario finaliza la simulación.

- ***PR/PM***

  - Estos procesos son los encargados de iniciar los *SAPs* del brazo robot y del teléfono celular respectivamente. Ambos procesos son instanciados por el proceso *USR* antes de que este realice la petición de conexión *(Ver ANEXO B.2 y ANEXO B.3).*

- ***SAPR/SAPM***

  Estos procesos realizan las siguientes tareas *(Ver ANEXO B.4 y ANEXO B.5)*:

  - Representan los *SAPs* del brazo robot y del teléfono celular respectivamente.

  - Reciben todas las peticiones del proceso *USR* y devuelve las respectivas confirmaciones (primitivas).

  - Cuando recibe una petición de establecimiento de conexión instancia los procesos *SECR* y *SECM*. Una vez establecida la conexión, *SAPR* y *SAPM* instancian los procesos *RXR* y *RXM* para permitir la recepción de datos en el brazo robot y en el teléfono celular respectivamente.

- Si reciben una petición de transmisión, *SAPR*/*SAPM* instancian los procesos *TXR* y *TXM* para transmitir un mensaje desde el brazo robot y el teléfono celular respectivamente.

- Cuando finaliza la simulación de la fase de transferencia de datos, se realiza una petición automática de finalización de la conexión. Se utiliza el proceso *FC* para este propósito.

- ***SECR/SECM***

  Estos procesos realizan las siguientes tareas *(Ver ANEXO B.6 y ANEXO B.7)*:

  - Estos procesos simulan el establecimiento de la conexión mediante la propiedad del no determinismo de *SPIN*. Los procesos eligen aleatoriamente si se conectan o no ambos equipos.

  - Informan el estado de la fase del establecimiento a su respetivo *SAP*.

- ***RXR/RXM***

  Estos procesos realizan las siguientes tareas *(Ver ANEXO B.8 y ANEXO B.9)*:

  - Permiten la recepción de los datos en el brazo robot y en el teléfono celular respectivamente.

  - Si *RXR* recibe un mensaje, instanciará al proceso *TXR* para devolver una respuesta. La respuesta se recibe en el proceso *RXM*.

  - *RXM* se encarga de simular el método de control de flujo *stop and wait*.

- ***TXR/TXM***

  Estos procesos realizan las siguientes tareas *(Ver ANEXO B.10 y ANEXO B.11)*:

  - Son procesos que solo existen mientras se transmite un mensaje.

  - *TXR* es el encargado de simular el no determinismo del canal de comunicación y también el envío del resultado de la lectura de un color por parte del brazo robot.

  - *TXM* es el encargado de transmitir cualquier mensaje proveniente del teléfono celular.

- ***FC***

  - Este proceso es instanciado automáticamente una vez que se haya finalizado la simulación de la fase de transferencia de datos *(Ver ANEXO B.12)*.

Una vez que se han establecido los tipos de mensajes, las reglas y procedimientos para el intercambio de datos para el modelo del protocolo de comunicación, se procedió a la escritura del programa en *PROMELA*.

Después de haber simulado en varias ocasiones el programa, se concluyó que las reglas y procedimientos especificados anteriormente no contienen errores que puedan provocar un mal funcionamiento del protocolo de comunicación. El programa se encuentra en el *ANEXO B*.

4.1.2.3.2. Planeamiento del protocolo de comunicación

Las funciones que debe cumplir el protocolo de comunicación estarán agrupadas en diferentes paquetes, los mismos que se describen a continuación:

- **protocoloRAP.SAP:** este paquete contiene la clase que permitirá el acceso a los servicios del protocolo de comunicación y es la raíz para los demás paquetes. (Ver ANEXO C.7 y ANEXO D.7)

- ***.servicios.conexion[34]:** este paquete brindará el servicio de establecimiento de la conexión. (Ver ANEXO C.10 y ANEXO D.10)

- ***.canalRFCOMM:** una vez que se estableció la conexión, la clase que se encuentra dentro de este paquete será la encargada de configurar el canal de transmisión serial. (Ver ANEXO C.8 y ANEXO D.8)

- ***.servicios.transmision:** los servicios de transmisión y recepción se encuentran encapsulados en este paquete. (Ver ANEXO C.11 Y ANEXO D.14)

- ***.servicios.OperacionesTramas:** en este paquete se encuentra todas las clases y funciones necesarias para las operaciones que se realizarán en las tramas que se enviarán y en las tramas que arribarán. (Ver ANEXO C.9 y ANEXO D.9)

- ***.servicios.escuchaBotones:** las clases de este paquete permitirán manipular los eventos que produzca el usuario cuando presiones los botones del *NXT.* (Ver ANEXO D.11)

[34] Se utilizará el símbolo * para representar a un paquete raíz

- ***.servicios.escuchaSensores:** este paquete permite la lectura de los valores adquiridos por los sensores ya sean de luz o de tacto. (Ver ANEXO D.12)

4.1.2.3.3. Planeamiento de la interfaz gráfica de usuario

La interfaz gráfica de usuario, como su nombre lo indica, comprende todos los procesos que van a interactuar con el usuario a través del teléfono celular. Esta interfaz estará escrita en la plataforma *J2ME,* la misma que contiene pocos elementos gráficos debido a las características de los sistemas embebidos.

Los paquetes y sus funciones se describen a continuación:

- **AppRobot.midRobot:** este paquete contendrá la *Midlet*, la misma que sirve para controlar el ciclo de vida de la interfaz gráfica y es la raíz para los demás paquetes. (Ver ANEXO C.1)

- ***.utileria:** contendrá todas las clases necesarias que representan a los elementos que se utilizarán en la interfaz gráfica, como son formularios, cajas de texto, etc. (Ver ANEXO C.2)

- ***.utileria.interfaces:** contendrá las interfaces de la interfaz gráfica, las mismas que encapsularán los métodos y constantes comunes a todas las clases del paquete *.utilería. (Ver ANEXO C.3)

- ***.utileria.pantallas:** este paquete contendrá todas las pantallas que se muestran al usuario, creadas a partir de todas las clases del paquete utilería. (Ver ANEXO C.4)

- ***.utileria.pantallas.peticionTxListeners:** este paquete contendrá un hilo especial el mismo que permitirá la petición de transmisión de mensajes al protocolo de comunicación. (Ver ANEXO C.5)

4.1.2.3.4. Planeamiento del programa para el NXT brick

El programa del *NXT brick* permitirá el control de los sensores y servomotores conectados a él. Debido a que no se dispone de los elementos gráficos de *J2ME*, se presentarán informes al usuario a través de una salida de pantallas básica.

Los paquetes y sus funciones se describen a continuación:

- **AppRobot.principal:** este paquete contendrá la clase *Main,* la misma que permitirá instanciar los objetos a partir de las clases que se encuentren debajo de este paquete raíz. (Ver ANEXO D.1)

- **AppRobotServer.control.sensores:** dentro de este paquete se encontrará la clase que permitirá el control del sensor de luz y el pulsador del brazo robot. (Ver ANEXO D.3)

- **AppRobotServer.control.servos:** este paquete contendrá las clases necesarias que permitirán regular la velocidad de los tres servomotores y también el control de los mismos. (Ver ANEXO D.4)

- **AppRobotServer.utileria.pantallas:** las pantallas que permitirán la interacción del usuario con el brazo robot se encontrarán encapsuladas en este paquete. (Ver ANEXO D.6)

4.1.2.4. Diseño del Sistema

4.1.2.4.1. Diseño del Protocolo de Comunicación [48]

El protocolo de comunicación deberá ser diseñado tomando en cuenta principalmente la propiedad de modularidad. Cada servicio que éste ofrecerá deberá corresponder a una clase o a un método, dependiendo de la flexibilidad para futuros cambios.

La Figura 4.4 muestra como se podría implementar el protocolo de comunicación utilizando la propiedad de modularidad.

Para el brazo robot se requiere el control de *hardware* específico como son los servomotores y sensores, además de los servicios antes mencionados.

La Figura 4.5 muestra los servicios del protocolo de comunicación a implementar en el brazo robot.

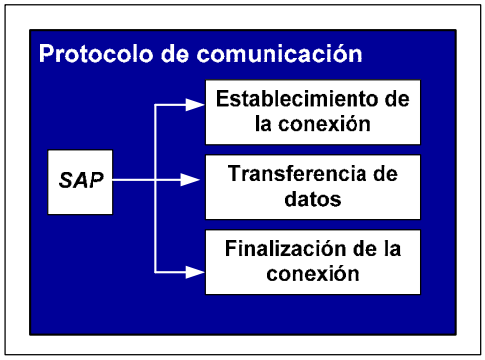


Figura 4.4 Protocolo de comunicación utilizando la propiedad de la modularidad de *Java*

Para el teléfono celular, se tiene únicamente que controlar específicamente su pantalla y los eventos cuando se presionan sus botones como se muestra en la Figura 4.6.

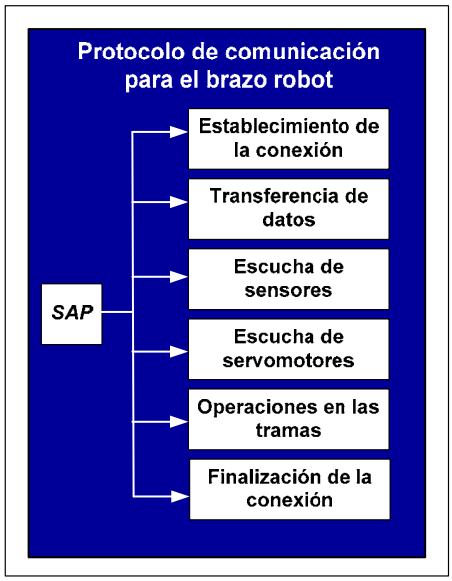


Figura 4.5 Servicios del protocolo de comunicación que se implementarán en el brazo robot

Cada servicio del protocolo de comunicación estará representado por una clase y estarán agrupados según se planeo anteriormente. Debido a que algunos servicios pueden ser hilos de ejecución, la clase protocolo de comunicación será una clase *singleton*[35] para evitar la clonación innecesaria de hilos de ejecución y se muestra en la Figura 4.7.

Encapsulada en la clase *singleton* protocolo de comunicación se encuentra la clase *PuntoAccesoServicio*, la misma que está compuesta por todos los servicios que puede brindar el protocolo de comunicación.

Para solicitar un servicio a la clase *PuntoAccesoServicio*, se necesita del manejo de las primitivas. Para implementar las primitivas se utilizó el método *CubbyHole*. Este método implementa un *slot* que puede tener solo un objeto a la vez para la comunicación entre hilos. Un hilo coloca un objeto en el *slot* y otro lo toma a través de dos métodos. Si otro hilo trata de poner un objeto en el *slot* cuando este se

35 Singleton: Es un patrón de diseño que garantiza que un sistema cree, como máximo, la instancia de un objeto de una clase.

encuentra ocupado, este hilo se bloquea hasta que el *slot* se libere. De esta manera, cualquier clase podrá solicitar un servicio al *SAP* de una manera segura (Ver ANEXO C.7.1 y ANEXO D.7.1).

Para la petición (colocación del objeto petición en el *slot*) y confirmación (liberación del *slot*) de servicios utilizando el método *CubbyHole* se utilizan las siguientes instrucciones:

```
Protocolo.obtenerSAP().peticionServicio(byte  servicio);
boolean conf = Protocolo.obtenerSAP().confirmacionServicio();
```

Los mensajes del protocolo de comunicación utilizarán un formato de mensaje con cabeceras y colas, debido a que se utilizará un módulo *Bluetooth* el mismo que implementa funcionalidades de capas inferiores.

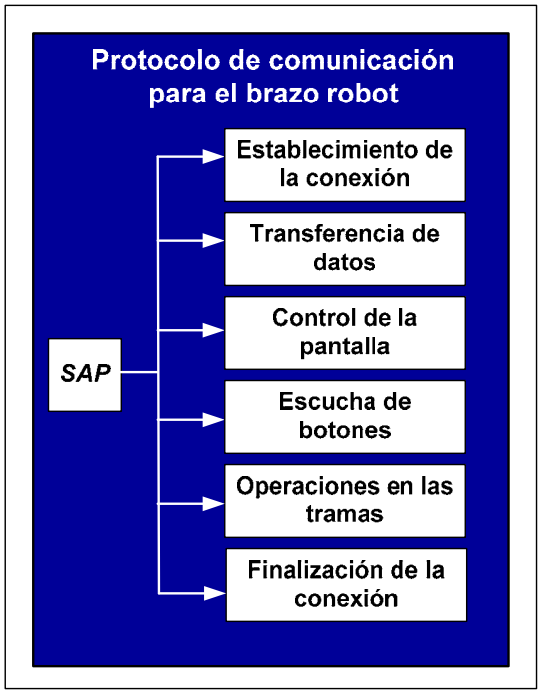


Figura 4.6 Servicios del protocolo de comunicación que se implementarán en el teléfono celular

```
1. public final class Protocolo {
2.
3. private static final Protocolo protocoSingleton = new Protocolo();
4. private static PuntoAccesoServicio sap;
5.
6. private Protocolo(){
7.          Protocolo.sap = new PuntoAccesoServicio(true);
8.  }
9.
10. public static Protocolo obtenerInstanciaProtocolo() {
11.         return protocoSingleton;
12.  }
13.
14.  public static PuntoAccesoServicio obtenerSAP() {
15.          return Protocolo.sap;
16. }
17.
18. }
```

Figura 4.7 Clase *singleton* del protocolo de comunicación

Los mensajes se clasificarán en:

- **Mensajes de datos:** que pueden ser de ejecución de comando, color, y fin de conexión.
- **Mensajes de control:** que pueden ser de acuses de recibo positivos (*ACK*) o negativos (*NACK*) para cada uno de los tipos de mensajes de datos.

La información que conformará la cabecera estará definida por los siguientes criterios y se muestra en la Figura 4.8:

- Un bit *Tp* para identificar el tipo de mensaje .
- Un bit *Sec* para establecer el número de secuencia del mensaje para el control de flujo.
- Un bit *A/N* para identificar si se trata de un *ACK* o un *NACK.*
- Dos bits *Clase* para identificar el tipo de mensaje de control.
- Un bit *Tx* para identificar el origen del mensaje, debido a que un mensaje de datos podrá ser utilizado para distintos propósitos, dependiendo de quién envíe el mensaje.

- Un bit *F* para indicar cuándo se debe finalizar la conexión

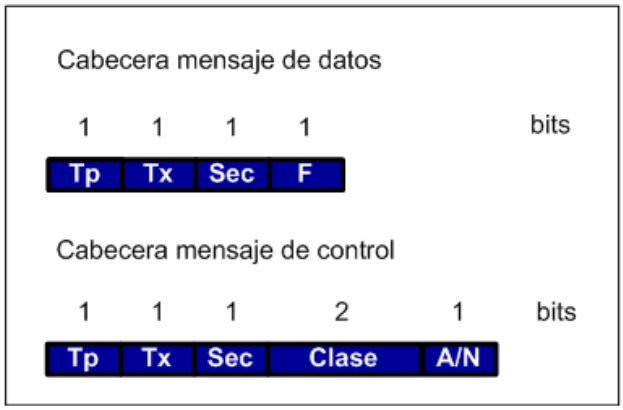


Figura 4.8 Cabeceras de los mensajes de datos y de control del protocolo de comunicación

Los valores de los bits que conforman la cabecera del protocolo de comunicación y su descripción se muestran en la Tabla 4.2.

Bit	Valor	Descripción
Tp	0	Mensaje de datos.
	1	Mensaje de control.
Tx	0	Transmite el teléfono celular.
	1	Transmite el brazo robot.
Sec	0	Número de secuencia.
	1	Número de secuencia.
F	0	No finaliza la conexión.
	1	Finaliza la conexión.
Clase	00	*NACK*..
	01	*ACK* de comando
	10	*ACK* de fin de conexión.
	11	*ACK* de mensaje de color.

Tabla 4.2 Valores y descripción de los bits que conformarán la cabecera del protocolo de comunicación.

Bit	Valor	Descripción
A/N	0	*ACK*
	1	*NACK*

Tabla 4.2 Valores y descripción de los bits que conformarán la cabecera del protocolo de comunicación. (Continuación)

La carga útil estará definida por el siguiente criterio:

- Cinco bits que se utilizarán para indicar el comando para la ejecución del movimiento, el color de la pelota leída por el brazo robot o configurar el color de la pelota por defecto para que el brazo robot realice la captura de la misma. Cuando los cinco bits de un mensaje de datos se encuentran en cero, significa que se trata de un mensaje de finalización de la conexión.

  La Figura 4.9 muestra la carga útil de los mensajes de datos y la información que representan y la Tabla 4.3 muestra los valores correspondientes que puede tomar la carga útil.

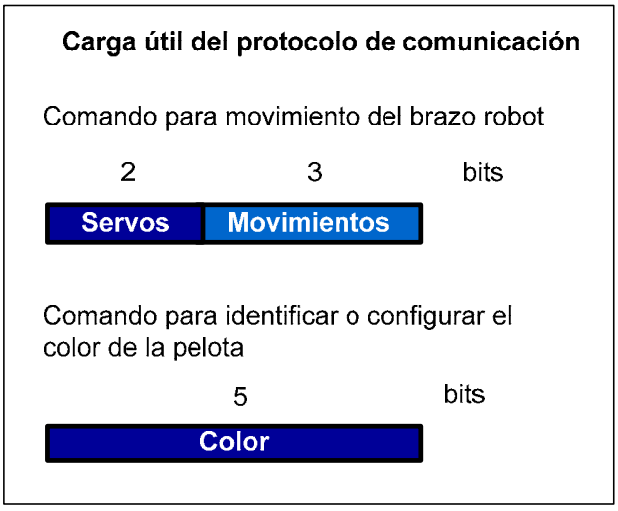


Figura 4.9 Carga útil de los mensajes de datos del protocolo de comunicación

La cola estará compuesta por la *FCS* del polinomio *CRC* y estará definida por los siguientes criterios:

- La eficiencia relativa E del protocolo de comunicación, en presencia de errores, según la Ecuación 4.1:

$$E = \frac{d}{R(d + tca + tco + a)}$$

Ecuación 4.1 Eficiencia relativa de un protocolo de comunicación en presencia de errores [1]

Las variables d, tca, tco y a de la Ecuación 4.1, representan el tamaño de la carga útil, el tamaño de la cabecera del mensaje de datos, el tamaño de la cola del mensaje de datos, y el tamaño del mensaje de control en bits del protocolo de comunicación respectivamente.

Bit	Valor	Descripción
Servos	01	Servo A
	10	Servo B
	11	Servo C
Movimiento	101	Mover servo hacia izquierda
	011	Parar servo
	100	Mover servo hacia derecha
	001	Mover servo hacia arriba
	010	Mover el servo hacia abajo
	110	Abrir la mano
	111	Cerrar la mano

Tabla 4.3 Valores y descripción de los bits que conformarán la carga útil del protocolo de comunicación

Bit	Valor	Descripción
Color	01010	Rojo
	01111	Azul

Tabla 4.3 Valores y descripción de los bits que conformarán la carga útil del protocolo de comunicación. (Continuación)

La variable R representa el número de transmisiones por mensaje y viene dada por la Ecuación 4.2:

$$R = \frac{1}{1 - pr}$$

Ecuación 4.2 Número de transmisiones por mensaje de un protocolo de comunicación [1]

La variable pr de la Ecuación 4.2 representa la probabilidad de que un mensaje sea retransmitido.

- El tamaño del mensaje de datos deberá ser múltiplo entero de un *byte* debido a la característica de los *streams* del lenguaje *Java.*

- La distancia de *Hamming.*

A continuación se realizará el cálculo de eficiencia relativa en función del tamaño de la cola considerando los criterios antes mencionados.

El valor del *BER* para dispositivos que soportan *Bluetooth* 2.0 con *EDR* es de 0.01%, el mismo que se considera para el cálculo de la variable *R* de la Ecuación 4.2:

$$R = \frac{1}{1 - 0.00001} = 0.999 \approx 1$$

Ecuación 4.3 Cálculo del valor de la variable R a partir de la ecuación 4.2

Nombre variable	Descripción	Valor
d	Bits de la carga útil	5
tca	Bits de la cabecera del mensaje de datos	4
tco	Bits de la cola del mensaje de datos	------
a	Bits del mensaje de control	6

Tabla 4.4 Variables y valores que corresponden a la cabecera, carga útil, cola y mensaje de control del protocolo de comunicación

Reemplazando los valores de la Tabla 4.4 y de la Ecuación 4.3 en la Ecuación 4.1 se tiene:

$$E = \frac{d}{R(d + tca + tco + a)} = \frac{5}{1(5 + 4 + tco + 6)} = \frac{5}{(15 + tco)}$$

Ecuación 4.4 Cálculo de la eficiencia relativa en función del tamaño de la cola a partir de la Ecuación 4.1

La variable tco representa el tamaño de la *FCS* a implementar. La Tabla 4.5 contiene un listado de los polinomios *CRC* más comunes para un tamaño de mensaje recomendado de datos:

El tamaño del mensaje recomendado para los polinomios de la Tabla 4.5 se deducen considerando la distancia de *Hamming* de cada polinomio *CRC* [75].

Nombre del polinomio	Orden polinomio	Tamaño mensaje recomendado en bits	Eficiencia %
CCITT-4	4	8 - 2048	26,32
CRC-5	5	8 - 10	25,00
DARC-6	6	12 - 25	23,81
DARC-8	8	8 - 9	21,74
CCITT-16	16	1015 - 2048	16,13

Tabla 4.5 Polinomios *CRC* y la eficiencia para el protocolo de comunicación utilizando la Ecuación 4.3 [32]

El polinomio *CCITT*-4 ofrece un buen rendimiento para tramas de diversos tamaños, y además una alta eficiencia debido a los cuatro bits de redundancia que ofrece, por tal razón éste será usado como polinomio generador de *CRC*.

En la Figura 4.10 se muestra el mensaje de datos considerando los bits de *CRC* (*tco*) y el mensaje de control.

Los mensajes de datos y de control estarán conformados entonces por trece y seis bits respectivamente como se muestra en la Figura 4.10, sin embargo, debido a que los *streams* de *bytes* del lenguaje *Java* están orientados a bloques de ocho bits, se debe añadir tres y dos bits de relleno a cada mensaje respectivamente.

La Figura 4.11 muestra un mensaje de datos y su respectivo mensaje de control después de añadirse los bits de relleno.

Una vez obtenida la variable tco y considerando los bits de relleno de los mensajes de datos y de control se obtienen los valores que se muestran en la Tabla 4.6.

Por el momento, los bits de relleno del mensaje de datos no cumplen una función específica, sin embargo es posible que en futuras revisiones estos bits se incluyan como parte de su cabecera o de su carga útil según se vea la necesidad.

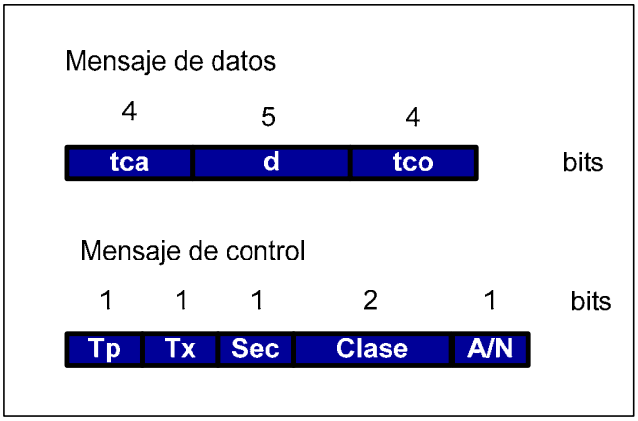


Figura 4.10 Tamaño del mensaje de datos y de control

Nombre variable	Descripción	Valor
d	Bits de la carga útil	5
tca	Bits de la cabecera del mensaje de datos	4
tco	Bits de la cola del mensaje de datos	4
a + rc	Bits del mensaje de control con relleno	8
rd	Bits de relleno para el mensaje de datos	3

Tabla 4.6 Variables y valores que corresponden a la cabecera, carga útil, cola y mensaje de control del protocolo de comunicación considerando los bits de relleno

Para el cálculo de la eficiencia relativa se considera ambas posibilidades y se muestran a continuación:

- Si se considera los bits de relleno del mensaje de datos como parte su carga útil del mensaje de datos y se reemplaza en la Ecuación 4.1 se tiene:

$$E = \frac{d}{R(d + tca + tco + a)} = \frac{8}{1(8 + 4 + 4 + 8)} = 0.333$$

Ecuación 4.5 Cálculo de la eficiencia relativa considerando los bits de relleno como parte de la carga útil a partir de la Ecuación 4.1

- Si se considera los bits de relleno del mensaje de datos como parte de su cabecera y se reemplaza en la Ecuación 4.6 se tiene:

$$E = \frac{d}{R(d + tca + tco + a)} = \frac{5}{1(5 + 7 + 4 + 8)} = 0.208$$

Ecuación 4.6 Cálculo de la eficiencia relativa considerando los bits de relleno como parte de la cabecera a partir de la Ecuación 4.1

La eficiencia relativa del protocolo de comunicación entonces, debido a los bits de relleno del mensaje de datos podría variar entre el 20.8 % y el 33.3%.

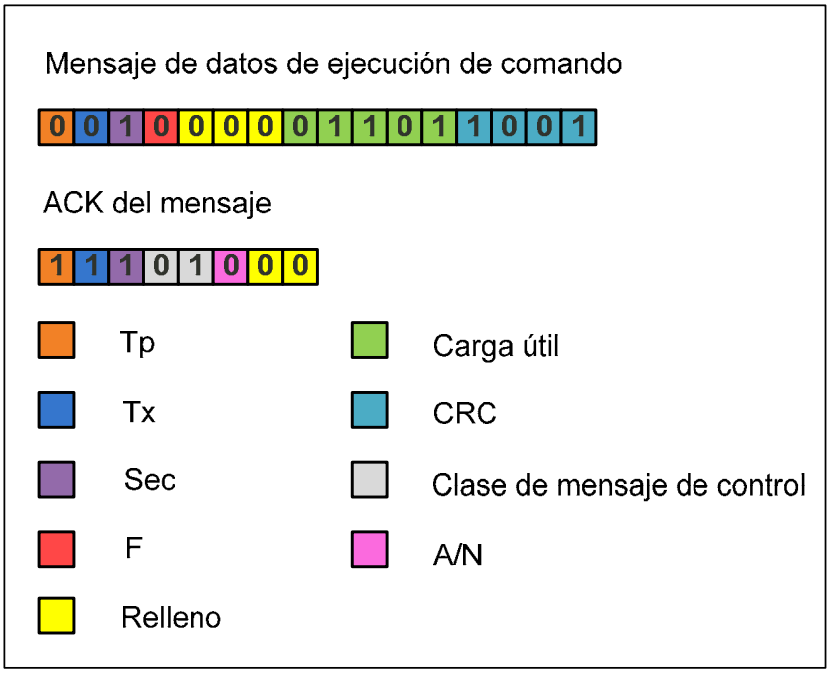


Figura 4.11 Ejemplo de dos tipos de mensajes del protocolo de comunicación

4.1.2.4.2. Diseño de la interfaz gráfica de usuario

La interfaz gráfica debe permitir el acceso a los servicios del protocolo de comunicación para el control del brazo robot, por tal razón las pantallas que permitirán cumplir con este objetivo son:

- **Presentación:** muestra información respecto al proyecto.

- **Búsqueda:** permitirá el ingreso del nombre del brazo robot requerido por *Bluetooth* para iniciar la fase del establecimiento de la conexión.

- **Espera Servidor:** esta pantalla se muestra mientras se realiza el proceso de establecimiento de la conexión.

- **Control Robot:** esta pantalla contiene los controles que permiten el control del brazo robot y aparecerá una vez que se halla establecido la conexión.

- **Configuración Color:** esta pantalla permite la selección del color de la pelota por parte del usuario para que el brazo robot la tome o no después de identificar el color.

- **Espera:** esta pantalla se muestra cuando un acuse de recibo no llegue en un determinado tiempo.

- **Log:** mantiene un historial de los mensajes enviados y recibidos en el teléfono celular en decimal y en binario.

Las Figuras 4.12, 4.13 y 4.14 muestran las pantallas *Presentación*, *Búsqueda* y *Espera Servidor* respectivamente.

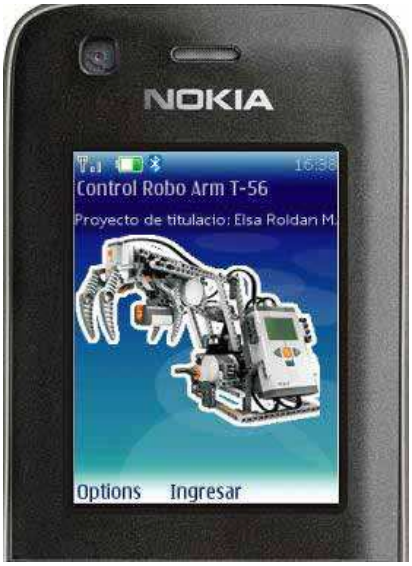


Figura 4.12 Pantalla Presentación del teléfono celular

Figura 4.13 Pantalla Búsqueda del teléfono celular

Figura 4.14 Pantalla Espera Servidor del teléfono celular

La pantalla *Presentación* contiene el botón *ingresar*, el mismo que permite al usuario avanzar hacia la pantalla *Búsqueda*. En la pantalla Búsqueda el usuario deberá ingresar el nombre requerido por *Bluetooth* que se configuró en el brazo robot. Una vez ingresado el nombre se pulsa el botón *buscar* y de inmediato el protocolo de comunicación llama al servicio de establecimiento de la conexión y aparecerá la pantalla *Espera Servidor*.

Las Figura 4.15 y 4.16, muestran la pantalla de *Control Robot* y sus opciones que son *Log* y *Conf* respectivamente. La pantalla *Control Robot* aparece una vez que la conexión se ha establecido con éxito y el protocolo de comunicación está listo para la transferencia de mensajes, los mismos que permitirán el control de los movimientos del brazo robot.

Cuando se selecciona la opción *Conf* de la pantalla *Control Robot* aparecerá la pantalla *Configuración Color* que se muestra en la Figura 4.17, la misma que permite al usuario configurar el color de la pelota por defecto que el brazo robot deberá tomar

cuando identifique un color. Las opciones de la pantalla *Configuración Color* se muestran en la Figura 4.18.

Figura 4.15 Pantalla Control Robot del teléfono celular

Figura 4.16 Opciones de la pantalla Control Robot del teléfono celular

Figura 4.17 Pantalla Configuración Color del teléfono celular

Figura 4.18 Opciones de la pantalla Configuración Color del teléfono celular

Cuando se selecciona la opción *Log* de la pantalla *Control Robot* aparecerá la pantalla que se muestra en la Figura 4.19, la misma que permite al usuario observar las tramas enviadas y recibidas en el teléfono celular en binario y en decimal. Esto ayuda al usuario medir la interferencia presente en el ambiente.

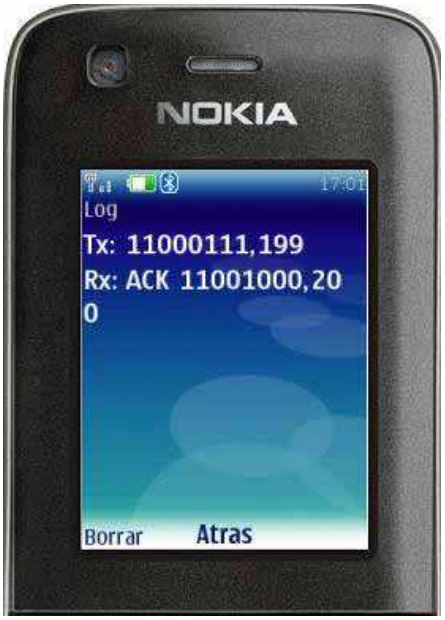


Figura 4.19 Pantalla *Log* del teléfono celular

La Figura 4.20 muestra la pantalla que aparecerá una vez que se haya finalizado la conexión. Cuando esto sucede la aplicación se cierra de inmediato liberando todos los recursos utilizados por la aplicación.

Figura 4.20 Pantalla por defecto de fin de la conexión y de la aplicación del teléfono celular

En ocasiones el usuario puede ingresar un nombre incorrecto del brazo robot, ó también puede que el servidor no se encuentre disponible. La Figura 4.21 muestra el proceso que sigue la interfaz gráfica cuando esto sucede. Se muestra una alerta informando al usuario que no se pudo establecer la conexión con el teléfono celular y luego de un tiempo determinado se cierra la aplicación automáticamente.

Figura 4.21 Pantalla de error que se muestra cuando no se puedo establecer la conexión

4.1.2.4.3. *Diseño del programa para el NXT brick*

El programa para el *NXT brick* tiene como objetivo permitir al usuario el inicio del servidor *Bluetooth*, el protocolo de comunicación y la configuración de los sensores y servomotores (Ver ANEXO D.4). Para esto se necesita de las siguientes pantallas:

- **Presentación:** muestra información respecto al proyecto.

- **Menú:** permite al usuario iniciar el servidor.

- **Espera Conexión:** esta pantalla se muestra mientras se realiza el proceso de establecimiento de la conexión.

- **Información Conexión:** esta pantalla se muestra cuando se estableció la conexión con éxito.

- **Log:** mantiene un historial de los mensajes enviados y recibidos en el brazo robot en decimal y en binario.

- **Fin Conexión:** informa al usuario que la conexión finalizó con éxito

La captura de pantallas para cada fase se muestra a continuación:

Figura 4.22 Pantalla Presentación del brazo robot

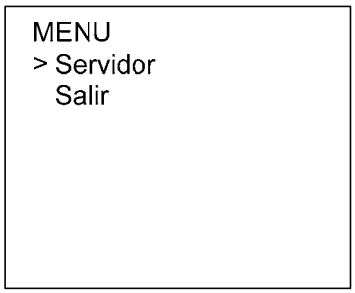


Figura 4.23 Pantalla Menú del brazo robot

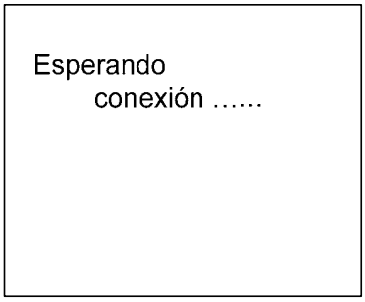


Figura 4.24 Pantalla Espera Conexión del brazo robot

Las Figuras 4.22, 4.23 y 4.24 muestran las pantallas de *Presentación*, *Menú* y *Espera Conexión* respectivamente.

La pantalla *Presentación* muestra la información básica del proyecto. Para continuar hacia la pantalla *Menú* para iniciar el servidor, se deberá presionar el botón *ENTER* del *NXT brick.* Una vez la pantalla *Menú* el usuario podrá elegir entre la opción *Servidor* y *Salir*. Si se elige la opción *Servidor*, el protocolo de comunicación se inicia automáticamente y llama al servicio de establecimiento de la conexión para aceptar la conexión con el cliente. Mientras el brazo robot espera al cliente se mostrará la pantalla *Espera Conexión* de la Figura 4.24.

La Figuras 4.25 y 4.26 muestran las pantallas *Información Conexión* y *Log* respectivamente.

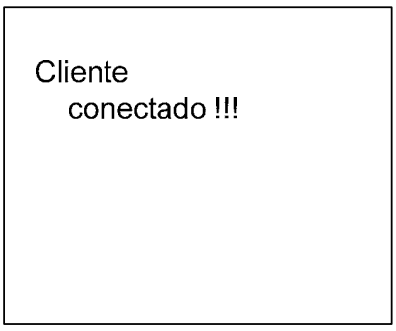


Figura 4.25 Pantalla Información Conexión del brazo robot

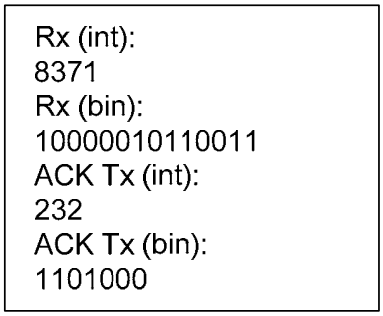


Figura 4.26 Pantalla *Log* del brazo robot

La pantalla de *Información Conexión* se muestra una vez que se haya establecido la conexión con el cliente. Si el cliente comienza a enviar mensajes, entonces se mostrará de inmediato la pantalla *Log*, con la información del mensaje recibido y el acuse de recibido transmitido que proporciona el protocolo de comunicación.
La pantalla *Fin Conexión* que se muestra en la Figura 4.27 aparece en los siguientes casos:

- Si el usuario finaliza la conexión desde el teléfono celular.
- Si el usuario finaliza la conexión desde el brazo robot.

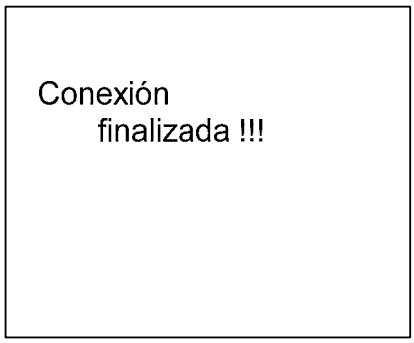


Figura 4.27 Pantalla Fin Conexión del brazo robot

Para finalizar la conexión desde el brazo robot, el usuario deberá presionar el botón *ENTER* del *NXT brick* mientras se encuentre en la pantalla *Log*. Al presionar el botón *ENTER* se está realizando una petición de finalización al protocolo de comunicación.

Si la conexión finalizó entonces se muestra la pantalla *Fin Conexión*. Para regresar al menú principal de sistema operativo *leJOS* se deberá presionar el botón *ESCAPE* del *NXT brick*.

Las consideraciones respecto a la configuración de los sensores y servomotores del brazo robot fueron las siguientes:

- El pulsador y el sensor de luz estarán conectados en el puerto *S4* y *S3* del *NXT brick* respectivamente.

- El sensor de luz se encenderá para realizar la lectura del color de la pelota únicamente cuando se presione el pulsador. Se controló el anti rebote que se origina mientras el pulsador se encuentra presionado.

- Los servomotores A y B se configuraron para que alcancen una rapidez de 720 *RPM*, mientras que el servomotor C alcanza una rapidez de 360 *RPM.*

4.2. PRUEBAS Y RESULTADOS DEL SISTEMA

Las pruebas consistieron en verificar el correcto funcionamiento de los principales servicios del protocolo de comunicación que se muestran en la Figura 4.4. Las pruebas se realizaron por cinco ocasiones a diferentes distancias, las mismas que se muestran en las Tablas 4.8 y 4.9, con cada uno de los celulares en presencia de dos equipos que se encontraban transmitiendo datos continuamente utilizando el estándar 802.11g. Los equipos con los que se realizaron las pruebas fueron un *router* y un adaptador de la marca *Linksys*. En la Tabla 4.7 se muestran las características los equipos inalámbricos.

El *router* y el adaptador inalámbrico se encontraban conectados a dos computadores respectivamente, lo que permitió el envío de un archivo de video con un tamaño de

4,35 Gbytes desde un computador a otro. De esta manera se generó el ambiente de interferencia para realizar las pruebas al protocolo de comunicación.

Equipo	Modelo	Potencia de salida (dBm)	Estándar	Velocidad de transmisión máxima
Router inalámbrico	WRT54GL	18	802.11b/g	54 Mbps
Adaptador inalámbrico	WUSB54GC	14	802.11b/g	54 Mbps

Tabla 4.7 Características de los equipos utilizados para realizar las pruebas de interferencia

Los resultados que se obtuvieron se encuentran tabulados en la Tabla 4.8 y Tabla 4.9.

Se puede concluir que mientras mayor sea la distancia a la que se transmite datos entre los teléfonos celulares y el brazo robot, la eficiencia relativa del protocolo disminuye debido al número de retransmisiones.

Otro factor que puede alterar estos resultados es la presencia de equipos que transfieran datos en la misma banda de frecuencia de *Bluetooth* (*routers*, *Access points* inalámbricos, etc.) dentro del ambiente acción del protocolo de comunicación.
Se debe tomar en cuenta que los equipos con los que se trabajó son dispositivos *Bluetooth* clase 2, por lo que están diseñados para trabajar correctamente a una distancia máxima de 10 metros, por tal razón se presentan fallas considerables en las fases del protocolo de comunicación a distancias mayores que las permitidas como se muestran en las Tablas 4.8 y 4.9.

Distancia	Fase 1[36]	Fase 2[37]	Fase 3[38]
(metros)	(%Exitoso)	(%Exitoso)	(%Exitoso)
9	100	100	100
10	100	100	100
11	100	100	100
12	100	100	100
13	100	100	100
14	100	100	100
15	100	100	100
16	100	100	100
17	100	100	100
18	60	40	60
19	40	30	40

Tabla 4.8 Resultado de las pruebas realizadas a las fases del protocolo de comunicación con el teléfono celular *Nokia* 5220

Distancia	Fase 1	Fase 2	Fase 3
(metros)	(%Exitoso)	(%Exitoso)	(%Exitoso)
9	100	100	100
10	100	100	100
11	80	80	80
12	80	80	60
13	100	100	100
14	80	80	80
15	80	80	80
16	80	80	80
17	40	60	40
18	40	60	40
19	20	20	20

Tabla 4.9 Resultado de las pruebas realizadas a las fases del protocolo de comunicación con el teléfono celular *Sony Ericcson* w580

[36] Fase 1: corresponde al servicio de establecimiento de la conexión.
[37] Fase 2: corresponde al servicio de transferencia de datos.
[38] Fase 3: corresponde al servicio para la finalización de la conexión.

4.3. COSTOS REFERENCIALES

La Tabla 4.10 y 4.11 muestran los costos de cada de uno de los elementos de *hardware* y *software* que conforman el sistema implementado.

Cantidad	Elemento	Marca	Costo incluido *IVA* (*USD*)
1	Cargador de baterías AA para el brazo robot	*Sony*	28,00
3	Par de baterías AA para el brazo robot	*Sony*	28,56
1	*Bluetooth USB DBT-122*[39]	*D-Link*	43,68
1	*Kit Lego Mindstorm NXT 1.0*	*Lego*	460,00
		Total:	560.24

Tabla 4.10 Listado de los elementos de *hardware* que intervinieron en el proyecto y sus respectivos costos.

Función del profesional	Número de horas trabajadas	Costo por hora mínimo[40] ($)	Costo incluido *IVA* (*USD*)
Programador Sénior	360	1.87	753.98
Diseñador de Software	280	*1.90*	595.84
		Total:	1349.82

Tabla 4.11 Costo total del *software* implementado [72]

Componentes	Costo incluido *IVA* (*USD*)
Hardware	560.24
Software	1349.82
Total:	1910.06

Tabla 4.12 Costo total del proyecto

[39] Se adquirió este dispositivo *Bluetooth* debido a que se encuentra dentro de la lista recomendada por la empresa *Lego Mindstorms*. [67]

[40] Estos valores fueron obtenidos de acuerdo a los salarios mínimos sectoriales (sector tecnología) establecidos por el Ministerio de Relaciones Laborales [72]. (Ver Anexo H).

CAPÍTULO 5: CONCLUSIONES Y RECOMENDACIONES

5.1. CONCLUSIONES

- Para la implementación del sistema se optó por la metodología de desarrollo de software con ciclo de vida en espiral ya que ofrece un mejor enfoque cuando el responsable del desarrollo del *software* está inseguro de la eficacia de un algoritmo o de la forma que debería tomar la interacción humano-máquina además de su fortaleza en el análisis de riesgos para cada prototipo que se implementa, lo que permite que el resultado sea un *software* robusto.

- La etapa más importante en el diseño de un protocolo de comunicación es la especificación de sus reglas y procedimientos debido a que está íntimamente relacionada con la etapa de verificación e implementación del mismo.

- La especificación de la hipótesis del medio permitió analizar las características de *software* y *hardware* de los equipos que componen el ambiente del protocolo de comunicación. Este análisis permite al diseñador conocer la compatibilidad que existe entre estos equipos para poder realizar la comunicación.

- La especificación de los servicios del protocolo de comunicación dependen de los dispositivos de comunicación que se utilizará para el intercambio de datos, debido a que estos dispositivos ya implementan funcionalidades de capas inferiores del modelo *OSI*.

- El análisis de riesgos permitió describir las áreas que se utilizarán en detalle para la implementación del protocolo de comunicación así como factores importantes para optimizar la aplicación.

- La herramienta que facilitó el desarrollo de la aplicación para los teléfonos celulares fue el simulador *Nokia SDK* de la serie 40 que cuenta con un *NCF* (*Nokia Conectivity Framework*) ya que permitió simular la comunicación inalámbrica *Bluetooth* y realizar las pruebas de la aplicación antes de que sea instalada en los teléfonos celulares.

- El *IDE Netbeans* permite a los programadores editar, compilar, depurar y ejecutar programas *Java*. Además cuenta con un *plugin* que ayuda a ejecutar la aplicación *J2ME* directamente sobre el simulador *Nokia SDK* de la serie 40 y otro para transferir los programas compilados a través de *Bluetooth* hacia los equipos.

- Los servicios del protocolo de comunicación fueron diseñados tomado en cuenta principalmente la propiedad de modularidad lo que permitirá realizar futuros cambios ya que servicio corresponde a una clase de *Java.*

- Se necesitaron hilos de ejecución para implementar varios de los servicios que ofrece el protocolo de comunicación, por tal razón la clase que los invoca fue implementada como *singleton* para evitar la clonación innecesaria de hilos.

- Para el diseño de los mensajes del protocolo de comunicación se utilizó un formato de cabecera y cola debido a que este formato es el común para protocolos de capas superiores. La cabecera incluye información de control como por ejemplo el número de secuencia requerido por el método *stop and wait*, tipos de mensaje, y estado de la conexión, etc. La carga útil está compuesta por el servomotor junto con el movimiento a realizar o también puede llevar el color de la pelota identificado por el brazo robot. La cola está compuesta por la *FCS* del polinomio *CRC*, el mismo que fue seleccionado en base a la eficiencia relativa del protocolo de comunicación, la distancia de *Hamming* para polinomios estandarizados y la característica de los *streams* de *Java*.

- El ambiente de un protocolo de comunicación no solo está compuesto por los equipos que conforman el sistema distribuido, sino también por los agentes externos, como son los usuarios que manipularán dichos equipos.

- La medida que tome el sensor de luz del brazo robot *Lego Mindstorms* depende de varios factores, principalmente del color, la textura y la distancia a la que se entre la superficie sobre la cual se va a obtener la medida.

- Debido a las pruebas que se realizaron al sistema se concluye que mientras mayor sea la distancia a la que se transmiten datos entre los teléfonos celulares y el brazo robot en la presencia de equipos que trabajen en la misma banda de frecuencia de *Bluetooth* (*routers*, *Access points* inalámbricos, etc.), la eficiencia relativa del protocolo disminuye debido al número de retransmisiones.

5.2. RECOMENDACIONES

- Mantener al brazo robot apagado mientras no se lo utilice, debido a que los procesos concurrentes que se ejecutan continuamente producen un alto consumo de batería.

- Si se va a manipular el brazo robot con el teléfono *Sony Ericcson*, se debe realizar primero pruebas a corta distancia para que se acoplen correctamente los equipos.

- Si se desea probar el *software* en otro teléfono celular, se debe asegurar que tenga las características de *software* y *hardware* similares a los equipos utilizados en este proyecto.

- Antes de realizar una actualización del sistema operativo del brazo robot, se debe tomar en cuenta las notas de la versión y también de obtener un respaldo de la información.

- Debido a que el sensor de luz del brazo robot *Lego Mindstorms* mide la intensidad de luz reflejada sobre los objetos, se debe considerar que la detección de los colores se la debe realizar en un sitio con poca luz ambiental ya que se obtuvieron mejores resultados en un sitio con esas características.

BIBLIOGRAFÍA

[1] HOLZMANN, Gerard, "*Design and Validation of Computer Protocols*". Primera edición. Prentice-Hall. New Jersey. 1991.

[2] POPOVIC, Miroslav, "*Communication Protocol engineering*". Primera edición. Taylor and Francis Group. Estados Unidos. 2006.

[3] SHARP, Robin, "*Principles of Protocol Design*". Primera edición. Springer. Berlin. 2008.

[4] FOROUZAN, Behrouz, "*Data Communications and Networking*". Tercera edición. McGraw Hill. 2009.

[5] TANENBAUM, Andrew, "*Redes de computadoras*". Cuarta edición. Pearson Educación. Amsterdan. 2003.

[6] LEÓN,Alberto, "*Communication Networks*". Primera edición. McGraw Hill. Estados Unidos. 2001.

[7] BAGAD,V., "*Computer Networks*". Primera edición. Technical Publications Pune. India. 2009.

[8] DUCK,Michael, "*Data Communications and Computer Networks*". Segunda edición. Prentice Hall. Inglaterra. 2003.

[9] BEN ARI, Mordechai, "*Principles of the Spin Model Checker*". Primera edición. Springer. London. 2008.

[10] BEN ARI, Mordechai, "*Principles of Concurrent and Distributed Programming*". Segunda edición. Addison-Wesley. London. 2006.

[11] BROOKE, Phillip, "*Practical Distributed Processing*". Primera edición. Springer. London. 2008.

[12] TANENBAUM, Andrew, "*Distributed Systems*". Segunda edición. Prentice Hall. Amsterdam. 2007.

[13] CARRETERO, Jesús, "*Sistemas Operativos*". Primera edición. McGraw-Hill. Madrid. 2001.

[14] GRAF, Susanne, "*Tools and Algorithms for the Construction and Analysis of*

Systems 6 conf.". Primera edición. Springer. Berlin. 2000.

[15] GOPALAKRISHNAN, Ganesh, "*Computation Engineering*". Primera edición. Springer. EEUU. 2006.

[16] McMILLAN, L., "*The SMV system*". Primera edición. EEUU. 2000.

[17] DAMS, Dennis, "*Theoretical and Practical Aspects of SPIN Model Checking* ". Primera edición. Springer. Berlin. 1999.

[18] HORE, R., "*Communicating Sequential Processes*". Primera edición. Prentice Hall International. 1985.

[19] THOMPSON, Timothy, "*Bluetooth Application Programming with the Java APIs*". Primera edición. Morgan Kaufman. 2008.

[20] LAPLANTE, Phillip, "*Software enginnering*". Primera edición. CRC Press. London. 2007.

[21] LABIOD, Houda, "*WI-FI, BLUETOOTH, ZIGBEE AND WIMAX*". Primera Edición. Springer. London. 2007.

[22] BORKO, Furht, "*Wireless Internet Handbook: technologies, standards, and application".* Primera Edición. CRC Press. London. 2003.

[23] PRASAD, Ramjee, "*From WPANs to Personal Networks: Technologies and Applications*". Primera Edición. Artech House. London. 2006.

[24] PRATHAP, Reddi, "*Bluetooth Technology and Its Applications with Java and J2ME*". Primera Edición. Prentice Hall. India. 2004.

[25] FIGUEROA, Mary, "*Desarrollo de Aplicaciones Bluetooth Utilizando el API Java JSR-82*

[26] RODRÍGUEZ, Oscar, "*Implementación de una red Inalámbrica Bluetooth"*. Tesis. Universidad del Valle. Santiago de Cali. 2003

[27] GIL, José, "*Bluetooth Visión General de una red inalámbrica*". Universidad de Valencia. 2004

[28] ARIAS, Diego, y MUELA Diego, "*Estudio comparativo entre las tecnologías Bluetooth y WI-FI en ambientes de corto alcance a través de la implementación de dos prototipos y de sus simulación".* Tesis. Escuela Politécnica Nacional. Quito. 2007.

[29] Bluetooth Special Interest Group, "*Specification of the Bluetooth System Covered Core Packege versión: 2.0+EDR*". Noviembre 2004

[30] MORÓN, María**,** "*Estudio del Rendimiento de Perfiles Bluetooth en Redes de Área Personal".* Tesis. Universidad de Málaga. Málaga. 2008

[31] VALENCIA, Virgilio, "*Diseño y construcción de un prototipo de adquisición de datos del consumo de energía eléctrica mediante equipos móviles con tecnología Bluetooth".* Tesis. Escuela Politécnica Nacional. Quito. 2007.

[32] ACOSTA, María, "*Estudio del Estándar IEEE 802.15.4 ZigBee para comunicaciones inalámbricas de área personal de bajo consumo de energía y su comparación con el Estándar IEEE 802.15.1 Bluetooth".* Tesis. Escuela Politécnica Nacional. Quito. 2006.

[33] Bluetooth Special Interest Group, "*Bluetooth Specification Serial Port Profile*". Noviembre 2004

[34] Lego Group, *"Lego Mindstorms NXT Bluetooth Developer Kit"****.*** 2006

[35] NAVARRO, Javier, "*Introducción a J2ME".* Departamento de Informática y Automática. Universidad de Salamanca. 2005

[36] VILLAGRÁN, Edison, "*Diseño y construcción de un sistema para control de dispositivos eléctricos dentro de una vivienda empleando tecnología Bluetooth".* Tesis. Escuela Politécnica Nacional. Quito. 2008.

[37] GÁLVEZ, Sergio, "*Java a Tope J2ME (JAVA 2 MICRO EDITION)".* Dpto. de Lenguajes y Ciencias de la Computación E.T.S. de Ingeniería Informática Universidad de Málaga.

[38] BORCHES, Pedro, "*Java 2 Micro Edition Soporte Bluetooth".* Universidad Carlos III de Madrid. 2004.

[39] KNUDSEN, Jonathank, "*Beginning J2ME from Novice to Profesional"*. Tercera Edición. Apress. 2005.

[40] HAMER, Carol, "*Creating Mobile Games Using Java ME Platform to Put the Fun into Your Mobile Device and Cell Phone".* Apress. 2007.

[41] SCHOLZ, Matthias, "*Advanced NXT The Da Vinci Inventions Book".* Apress. 2007.

[42] FERRARI, Mario, "*Building Robots with LEGO MINDSTORMS NXT".* SYNGRESS. 2007.

[43] QUINTAS, Agustin, *"J2ME Java 2 Micro Edition Manual de usuario y tutorial"*. Alfaomega. 2004.

[44] CAPRILLE, Sergio, *"Desarrollo de aplicaciones con comunicación remota basadas en módulos ZigBee y 802.15.4".* Primera Edición. Gran Aldea. 2009

[45] GUERRÓN, Karina, "*Implementación de un prototipo de prueba para la automatización del manejo de la información del estado clínico de pacientes y la medición de signos vitales a través de sensores, para la Clínica "Durán" de la ciudad de Ambato".* Tesis. Escuela Politécnica Nacional. Quito. 2006.

[46] Ing. Pablo Hidalgo, "Apuntes de Redes LAN", Escuela Politécnica Nacional Septiembre 2007.

[47] STALLINGS, William, "Comunicación y redes de computadoras". Séptima edición. Prentice Hall.2004.

[48] HYDE, Paul, *"Java Thread Programmingt".* Sams. 1999

[49] Lego Group, *"Lego Mindstorms NXT Hardware Developer Kit".* 2006

[50] BREÑA, Juan, "*Develop leJOS programs Step by Step*. 2009. *URL*: http://www.juanantonio.info/lejos-ebook/

[51] *JSR* 139 *Connected, Limited Device Configuration* (*CLDC* 1.1) *URL*: http://jcp.org/en/jsr/detail?id=139

[52] *JSR* 118 *Mobile Information Device Profile* (*MIDP* 2.0). *URL*: http://www.jcp.org/en/jsr/detail?id=118

[53] *JSR* 82 *Java APIs for Bluetooth*. *URL*: http://jcp.org/en/jsr/detail?id=82

[54] *Connected, Limited Device Configuration* (*CLDC* 1.1) *API*. *URL*: http://download.oracle.com/javame/config/cldc/ref-impl/cldc1.1/jsr139/ index.html

[55] *Mobile Information Device Profile* (*MIDP* 2.0) *API*. *URL*: http://download.oracle.com/javame/config/cldc/ref-impl/midp2.0 /jsr118/index. html

[56] *Java APIs for Bluetooth*. *URL*: http://download.oracle.com/javame /config/

cldc/opt-pkgs/api/bluetooth/jsr082/ index.html

[57] *leJOS NXJ API. URL* http://lejos.sourceforge.net/nxt/ nxj/api/index.html

[58] *leJOS NXJ Tutorial. URL*: http://lejos.sourceforge.net/nxt/nxj/tutorial/ index.htm

[59] http://authors.phptr.com/bluetooth/bray/pdf/cr_ch10.pdf

[60] http://www.kjhole.com/Standards/BT/BT-PDF/Bluetooth7alt.pdf

[61] http://www.csr.com/products/technology/bluetooth

[62] http://gruaboy.blogsome.com/2009/04/19/lego-mindstorms-nxt/

[63] http://www.philohome.com/nxtmotor/nxtmotor.htm

[64] http://www.electricbricks.com/lego-educativo-mindstorms-sensor-c-25_21_70_145.html

[65] http://lego.brandls.info/nxt_sonic.jpg

[66] http://www.palowireless.com/infotooth/tutorial/hci.asp

[67] http://thenxtstep.blogspot.com/2011/01/we-need-your-bluetooth-experience.html

[68] www.nokia.com/pcsuite

[69] http://www.slideshare.net/Dryope/tecnologia-java-420996

[70] http://www.kamami.pl/dl/BlueCore4_External_Datasheet.pdf

[71] http://www.cepeu.edu.py/LIBROS_ELECTRONICOS_3/lpcu097%20-%2001.pdf

[72] http://www.mintrab.gov.ec/

[73] http://theory.stanford.edu/~rvg/process.html

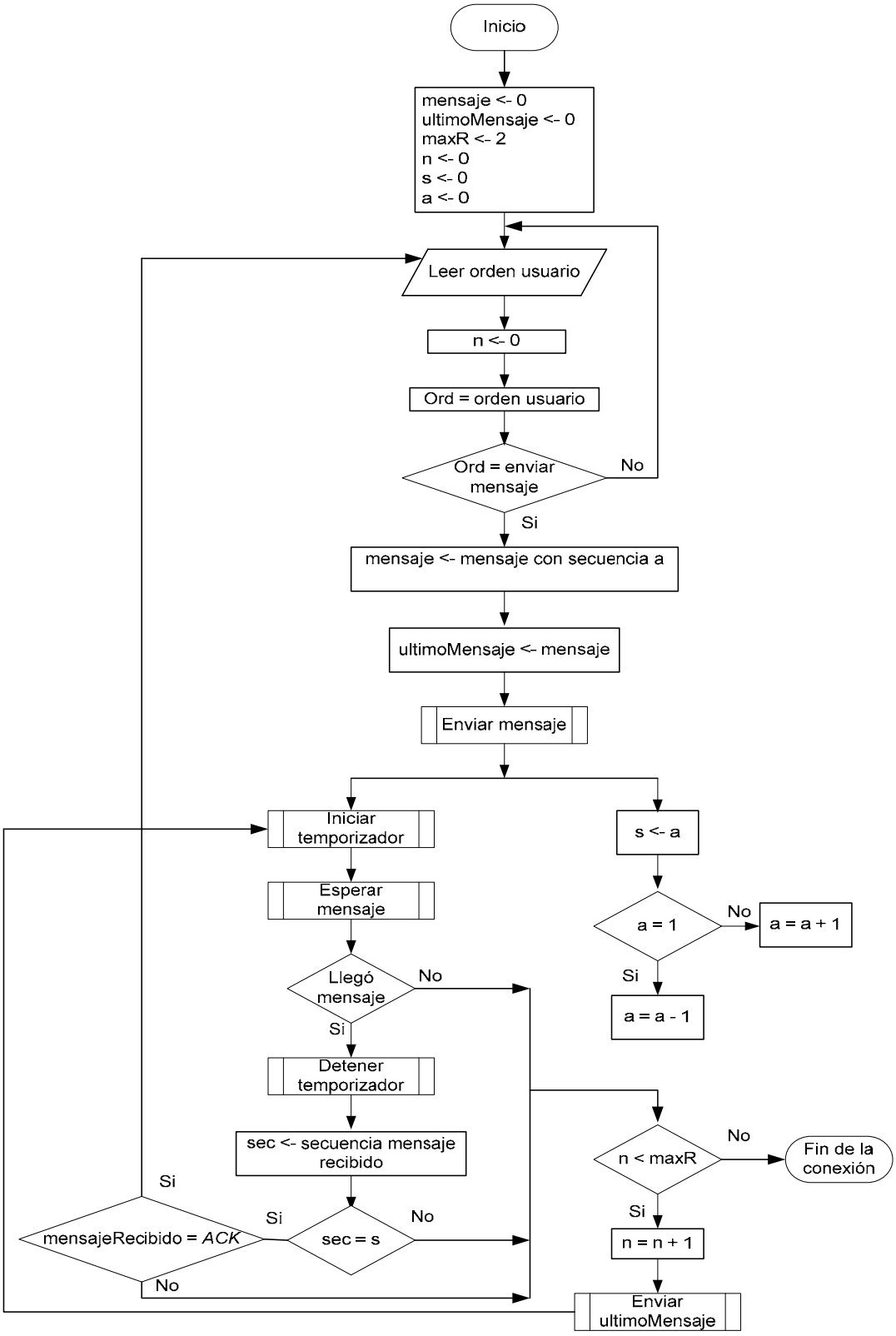
[74] http://www.voronkov.com/

[75] http://www.ece.cmu.edu/~koopman/crc/index.html

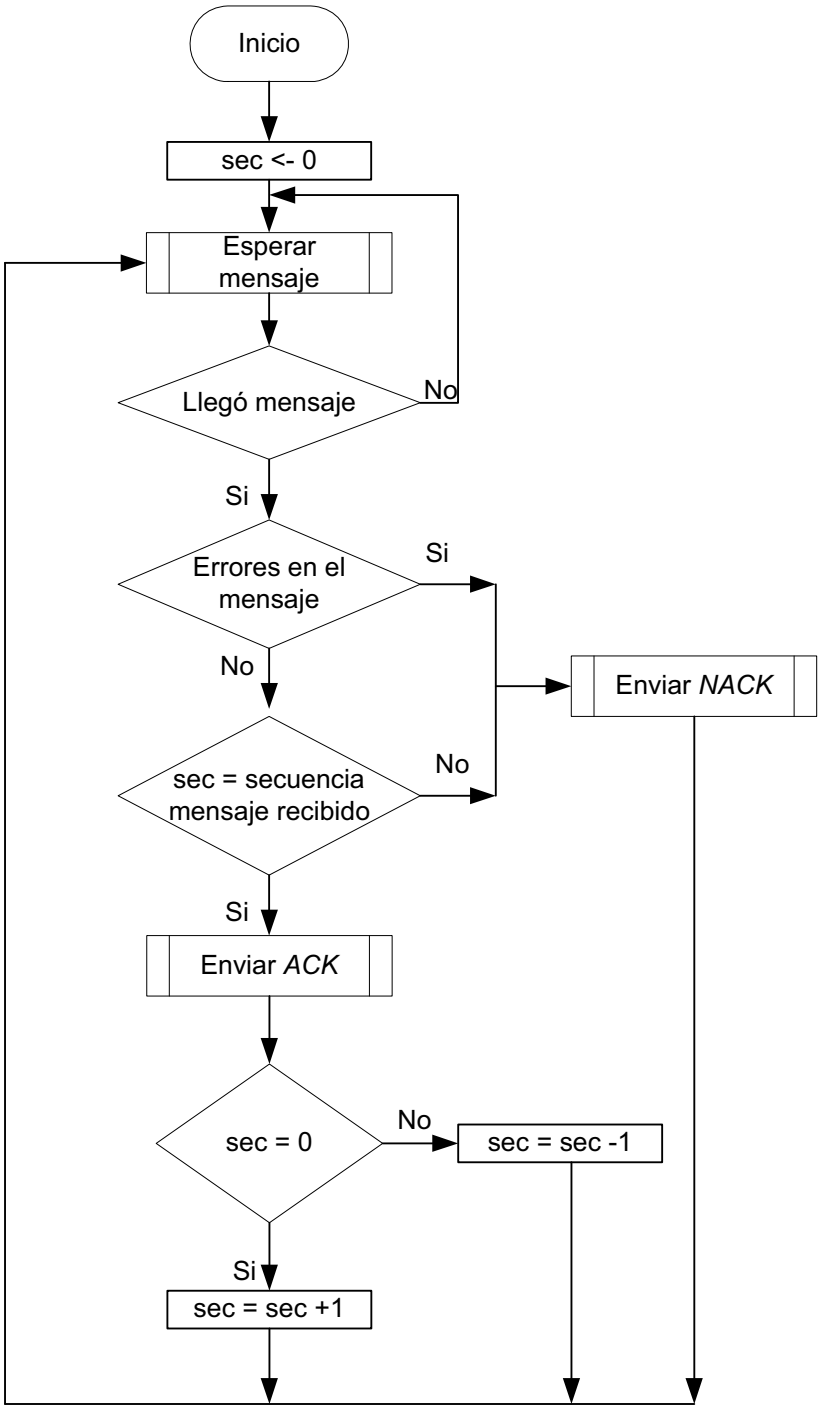
[76] http://www.forum.nokia.com/Devices/

ANEXO A: DIAGRAMAS DE FLUJO DEL MÉTODO *STOP AND WAIT* PARA EL TRANSMISOR Y RECEPTOR

A.1 DIAGRAMA DE FLUJO DEL MÉTODO *STOP AND WAIT* PARA EL TRANSMISOR



A.2 DIAGRAMA DE FLUJO DEL MÉTODO *STOP AND WAIT* PARA EL RECEPTOR



A.3 EXPLICACIÓN DE LAS VARIABLES DE LOS DIAGRAMAS DE FLUJO DEL MÉTODO STOP AND WAIT PARA EL TRANSMISOR Y EL RECEPTOR

Las variables del transmisor y su función se detallan a continuación:

- La variable *ultimoMensaje* se utiliza para almacenar el mensaje que se envió después de recibir una orden del usuario. Si se recibe un *NACK* por parte del brazo robot, el teléfono celular utilizará el valor de esta variable para realizar la retransmisión.
- La variable *a* se utiliza para establecer el número de secuencia que se debe enviar con el próximo mensaje.
- La variable *s* se utiliza para almacenar el número de secuencia que se envió con el último mensaje.
- La variable *sec* se utiliza para almacenar el número de secuencia del acuse de recibo enviado por el brazo robot.
- La variable *mensajeRecibido* permite verificar al transmisor identificar el tipo de acuse recibido.
- La variable *maxR* es una constante que representa el número máximo de retransmisiones que realizará el transmisor en caso de recibir *NACK* y mensajes con error.
- La variable *n* permite conocer al transmisor cuántas retransmisiones ha realizado hasta el momento. Si sobrepasa el valor *maxR* informará al transmisor que debe finalizar la conexión.

El receptor requiere únicamente de una sola variable que almacene el número de secuencia que se espera recibir y en este caso se denomina *sec*.

ANEXO B: PROGRAMA DEL MODELO DEL PROTOCOLO DE COMUNICACIÓN ESCRITO EN *PROMELA*

```
#define MAX 3  /*máximo número de rtx*/

mtype = {pCon,uval,nuval,con,ncon,pTx,pFinCon,conf,nconf,null,msg,ack,nack,error,rojo,azul};

chan pr2sapr = [0] of {mtype};
chan sapr2usr = [0] of {mtype};
chan secr2sapr = [0] of {mtype};
chan pr2sapm = [0] of {mtype};
chan sapm2usr = [0] of {mtype};
chan secm2sapm = [0] of {mtype};
chan secm2secr = [0] of {mtype};
chan secr2secm = [0] of {mtype};
chan pr2rxr = [0] of {mtype};
chan pr2rxm = [0] of {mtype};
chan rxm2fc = [0] of {mtype};
```

B.1. Proceso USR

```
proctype USR(){
mtype confir;
        atomic {
                run PR();
                pr2sapr!pCon;
                sapr2usr?confir;
                assert(confir != null);
                if
                :: confir == conf ->
                        confir = null;
                        printf("st1");
                        run PM();
                        assert(confir == null);
                        pr2sapm!pCon;
                        sapm2usr?confir;
                        assert(confir != null);
                        if
                        :: confir == conf ->
                                confir = null;
                                printf("st2");
                                assert(confir == null);
                        :: confir == nconf ->
                                confir = null;
                                printf("err2");
                                assert(confir == null);
                        fi;
                        sapr2usr?confir;
                        assert(confir != null);
                        if
                        :: confir == con ->
                                confir = null;
                                printf("st3");
                        :: confir == ncon ->
```

```
                                        confir = null;
                                        printf("err3");
                                fi;
                                sapm2usr?confir;
                                assert(confir != null);
                                if
                                :: confir == con ->
                                        confir = null;
                                        printf("st4");
                                        assert(confir == null);
                                        pr2sapm!pTx;
                                :: confir == ncon ->
                                        confir = null;
                                        printf("err4");
                                        assert(confir == null);
                                fi;
                        :: confir == nconf ->
                                confir = null;
                                printf("err1");
                                assert(confir == null);
                        fi
                }
}
```

B.2. Proceso PR

```
proctype PR(){
        atomic{
                run SAPR();
        }
}
```

B.3 Proceso PM

```
proctype PM(){
        atomic{
                run SAPM();
        }
}
```

B.4. Proceso SAPR

```
proctype SAPR(){
        mtype pet;
        mtype confir;
        atomic{
                do
                :: pr2sapr?pet ->
                if
                        :: pet == pCon ->
                                pet = null;
                                run SECR();
                                assert(pet == null);
                                secr2sapr?confir;
                                sapr2usr!confir;
                                secr2sapr?confir;
                                if
                                :: confir == con ->
                                        run RXR();

                                :: confir == ncon ->
```

```
                                        skip;
                                fi;
                                sapr2usr!confir;
                        :: pet == pTx ->
                                pet = null;
                                run TXR();
                                assert(pet == null);
                        :: pet == pFinCon ->
                                pet = null;
                                assert(pet == null);
                                goto terminarSAPR;
                        fi;
                od;
                terminarSAPR:
                        printf("FSPR");
                        skip;
        }
}
```

B.5. Proceso SAPM

```
proctype SAPM(){
        mtype pet;
        mtype confir;
        atomic{
                do
                :: pr2sapm?pet->
                if
                        :: pet == pCon ->
                                pet = null;
                                run SECM();
                                assert(pet == null);
                                secm2sapm?confir;
                                sapm2usr!confir;
                                secm2sapm?confir;
                                if
                                :: confir == con ->
                                        run RXM(MAX);
                                :: confir == ncon ->
                                        skip;
                                fi;
                                sapm2usr!confir;
                        :: pet == pTx ->
                                pet = null;
                                run TXM();
                                assert(pet == null);
                        :: pet == pFinCon ->
                                pet = null;
                                assert(pet == null);
                                goto terminarSAPM;
                        fi;
                od;
                terminarSAPM:
                        printf("FSPM");
                        skip;
        }
}
```

B.6. Proceso SECR

```
proctype SECR(){
        bool a;
        mtype confir;
        mtype usr;
        a = true;
```

```
	atomic{
		if
		:: a == true ->
			secr2sapr!conf;
			confir = conf;
		:: a == true ->
			secr2sapr!nconf;
			confir = nconf;
		fi;
		if
		:: confir == conf ->
			secm2secr?usr;
			if
			:: usr == uval ->
				secr2sapr!con;
				secr2secm!con;
			:: usr == nuval ->
				secr2sapr!ncon;
				secr2secm!ncon;
			fi;
		:: confir == nconf ->
			skip;
		fi
	}
}
```

B.7. Proceso SECM

```
proctype SECM(){
	bool a;
	mtype confir;
	mtype resp;
	a = true;
	atomic{
		if
		:: a == true ->
			secm2sapm!conf;
			confir = conf;
		:: a == true ->
			secm2sapm!nconf;
			confir = nconf;
		fi;
		if
		:: confir == conf ->
			if
			:: a == true ->
				secm2secr!uval;
			:: a == true ->
				secm2secr!nuval;
			fi;
			secr2secm?resp;
			if
			:: resp == con ->
				secm2sapm!con;
			:: resp == ncon ->
				secm2sapm!ncon;
			fi
		:: confir == nconf ->
			skip;
		fi;
	}
}
```

B.8. Proceso RXR

```
proctype RXR(){
	mtype rec;
	atomic{
		do
		:: pr2rxr?rec ->
			if
			:: rec == msg ->
				pr2sapr!pTx;
			:: rec == pFinCon ->
				goto terminarRXR;
			fi;
		od;
		terminarRXR:
			printf("FRXR");
			skip;
	}
}
```

B.9. Proceso RXM

```
proctype RXM(byte maxR){
	mtype rec;
	byte n;
	n = 0;
	atomic{
		do
			:: pr2rxm?rec ->
				if
				:: rec == ack ->
					run FC();
					rxm2fc!pFinCon;
				:: rec == nack ->
					n = n + 1;
					if
					:: n < maxR ->
						pr2sapm!pTx;
					:: n >= maxR ->
						run FC();
						rxm2fc!pFinCon;
					fi;
				:: rec == error ->
					n = n + 1;
					if
					:: n < maxR ->
						pr2sapm!pTx;
					:: n >= maxR ->
						run FC();
						rxm2fc!pFinCon;
					fi;
				:: rec == rojo ->
					printf("rojo");
				:: rec == azul ->
					printf("azul");
				:: rec == pFinCon ->
					goto terminarRXM;
				fi;
		od;
	  terminarRXM:
			printf("FRXM");
			skip;
	}
}
```

B.10. Proceso TXR

```
proctype TXR(){
	bool a;
	a = true;
	atomic{
		if
		:: a == true ->
			pr2rxm!ack;
		:: a == true ->
			pr2rxm!nack;
		:: a == true ->
			pr2rxm!error;
		:: a == true ->
			pr2rxm!rojo;
			if
			:: a == true ->
				pr2rxm!ack;
			:: a == true ->
				pr2rxm!nack;
			:: a == true ->
				pr2rxm!error;
			fi;
		:: a == true ->
			pr2rxm!azul;
			if
			:: a == true ->
				pr2rxm!ack;
			:: a == true ->
				pr2rxm!nack;
			:: a == true ->
				pr2rxm!error;
			fi;
		fi;
	}
}
```

B.11. Proceso TXM

```
proctype TXM(){
	atomic{
		pr2rxr!msg;
	}
}
```

B.12. Proceso FC

```
proctype FC(){
	mtype rec;
	atomic{
			rxm2fc?rec;
			if
			:: rec == pFinCon ->
				pr2rxm!rec;
				pr2rxr!rec;
				pr2sapm!rec;
				pr2sapr!rec;
			fi
	}
}
```

B.13. Proceso init

```
init{
        atomic {
                run USR();
        }
}
```

ANEXO C: PROGRAMA DESARROLLADO PARA LOS TELÉFONOS CELULARES

C.1. Paquete AppRobot.midRobot

C.1.1. Clase MidletRobot

```
package AppRobot.midRobot;

import AppRobot.midRobot.utileria.pantallas.peticionTxListeners.*;
import AppRobot.midRobot.utileria.interfaces.*;
import AppRobot.midRobot.utileria.pantallas.*;
import protocoloRAP.servicios.transmision.*;
import protocoloRAP.servicios.conexion.*;
import AppRobot.midRobot.utileria.*;
import AppRobot.midRobot.utileria.pantallas.peticionTxListeners.PeticionTxListeners;
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import protocoloRAP.canalRFCOMM.*;
import protocoloRAP.SAP.*;
import protocoloRAP.*;

public final class MidletRobot extends MIDlet implements CommandListener, InterfazMensajes {

    private Comando ingresar,info,buscar,fin,mostrarLog,mostrarLogAux,
            atras,borrar,enviarConfig,configurarColor,salir;
    private Presentacion presentacion;
    private ControlRobot controlRobot;
    private EsperaServidor esperaServidor;
    private Protocolo protocolo;
    private Alerta informacion,error,colorRojo,colorAzul,errorTramaColor,
            errorConexion,finalizacion,errorFinalizacion;
    private Busqueda busqueda;
    private Espera espera;
    private Log log,logAux;
    private ConfiguracionColor confColor;
    private Imagen bolar,bolaa;
    private Image[] imaarr;
    private String[] opciorr;

    public MidletRobot () {

        try {
            Display disp = Display.getDisplay(this);
            this.ingresar = new Comando("Ingresar",Command.OK,0);
            this.info = new Comando("Info",Command.HELP,0);
            this.buscar = new Comando ("Buscar",Command.OK,0);
            this.fin = new Comando ("Fin",Command.OK,0);
            this.mostrarLog = new Comando ("Log",Command.OK,0);
            this.mostrarLogAux = new Comando ("Log",Command.OK,0);
            this.atras = new Comando ("Atras",Command.OK,0);
            this.borrar = new Comando ("Borrar",Command.OK,0);
            this.enviarConfig = new Comando ("Enviar",Command.OK,0);
            this.configurarColor = new Comando ("Conf",Command.OK,0);
            this.salir = new Comando ("Salir",Command.OK,0);
            this.controlRobot = new ControlRobot("Control Robot",disp);
            this.espera = new Espera("Buscando dispositivos ...",disp);
            this.esperaServidor = new EsperaServidor("Esperando ...",disp);
```

```
this.log = new Log("Log",disp);
this.logAux = new Log("Log",disp);
this.bolar = new Imagen("/bolar.GIF");
this.bolaa = new Imagen("/bolaa.GIF");
Image[] imagenes = {this.bolar.obtenerImagen(),
          this.bolaa.obtenerImagen()};
this.imaarr = imagenes;
String[] opciones = {"Color rojo","Color azul"};
this.opciorr = opciones;
this.confColor = new ConfiguracionColor("Elija el color",
 List.EXCLUSIVE,this.opciorr,this.imaarr,disp);
String mensajeErrorConexion = "No se establecio conexion";
this.errorConexion = new Alerta("Info Aplicacion", mensajeErrorConexion,
AlertType.ERROR, "/error.gif",disp);
this.errorConexion.ingresarComando(this.fin);
this.errorConexion.setCommandListener(this);
this.busqueda = new Busqueda("Busqueda del
robot",this.controlRobot,this.errorConexion,disp);
String mensajeError = "Se cerrará la aplicación debido a un fallo en la transmisión";
this.error = new Alerta("Error en la transmisión", mensajeError,
AlertType.ERROR, "/error.gif",disp);
String mensajeRobotR = "Se detectó pelota de color rojo";
this.colorRojo = new Alerta("Info aplicación", mensajeRobotR,
AlertType.INFO, "/bolar.GIF",disp);
this.colorRojo.setTimeout(Alert.FOREVER);
String mensajeRobotA = "Se detectó pelota de color azul";
this.colorAzul = new Alerta("Info aplicación", mensajeRobotA,
AlertType.INFO, "/bolaa.GIF",disp);
this.colorAzul.setTimeout(Alert.FOREVER);
String mensajeErrorColor = "El robot detectó un color, " +
       "sin embargo la información llegó errada";
this.errorTramaColor = new Alerta("Info aplicación", mensajeErrorColor,
AlertType.INFO, "/info.gif",disp);
this.errorTramaColor.setTimeout(Alert.FOREVER);
this.error.ingresarComando(this.fin);
this.error.ingresarComando(this.mostrarLogAux);
this.error.setCommandListener(this);
String mensajeFinConexion = "La conexión ha finalizado";
this.finalizacion = new Alerta("Info Aplicacion", mensajeFinConexion,
AlertType.INFO, "/info.gif",disp);
String mensajeErrorFinConexion = "Error en la finalización de la conexión";
this.errorFinalizacion = new Alerta("Info Aplicacion", mensajeErrorFinConexion,
AlertType.INFO, "/info.gif",disp);
String mensaje = "Esta aplicacion controla los movimientos " +
   "del brazo robot T-56";
this.informacion = new Alerta ("Info Aplicacion",mensaje,AlertType.
       INFO,"/info.gif",disp);
this.informacion.setTimeout(Alert.FOREVER);
this.presentacion = new Presentacion("Control Robo Arm T-56",
       this.busqueda, disp);
this.finalizacion.ingresarComando(this.fin);
this.presentacion.ingresarComando(this.ingresar);
this.presentacion.ingresarComando(this.info);
this.presentacion.ingresarComando(this.salir);
this.presentacion.setCommandListener(this);
this.busqueda.ingresarComando(this.buscar);
this.busqueda.setCommandListener(this);
this.controlRobot.ingresarComando(this.fin);
this.controlRobot.ingresarComando(this.mostrarLog);
this.controlRobot.ingresarComando(this.configurarColor);
this.controlRobot.setCommandListener(this);
this.log.ingresarComando(this.atras);
this.log.ingresarComando(this.borrar);
this.log.setCommandListener(this);
this.logAux.ingresarComando(this.fin);
this.logAux.setCommandListener(this);
this.confColor.ingresarComando(this.enviarConfig);
```

```
        this.confColor.ingresarComando(this.atras);
        this.confColor.setCommandListener(this);
        this.protocolo = Protocolo.obtenerInstanciaProtocolo();
        Protocolo.obtenerSAP().establecerControlRobot(this.controlRobot);
        Protocolo.obtenerSAP().establecerEsperaServidor(this.esperaServidor);
        Protocolo.obtenerSAP().establecerLog(this.log);
        Protocolo.obtenerSAP().establecerLogAux(this.logAux);
        Protocolo.obtenerSAP().establecerAlerta(this.error);
        Protocolo.obtenerSAP().establecerColorAzul(this.colorAzul);
        Protocolo.obtenerSAP().establecerColorRojo(this.colorRojo);
        Protocolo.obtenerSAP().establecerAlertaColorError
                                (this.errorTramaColor);
        TxListenerSingleton.obtenerInstanciaTxListeners();
    } catch (Exception ex) {
    }
}

public void startApp() {
      this.presentacion.mostrarFormulario();
}

public void pauseApp() {
}

public void destroyApp(boolean unconditional)throws
        MIDletStateChangeException {
    if(this.presentacion != null){
        this.presentacion = null;
    }

    if(this.espera != null){
        if(this.espera.obtenerEstadoHiloEsperar()){
            this.espera.terminarHilo();
            this.espera = null;
        }else{
            this.espera = null;
        }
    }
    if(this.controlRobot != null){
        this.controlRobot = null;
    }
    if(this.busqueda != null){
        this.busqueda = null;
    }
    if(this.log != null){
        this.log = null;
    }
    if(this.logAux != null){
        this.logAux = null;
    }
    if(this.colorAzul != null){
        this.colorAzul = null;
    }
    if(this.colorRojo != null){
        this.colorRojo = null;
    }
    if(this.errorTramaColor != null){
        this.errorTramaColor = null;
    }
    if(this.atras != null){
        this.atras = null;
    }
    if(this.borrar!= null){
        this.borrar = null;
    }
    if(this.buscar!= null){
        this.buscar = null;
```

```
}
if(this.error!= null){
    this.error = null;
}
if(this.esperaServidor!= null){
    this.esperaServidor = null;
}
if(this.fin!= null){
    this.fin = null;
}
if(this.info!= null){
    this.info = null;
}
if(this.informacion!= null){
    this.informacion = null;
}
if(this.ingresar!= null){
    this.ingresar = null;
}
if(this.mostrarLog!= null){
    this.mostrarLog = null;
}
if(this.mostrarLogAux!= null){
    this.mostrarLogAux = null;
}
if(this.enviarConfig!= null){
    this.enviarConfig = null;
}
if(this.configurarColor!= null){
    this.configurarColor = null;
}
if(this.salir!= null){
    this.salir = null;
}
if(this.errorConexion!= null){
    this.errorConexion = null;
}
if(this.finalizacion!= null){
    this.finalizacion = null;
}
if(this.errorFinalizacion!= null){
    this.errorFinalizacion = null;
}
if(this.confColor!= null){
    this.confColor = null;
}
if(this.bolaa!= null){
    this.bolaa = null;
}
if(this.bolar!= null){
    this.bolar = null;
}
if(this.imaarr!= null){
    this.imaarr = null;
}
if(this.opciorr!= null){
    this.opciorr = null;
}
TxRx txrx = Protocolo.obtenerSAP().obtenerTxRx();
if(txrx != null){
```

```
        if(txrx.isAlive()){
          txrx.terminarHilo();
          txrx = null;
        }else{
          txrx = null;
        }
    }
    CanalRFCOMM canalRFCOMM = Protocolo.obtenerSAP().obtenerCanalRFCOMM();
    if(canalRFCOMM != null){
        canalRFCOMM.desconectar();
        canalRFCOMM = null;
    }
    ConexionBTclt conexionBTclt = Protocolo.obtenerSAP().
            obtenerConexionBTclt();
    if(conexionBTclt != null){
            conexionBTclt = null;
    }
    PeticionTxListeners ptxl = TxListenerSingleton.obtenerTxListeners();
    if(ptxl.isAlive()){
        ptxl.terminarHilo();
        ptxl = null;
    }else{
        ptxl = null;
    }
    TxListenerSingleton ptx = TxListenerSingleton.obtenerInstanciaTxListeners();
    if(ptx != null){
      ptx = null;
    }
    PuntoAccesoServicio sap = Protocolo.obtenerSAP();
    if(sap.isAlive()){
        sap.terminarHilo();
        sap = null;
    }else{
        sap = null;
    }
    if(this.protocolo != null){
        this.protocolo = null;
    }
    System.gc();
}

public void commandAction(Command cmd, Displayable disp) {
    if(disp == this.presentacion){
        if (cmd == this.ingresar){
            this.busqueda.mostrarFormulario();
        }else if (cmd == this.info){
            this.informacion.mostrarAlerta();
        }else if (cmd == this.salir){
            try {
                this.destroyApp(false);
                this.notifyDestroyed();
            } catch (MIDletStateChangeException ex) {
            }
        }else{
        }

    }else if(disp == this.busqueda){
        if (cmd == this.buscar){
```

```
        try {
            this.espera.ejecutarFormularioEspera();
            this.espera.mostrarFormulario();
            Thread.sleep(200);
            this.busqueda.HiloAdministradorConexion(this.espera);
        } catch (Exception ex) {
        }
    }else{
    }
}else if(disp == this.controlRobot){
    if(cmd == this.fin){
        try {
            Protocolo.obtenerSAP().peticionServicio
                (MidletRobot.P_FINALIZACION_CONEXION);
            boolean respuesta = Protocolo.obtenerSAP().
                confirmacionServicio();
            if (respuesta == MidletRobot.ACK_FINALIZACION_CONEXION){
                this.finalizacion.mostrarAlerta();
                this.destroyApp(false);
                this.notifyDestroyed();
            }else{
                this.errorFinalizacion.mostrarAlerta();
                this.destroyApp(false);
                this.notifyDestroyed();
            }
        } catch (Exception ex) {
        }
    }else if(cmd == this.mostrarLog){
        this.log.mostrarFormulario();
    }else if(cmd == this.configurarColor){
        this.confColor.mostrarLista();
    }else{
    }
}else if(disp == this.log){
    if(cmd == this.atras){
        this.controlRobot.mostrarFormulario();
    }else if(cmd == this.borrar){
        this.log.deleteAll();
    }else{
    }
}else if(disp == this.error){
    if(cmd == this.fin){
        try {
            this.destroyApp(false);
            this.notifyDestroyed();
        } catch (MIDletStateChangeException ex) {
        }
    }else if(cmd == this.mostrarLogAux){
        this.logAux.mostrarFormulario();
    }else{
    }
}else if(disp == this.logAux){
    if(cmd == this.fin){
         try {
            this.destroyApp(false);
            this.notifyDestroyed();
        } catch (MIDletStateChangeException ex) {
        }
```

```
            }else{
            }

        }else if(disp == this.errorConexion){
            if(cmd == this.fin){
                try {
                    this.destroyApp(false);
                    this.notifyDestroyed();
                } catch (MIDletStateChangeException ex) {
                }
            }else{
            }

        }else if(disp == this.confColor){
            if(cmd == this.enviarConfig){
                int indiceSeleccionado = this.confColor.getSelectedIndex();
                switch(indiceSeleccionado){
                    case 0:
                        byte[] tramaEnviarR = {MidletRobot.COLOR_ROJO};
                        Protocolo.obtenerSAP().obtenerTxRx().
                                establecerEstadoColorRojo(true);
                        TxListenerSingleton.obtenerTxListeners()
                                .peticionTxListeners(tramaEnviarR,false);
                        break;
                    case 1:
                        byte[] tramaEnviarA = {MidletRobot.COLOR_AZUL};
                        Protocolo.obtenerSAP().obtenerTxRx().
                                establecerEstadoColorRojo(false);
                        TxListenerSingleton.obtenerTxListeners()
                                .peticionTxListeners(tramaEnviarA,false);
                        break;
                    default:
                }

            }else if(cmd == this.atras){
                this.controlRobot.mostrarFormulario();
            }else{
            }
        }else{
        }
    }
}
```

C.2. Paquete AppRobot.midRobot.utileria

C.2.1. Clase Alerta

```
package AppRobot.midRobot.utileria;

import AppRobot.midRobot.utileria.interfaces.*;
import javax.microedition.lcdui.*;
import java.io.*;

public class Alerta extends Alert implements InterfazAlertas {
    private Imagen imagen;
    private Display disp;
    public Alerta(String motivo,String mensaje,AlertType tipoAlerta,
```

```
            String pathImagen,Display disp) throws IOException{
        super(motivo,mensaje,null,tipoAlerta);
        this.disp = disp;
        this.imagen = new Imagen(pathImagen);
        this.setImage(this.imagen.obtenerImagen());
    }
    public void mostrarAlerta() {
        this.disp.setCurrent(this);
    }
    public void ingresarComando(Comando comando) {
        this.addCommand(comando);
    }
}
```

C.2.2. Clase CajaTexto

```
package AppRobot.midRobot.utileria;

import javax.microedition.lcdui.*;

public class CajaTexto extends TextField {
    public CajaTexto(String mensaje,int max,int propiedades){
        super(null,mensaje,max,propiedades);
    }
}
```

C.2.3. Clase Comando

```
package AppRobot.midRobot.utileria;

import javax.microedition.lcdui.*;

public class Comando extends Command {
    public Comando(String nombre,int tipo,int prioridad) {
        super(nombre,tipo,prioridad);
    }
}
```

C.2.4. Clase Formulario

```
package AppRobot.midRobot.utileria;

import AppRobot.midRobot.utileria.interfaces.*;
import javax.microedition.lcdui.*;

public class Formulario extends Form implements InterfazFormularios {
    private Display disp;
    public Formulario(String titulo,Display disp){
        super (titulo);
        this.disp = disp;
    }
    public void ingresarImagen (Image imagen){
        this.append(imagen);
    }
    public void ingresarComando (Command comando){
        this.addCommand(comando);
    }
    public void ingresarTicker (Ticker tick) {
```

```
        this.setTicker(tick);
    }
    public void ingresarItem (Item item) {
        this.append(item);
    }
    public void mostrarFormulario() {
        this.disp.setCurrent(this);
    }
    public Display obtenerDisplay(){
        return this.disp;
    }
    public void anadirString(String texto) {
        this.append(texto);
    }
}
```

C.2.5. Clase Gauges

```
package AppRobot.midRobot.utileria;

import javax.microedition.lcdui.*;

public class Gauges extends Gauge {
    public Gauges(String titulo,boolean interactivo,int min,int max){
        super(titulo,interactivo,max,min);
        this.setLayout(Item.LAYOUT_CENTER);
    }
}
```

C.2.6. Clase Imagen

```
package AppRobot.midRobot.utileria;

import javax.microedition.lcdui.*;
import java.io.*;

public class Imagen {
    private Image imagen;
    public Imagen(String pathnombre) throws IOException {
        this.imagen = Image.createImage(pathnombre);
    }
    public Image obtenerImagen() {
        if (this.imagen != null){
            return this.imagen;
        }else{
            return null;
        }
    }
}
```

C.2.7. Clase Lista

```
package AppRobot.midRobot.utileria;

import AppRobot.midRobot.utileria.interfaces.*;
import javax.microedition.lcdui.*;

public class Lista extends List implements InterfazLista {
```

```
    private Display disp;
    public Lista(String titulo,int tipo,String[] opciones,Image[] imagenes,Display disp){
        super(titulo,tipo,opciones,imagenes);
        this.disp = disp;
    }
    public void mostrarLista() {
        this.disp.setCurrent(this);
    }
    public void ingresarComando(Command comando) {
        this.addCommand(comando);
    }
}
```

C.2.8. Clase StringI

```
package AppRobot.midRobot.utileria;

import javax.microedition.lcdui.*;

public class StringI extends StringItem {
    public StringI (String texto) {
        super(texto,null);
    }
}
```

C.2.9. Clase Tick

```
package AppRobot.midRobot.utileria;

import javax.microedition.lcdui.*;

public class Tick extends Ticker {
    public Tick (String etiqueta) {
        super (etiqueta);
    }
}
```

C.3. Paquete AppRobot.midRobot.utileria.interfaces

C.3.1. Interfaz InterfazAlertas

```
package AppRobot.midRobot.utileria.interfaces;

import AppRobot.midRobot.utileria.*;

public interface InterfazAlertas {
    public abstract void mostrarAlerta();
    public abstract void ingresarComando (Comando comando);
}
```

C.3.2. Interfaz InterfazFormularios

```
package AppRobot.midRobot.utileria.interfaces;
```

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

public interface InterfazFormularios {
   public abstract void ingresarImagen (Image imagen);
   public abstract void ingresarComando (Command comando);
   public abstract void ingresarTicker (Ticker tick);
   public abstract void ingresarItem (Item item);
   public abstract void mostrarFormulario();
   public abstract void anadirString(String texto);
}
```

C.3.3. Interfaz InterfazLista

```
package AppRobot.midRobot.utileria.interfaces;

import javax.microedition.lcdui.*;

public interface InterfazLista {
   public abstract void ingresarComando (Command comando);
   public abstract void mostrarLista();
}
```

C.3.4. Interfaz InterfazMensajes

```
package AppRobot.midRobot.utileria.interfaces;

   public interface InterfazMensajes {
   public static final int NUMERO_REPETICIONES = 2;
   public static final byte P_ESTABLECIMIENTO_CONEXION  = 1;
   public static final byte P_TRANSFERENCIA_DATOS = 2;
   public static final byte P_FINALIZACION_CONEXION  = 3;
   public static final boolean ACK_ESTABLECIMIENTO_CONEXION  = true;
   public static final boolean NACK_ESTABLECIMIENTO_CONEXION  = false;
   public static final boolean ACK_FINALIZACION_CONEXION  = true;
   public static final boolean NACK_FINALIZACION_CONEXION  = false;
   public static final boolean ERROR = false;
   public static final byte SIN_RESPUESTA = 0;
   public static final int VALOR_TEMPORIZADOR = 500;
   public static final byte ACK_TRANS_TRAMA = 4;
   public static final byte NACK_TRANS_TRAMA = 5;
   public static final boolean ACK_TRANS_TRAMA_SAP = true;
   public static final boolean NACK_TRANS_TRAMA_SAP = false;
   public static final boolean SIN_RESPUESTA_SAP = false;
   public static final byte COLOR_ROJO = 10;
   public static final byte COLOR_AZUL = 15;
}
```

C.3.5. Interfaz InterfazMetodosHilos

```
package AppRobot.midRobot.utileria.interfaces;

public interface InterfazMetodosHilos {
   public abstract void terminarHilo();
}
```

C.3.6. Interfaz InterfazParametrosBluetooth

```
package AppRobot.midRobot.utileria.interfaces;
```

```
public interface InterfazParametrosBluetooth {
   public static final String  PROTOCOLO_BLUETOOTH = "btspp://";
   public static final String  CANAL_UNO = ":1";
   public static final String  CANAL_DOS = ":2";
   public static final String  CANAL_TRES = ":3";
   public static final String  CANAL_SIMULACION_NOKIA = ":22";
   public static final String AUTENTICACION = ";authenticate=false";
   public static final String ENCRIPCION = ";encrypt=false";

   public static final String MASTER = ";master=false";
}
```

C.4. Paquete AppRobot.midRobot.utileria.pantallas

C.4.1. Clase Busqueda

```
package AppRobot.midRobot.utileria.pantallas;

import AppRobot.midRobot.utileria.interfaces.*;
import AppRobot.midRobot.utileria.*;
import javax.microedition.lcdui.*;
import protocoloRAP.*;
import java.io.*;

public final class Busqueda extends Formulario implements InterfazMensajes{
   private CajaTexto lblNombreNXT;
   private StringI txtNombreNXT;
   private ControlRobot controlRobot;
   private Alerta errorConexion;
   public Busqueda(String titulo,ControlRobot control,
         Alerta errorConexion, Display disp) {
      super(titulo,disp);
      this.controlRobot = control;
      this.errorConexion = errorConexion;
      String textoNXT = "Ingrese el nombre del robot: ";
      String nombreNXT = "NXT";
      this.txtNombreNXT = new StringI (textoNXT);
      this.lblNombreNXT = new CajaTexto (nombreNXT,20,TextField.LAYOUT_LEFT);
      this.ingresarItem(this.txtNombreNXT);
      this.ingresarItem(this.lblNombreNXT);
   }
   private void administradorConexion(Espera espera) throws
         InterruptedException, IOException{
      String nombreNXT = this.lblNombreNXT.getString();
      boolean respuesta = this.peticionConexionServidor(nombreNXT, 1);
      if(respuesta){
            this.controlRobot.mostrarFormulario();
            espera.terminarHilo();
      }else{
             this.mostrarFormulario();
             espera.terminarHilo();
             this.errorConexion.mostrarAlerta();
      }
   }
   private boolean peticionConexionServidor(String nombreNXT,int canal)
         throws InterruptedException, IOException{
      Protocolo.obtenerSAP().establecerNombreNXT(nombreNXT);
```

```
      Protocolo.obtenerSAP().establecerCanalComunicacion(canal);
      Protocolo.obtenerSAP().peticionServicio(Busqueda.
            P_ESTABLECIMIENTO_CONEXION);
      boolean respuestaSAP = Protocolo.obtenerSAP().confirmacionServicio();
      if(respuestaSAP){
         return true;
       }else {
          return false;
       }
   }
   public void HiloAdministradorConexion(Espera espera){
     final Espera esp = espera;
      Runnable Admin = new Runnable(){
      public void run() {
            try {
               administradorConexion(esp);
            } catch (Exception ex) {
            }
         }
     };

     Thread TAdmin = new Thread(Admin);
     TAdmin.start();
  }
}
```

C.4.2. Clase ConfiguracionColor

```
package AppRobot.midRobot.utileria.pantallas;

import AppRobot.midRobot.utileria.*;
import javax.microedition.lcdui.*;

public class ConfiguracionColor extends Lista {
   public ConfiguracionColor(String titulo,int tipo,String[] opciones,Image[] imagenes,Display disp){
      super(titulo,tipo,opciones,imagenes,disp);
   }
}
```

C.4.3. Clase ControlRobot

```
package AppRobot.midRobot.utileria.pantallas;

import AppRobot.midRobot.utileria.pantallas.peticionTxListeners.*;
import AppRobot.midRobot.utileria.interfaces.*;
import AppRobot.midRobot.utileria.*;
import javax.microedition.lcdui.*;

public final class ControlRobot extends Formulario
                implements ItemStateListener,InterfazMensajes {
   private Gauges id;
   private Gauges aa;
   private Gauges ts;
   public ControlRobot(String nombre, Display disp){
      super(nombre,disp);
      this.id = new Gauges ("<--izquierda  derecha-->",true,1,2);
      this.aa = new Gauges ("<--arriba           abajo-->",true,1,2);
      this.ts = new Gauges ("<--soltar          tomar-->",true,0,1);
```

```
    this.ingresarItem(this.id);
    this.ingresarItem(this.aa);
    this.ingresarItem(this.ts);
    this.setItemStateListener(this);
}
 public synchronized Gauges obtenerGaugeTomar(){
    return this.ts;
 }
 public void itemStateChanged(Item it) {
       if(it == this.id){
         int valorGauge = this.id.getValue();
         switch(valorGauge){
            case 0:
                  byte[] trama0 = {(byte)1,(byte)valorGauge};
                  TxListenerSingleton.obtenerTxListeners()
                       .peticionTxListeners(trama0,true);
               break;
            case 1:
                   byte[] trama1 = {(byte)1,(byte)valorGauge};
                   TxListenerSingleton.obtenerTxListeners()
                      .peticionTxListeners(trama1,true);
               break;
            case 2:
                   byte[] trama2 = {(byte)1,(byte)valorGauge};
                   TxListenerSingleton.obtenerTxListeners()
                      .peticionTxListeners(trama2,true);
                   break;
            default:
         }
     }else if (it == this.aa) {
          int valorGauge = this.aa.getValue();
          switch(valorGauge){
            case 0:
                  byte[] trama0 = {(byte)2,(byte)valorGauge};
                  TxListenerSingleton.obtenerTxListeners()
                       .peticionTxListeners(trama0,true);
              break;
            case 1:
                  byte[] trama1 = {(byte)2,(byte)valorGauge};
                  TxListenerSingleton.obtenerTxListeners()
                       .peticionTxListeners(trama1,true);
               break;
            case 2:
                  byte[] trama2 = {(byte)2,(byte)valorGauge};
                  TxListenerSingleton.obtenerTxListeners()
                       .peticionTxListeners(trama2,true);
               break;
            default:
         }
    }else if(it == this.ts){
        int valorGauge = this.ts.getValue();
        switch(valorGauge){
            case 0:
                  byte[] trama0 = {(byte)3,(byte)valorGauge};
                  TxListenerSingleton.obtenerTxListeners()
                       .peticionTxListeners(trama0,true);
              break;
            case 1:
```

```
                    byte[] trama1 = {(byte)3,(byte)valorGauge};
                    TxListenerSingleton.obtenerTxListeners()
                        .peticionTxListeners(trama1,true);
                break;
            default:
        }
    }else{
    }
  }
}
```

C.4.4. Clase Espera

```
package AppRobot.midRobot.utileria.pantallas;

import AppRobot.midRobot.utileria.interfaces.*;
import AppRobot.midRobot.utileria.*;
import javax.microedition.lcdui.*;
import java.io.*;

public final class Espera extends Formulario implements Runnable,
      InterfazMetodosHilos {
  private String[] pathsimagenes = {"/r1.GIF","/r2.GIF","/r3.GIF","/r4.GIF"};
  private Imagen img;
  private boolean ejecutar;
  private Thread iniciarEspera;
  public Espera (String titulo, Display disp)  {
    super(titulo,disp);
    this.ejecutar = true;
    try {
      this.img = new Imagen(this.pathsimagenes[0]);
      this.ingresarImagen(this.img.obtenerImagen());
      this.get(0).setLayout(Item.LAYOUT_CENTER);
    } catch (IOException ex) {
    }
    this.iniciarEspera = new Thread(this);
  }
  public void run() {
    this.iniciarEspera();
  }
  private void iniciarEspera(){

    while(this.ejecutar != false) {
      synchronized (this) {
        for(int i = 0;i < this.pathsimagenes.length;i++){
          this.deleteAll();
          try {
            this.img = new Imagen(this.pathsimagenes[i]);
            this.ingresarImagen(this.img.obtenerImagen());
            this.get(0).setLayout(Item.LAYOUT_CENTER);
            Thread.sleep(300);
          } catch (Exception ex) {
          }
        }
      }
    }
  }
  public void ejecutarFormularioEspera(){
```

```
        this.iniciarEspera.start();
    }
    public boolean obtenerEstadoHiloEsperar(){
        if(this.iniciarEspera.isAlive()){
            return true;
        }else{
            return false;
        }
    }
    public synchronized void terminarHilo() {
        this.ejecutar = false;
    }
}
```

C.4.5. Clase EsperaServidor

```
package AppRobot.midRobot.utileria.pantallas;

import AppRobot.midRobot.utileria.*;
import javax.microedition.lcdui.*;

public class EsperaServidor extends Formulario {
    private Imagen fondo;
    public EsperaServidor(String titulo,Display disp){
        super(titulo,disp);
        try {
            this.fondo = new Imagen("/espera.GIF");
            this.ingresarImagen(this.fondo.obtenerImagen());
            this.get(0).setLayout(Item.LAYOUT_CENTER);
        } catch (Exception ex) {
        }
    }
}
```

C.4.6. Clase Log

```
package AppRobot.midRobot.utileria.pantallas;

import AppRobot.midRobot.utileria.*;
import javax.microedition.lcdui.*;

public class Log extends Formulario {
    public Log (String titulo,Display disp){
        super(titulo,disp);
    }
}
```

C.4.7. Clase Presentacion

```
package AppRobot.midRobot.utileria.pantallas;

import AppRobot.midRobot.utileria.*;
import javax.microedition.lcdui.*;

public final class Presentacion extends Formulario {
    private Busqueda busqueda;
    private Tick etiqueta;
    private Imagen fondo;
```

```
    public Presentacion(String titulo,Busqueda busqueda,Display disp)
            throws Exception {
        super(titulo,disp);
        this.busqueda = busqueda;
        String etiq ="Proyecto de titulacion: Elsa Roldan M. - " +
                "Marco Almeida P.";
        this.etiqueta = new Tick (etiq);
        try {
            this.fondo = new Imagen("/roboarm.gif");
        } catch (Exception ex) {
        }
        this.ingresarTicker((Ticker)etiqueta);
        this.ingresarImagen(this.fondo.obtenerImagen());
    }
}
```

C.5. Paquete MidletRobot.utileria.pantallas.peticionTxListeners

C.5.1. Clase PeticionTxListeners

```
package AppRobot.midRobot.utileria.pantallas.peticionTxListeners;

import AppRobot.midRobot.utileria.interfaces.*;
import protocoloRAP.*;
import java.io.*;

public final class PeticionTxListeners extends Thread implements InterfazMensajes {
    private int a;

    private int s;
    private boolean ejecutar;
    public PeticionTxListeners(boolean iniciarThread){
        this.a = 0;
        this.s = 0;
        if (iniciarThread)
            this.start();
    }
    public void run(){
        this.iniciarPeticiones();
    }
    private void iniciarPeticiones() {
        while(this.ejecutar != false) {
            synchronized (this) {
                try {
                    this.wait();
                } catch (InterruptedException ex) {
                }
            }
        }
    }
   public synchronized void peticionTxListeners(byte[] datos, boolean comando){
        this.HiloPeticionTransmision(datos,this.a,comando);
        if(this.a == 0){
              this.a++;
        }else{
              this.a--;
        }
   }
```

```
 private boolean peticionTransmision(byte[] datos,boolean comando)
         throws InterruptedException{
     Protocolo.obtenerSAP().establecerTipoTramaEnviar(comando);
     Protocolo.obtenerSAP().establecerDatosEnviar(datos);
      try {           Protocolo.obtenerSAP().peticionServicio(PeticionTxListeners.
                P_TRANSFERENCIA_DATOS);
          boolean respuestaSAP = Protocolo.obtenerSAP().confirmacionServicio();
          if (respuestaSAP) {
            Protocolo.obtenerSAP().obtenerTxRx().
                  establecerSecuenciaEnviada(this.a);
            this.s = this.a;
            return true;
          } else {
            return false;
          }
      } catch (IOException ex) {
          return false;
      }
   }
private void HiloPeticionTransmision(byte[] datos, int numeroSecuencia,
      boolean comando){
  final byte[] trama = datos;

  final int sec = numeroSecuencia;
  final boolean com = comando;
  Runnable runTran = new Runnable() {
  public void run() {
      try {
            if(trama != null){
               Protocolo.obtenerSAP().establecerNumeroSecuenciaTx(sec);
                  boolean respuesta = peticionTransmision(trama,com);
                  if(respuesta){
                  }else{
                  }
               }else{
               }
           } catch (Exception ex) {
           }
       }
  };
      Thread transferencia = new Thread(runTran);
      transferencia.start();
}
public synchronized void terminarHilo() {
      this.ejecutar = false;
      this.notifyAll();
   }
}
```

C.5.2. Clase TxListenerSingleton

```
package AppRobot.midRobot.utileria.pantallas.peticionTxListeners;

public final class TxListenerSingleton {
   private static final TxListenerSingleton txListenerSingleton
         = new TxListenerSingleton();
   private static PeticionTxListeners pTxL;
   private TxListenerSingleton(){
```

```
        TxListenerSingleton.pTxL = new PeticionTxListeners(true);
    }

    public static TxListenerSingleton obtenerInstanciaTxListeners() {
        return txListenerSingleton;
    }
    public static PeticionTxListeners obtenerTxListeners() {
        return TxListenerSingleton.pTxL;
    }
}
```

C.6. Paquete protocoloRAP

C.6.1. Clase Protocolo

```
package protocoloRAP;

import protocoloRAP.SAP.*;

public final class Protocolo {
    private static final Protocolo protocoSingleton = new Protocolo();
    private static PuntoAccesoServicio sap;
    private Protocolo(){
        Protocolo.sap = new PuntoAccesoServicio(true);
    }
    public static Protocolo obtenerInstanciaProtocolo() {
        return protocoSingleton;
    }
    public static PuntoAccesoServicio obtenerSAP() {
        return Protocolo.sap;
    }
}
```

C.7. Paquete protocoloRAP.SAP

C.7.1. Clase PuntoAccesoServicio

```
package protocoloRAP.SAP;

import protocoloRAP.servicios.OperacionesTramas.*;
import AppRobot.midRobot.utileria.interfaces.*;
import AppRobot.midRobot.utileria.pantallas.*;
import protocoloRAP.servicios.transmision.*;
import protocoloRAP.servicios.conexion.*;
import AppRobot.midRobot.utileria.*;
import protocoloRAP.canalRFCOMM.*;
import java.util.*;
import java.io.*;

public final class PuntoAccesoServicio extends Thread implements
        InterfazMetodosHilos,InterfazMensajes {
  private CanalRFCOMM canalRFCOMM;
  private int canalComunicacion;
  private byte[] datosInterfaz;
  private ConexionBTclt con;
  private String nombreNXT;
  private boolean ejecutar;
  private byte[] servicio;
```

```
private TxRx txrx;
private int a;
private OperacionesTramaclt oT;
private Vector tramas;
private ControlRobot controlRobot;
private EsperaServidor esperaServidor;
private Log log,logAux;
private Alerta error,colorAzul,colorRojo,errorTramaColor;
private boolean comando;
public PuntoAccesoServicio (boolean iniciarThread) {
    this.canalComunicacion = 0;
    this.nombreNXT = null;
    this.servicio = null;
    this.canalRFCOMM = null;
    this.txrx = null;
    this.con = null;
    this.datosInterfaz = null;
    this.ejecutar = true;
    this.comando = true;
    this.tramas = new Vector();
    this.oT = null;
    this.con = new ConexionBTclt(false);
    this.controlRobot = null;
    this.esperaServidor = null;
    this.log = null;
    this.logAux = null;
    this.error = null;
    this.colorAzul = null;
    this.colorRojo = null;
    this.errorTramaColor = null;
    if (iniciarThread)
       this.start();
 }
 public void run(){
    this.iniciarSAP();
 }
 private void iniciarSAP() {
    while(this.ejecutar != false) {
        synchronized (this) {
          try {
              this.wait();
          } catch (InterruptedException ex) {
          }
        }
    }
 }
 public synchronized void peticionServicio(byte mensaje)
                                  throws InterruptedException{
    while( this.servicio != null){
       this.wait();
    }
       byte[] mensajeC = {mensaje};
       this.servicio = mensajeC;
       this.notifyAll();
 }
public synchronized boolean confirmacionServicio() throws
      InterruptedException, IOException{
   while( this.servicio == null){
```

```
            this.wait();
        }
        boolean respuesta = this.determinacionServicio(this.servicio);
        this.servicio = null;
        return respuesta;
    }
     private boolean determinacionServicio(byte[] peticion) throws InterruptedException, IOException{
        switch(peticion[0]){
            case PuntoAccesoServicio.P_ESTABLECIMIENTO_CONEXION:
                 this.con.establecerNombreNXT(this.obtenerNombreNXT());
                 this.con.establecerCanalComunicacion(this.
                         obtenerCanalComunicacion());
                  this.con.start();
                  boolean estadoCon = this.con.conexionEstablecida();
                  if(estadoCon){
                     this.canalRFCOMM = this.con.obtenerCanalRFCOMM();
                     this.oT = new OperacionesTramaclt(this.log,this.logAux);
                     this.txrx = new TxRx (this.canalRFCOMM,
                           this.controlRobot,this.esperaServidor,
                           this.colorRojo,this.colorAzul,
                           this.errorTramaColor,this.log,this.logAux,true);
                     return PuntoAccesoServicio.ACK_ESTABLECIMIENTO_CONEXION;
                  }else{
                     return PuntoAccesoServicio.NACK_ESTABLECIMIENTO_CONEXION;
                  }
            case PuntoAccesoServicio.P_TRANSFERENCIA_DATOS:
                  byte[] datos = this.datosInterfaz;
                  int secuencia = this.a;
                  byte[] trama = null;
                  if(this.comando){
                    trama = this.oT.tramaComandosMovimientos
                                          (datos,secuencia);
                  }else{
                    trama = this.oT.tramaColor(datos,secuencia);
                  }
                  this.tramas.addElement(trama);
                  Vector tramasC = this.tramas;
                  for(int i = 0;i < this.tramas.size(); i++){
                     byte[] te = (byte[]) this.tramas.elementAt(i);
                     byte acuse_td = this.txrx.controlFlujo(te,
                     PuntoAccesoServicio.NUMERO_REPETICIONES);
                     if(acuse_td == PuntoAccesoServicio.ACK_TRANS_TRAMA){
                        tramasC.removeElementAt(i);
                        System.out.println("ACK_TRANS_TRAMA");
                        return PuntoAccesoServicio.ACK_TRANS_TRAMA_SAP;
                     }else if(acuse_td == PuntoAccesoServicio.NACK_TRANS_TRAMA){
                        this.error.mostrarAlerta();
                        return PuntoAccesoServicio.NACK_TRANS_TRAMA_SAP;
                     }else if(acuse_td == PuntoAccesoServicio.SIN_RESPUESTA){
                        this.error.mostrarAlerta();
                        return PuntoAccesoServicio.SIN_RESPUESTA_SAP;
                     }else{
                        this.error.mostrarAlerta();
                        return PuntoAccesoServicio.ERROR;
                     }
                  }
                  this.tramas = tramasC;
            case PuntoAccesoServicio.P_FINALIZACION_CONEXION:
```

```
            byte[] tramaf = this.oT.tramaFinalizacion();
            byte acuse_f = this.txrx.controlFlujo(tramaf,
                  PuntoAccesoServicio.NUMERO_REPETICIONES);
            if(acuse_f == PuntoAccesoServicio.ACK_TRANS_TRAMA){
               return PuntoAccesoServicio.ACK_FINALIZACION_CONEXION;
            }else if(acuse_f == PuntoAccesoServicio.NACK_TRANS_TRAMA){
               return PuntoAccesoServicio.NACK_FINALIZACION_CONEXION;
            }else if(acuse_f == PuntoAccesoServicio.SIN_RESPUESTA){
               return PuntoAccesoServicio.NACK_FINALIZACION_CONEXION;
            }else{
               return PuntoAccesoServicio.ERROR;
            }
         default:
            return PuntoAccesoServicio.ERROR;
      }
   }   public void establecerNombreNXT(String nombreNXT) {
      this.nombreNXT = nombreNXT;
   }
   public void establecerCanalComunicacion(int canalComunicacion) {
      this.canalComunicacion = canalComunicacion;
   }
   private String obtenerNombreNXT(){
      return this.nombreNXT;
   }
   private int obtenerCanalComunicacion(){
      return this.canalComunicacion;
   }
   public synchronized void establecerNumeroSecuenciaTx(int secuencia){
      this.a = secuencia;
   }
   public TxRx obtenerTxRx (){
      return this.txrx;
   }
   public OperacionesTramaclt obtenerOperacionesTrama(){
      return this.oT;
   }
   public CanalRFCOMM obtenerCanalRFCOMM(){
      return this.canalRFCOMM;
   }
   public ConexionBTclt obtenerConexionBTclt(){
      return this.con;
   }
   public synchronized void establecerDatosEnviar(byte[] tramaEnviar){
      this.datosInterfaz = tramaEnviar;
   }
   private synchronized void finalizarConexion(){
      if(this.txrx != null){
         if(this.txrx.isAlive()){
           this.txrx.terminarHilo();
           this.txrx = null;
         }else{
           this.txrx = null;

         }
      }
      if(this.canalRFCOMM != null){
         this.canalRFCOMM.desconectar();
         this.canalRFCOMM = null;
```

```
        }
        if(this.con != null){
            this.con = null;
        }
        this.terminarHilo();
        System.gc();
    }
    public synchronized Vector obtenerVectorTramas(){
        return this.tramas;
    }
    public void establecerControlRobot(ControlRobot controlRobot){
        this.controlRobot = controlRobot;
    }
    public void establecerEsperaServidor(EsperaServidor esperaServidor){
        this.esperaServidor = esperaServidor;
    }
    public void establecerLog (Log log){
        this.log = log;
    }
    public void establecerLogAux (Log logAux){
        this.logAux = logAux;
    }
    public void establecerAlerta(Alerta error){
        this.error = error;
    }
    public void establecerColorAzul(Alerta colorAzul){
        this.colorAzul = colorAzul;
    }
    public void establecerColorRojo(Alerta colorRojo){
        this.colorRojo = colorRojo;
    }
    public void establecerAlertaColorError(Alerta errorTramaColor){
        this.errorTramaColor = errorTramaColor;
    }
    public void establecerTipoTramaEnviar(boolean comando){
        this.comando = comando;
    }
    public synchronized void terminarHilo() {
        this.ejecutar = false;
        this.notifyAll();
    }
}
```

C.8. Paquete protocoloRAP.canalRFCOMM

C.8.1. Clase CanalRFCOMM

```
package protocoloRAP.canalRFCOMM;

import javax.microedition.io.*;
import java.io.*;

public final class CanalRFCOMM {
    private StreamConnection strcon;
    private DataInputStream  streamEntrada;
    private DataOutputStream streamSalida;
    public CanalRFCOMM (StreamConnection strcon) {
        try {
```

```
            this.strcon = strcon;
            this.streamEntrada = this.strcon.openDataInputStream();
            this.streamSalida = this.strcon.openDataOutputStream();
        } catch (IOException ex) {
        }
    }
    public byte[] recibirTrama()throws IOException {
        if(this.streamEntrada.available() == 0){
            return null;
        } else {
            byte tramaRecibida[] = new byte[this.streamEntrada.available()];
            this.streamEntrada.read(tramaRecibida);
            return tramaRecibida;
        }
    }
    public void transmitirTrama (byte[] trama) {
        final DataOutputStream dout = this.streamSalida;
        final byte[] datos = trama;
        Runnable env = new Runnable() {
            public void run() {
                try {
                    dout.write(datos);
                    dout.flush();
                } catch (Exception ex) {
                }
            }
        };
        Thread tenv = new Thread(env);
        tenv.start();
    }
    public DataInputStream obtenerStreamEntrada(){
        return this.streamEntrada;
    }
    public void desconectar() {
        try{
            if(this.streamEntrada != null){
                this.streamEntrada.close();
            }
        }catch(Exception e) {
        }
        try {
            if(this.streamSalida != null)
                this.streamSalida.close();
        }catch(Exception e) {
        }
        try {
            if( this.strcon != null)
                this.strcon.close();
        }catch(Exception e) { }
            this.streamEntrada = null;
            this.streamSalida = null;
            this.strcon = null;
    }
}
```

C.9. Paquete protocoloRAP.servicios.OperacionesTramas

C.9.1. Clase OperacionesTramaclt

```
package protocoloRAP.servicios.OperacionesTramas;
import AppRobot.midRobot.utileria.pantallas.Log;
public class OperacionesTramaclt {
   private Log log,logAux;
   public  OperacionesTramaclt(Log log,Log logAux) {
      this.log = log;
      this.logAux = logAux;
   }
     private int obtenerCRCInt(byte [] arreglo){
        int crc = 0;
        int i = 0;
        int datos = 0;
        int datosInvertidos= 0;
        int poly = 25;
        int opXor = 0;
        datos = this.arregloBytesInt(arreglo);
        datosInvertidos = this.obtenerNumero(datos,16);
        if (datosInvertidos%2==0){
           opXor = (int)(datosInvertidos^0);
        }else {
           opXor = (int)(datosInvertidos^poly);
        }
        for (i = 1;i<12;i++){
           crc= (int)(opXor>>>1);
           if (crc % 2 == 0){
              opXor = (int)(crc^0);
           }else {
              opXor = (int)(crc^poly);
              crc = opXor;
         }
        }
        crc= (int)(opXor>>>1);
        crc = this.obtenerNumero(crc,4);
          return crc;
   }
public  int arregloBytesInt(byte[] arreglo){
       int valor = 0;
       for (int k = 0; k < arreglo.length; k++){
          valor = (valor << 8) + (arreglo[k] & 0xff);
       }
       return valor;
   }
   private int obtenerNumero(int valor, int bits){
        int[] tabla = {1,2,4,8,16,32,64,128,256,512,
                  1024,2048,4096,8192,16384,32768};
        int numero = valor;
        int nBits = bits;
        int numeroInvertido =0;
        for (int j = nBits-1; j>=0;j--){
           if (numero % 2 != 0){
              numeroInvertido = numeroInvertido + tabla[j];
           }
           numero = numero>>>1;
        }
```

```
      return numeroInvertido;
 }
private  byte[] intArregloBytes(int valor){
  byte[] resultado;
  byte b1 = (byte)(( valor >> 8 ) & 0xff) ;
  byte b0 = (byte)( valor  & 0xff) ;
  resultado = new byte[]{ b1 , b0 };
  return resultado;
}
public byte[] tramaComandosMovimientos (byte[] datosC, int sec){
    int secuencia = sec;
    byte[] datosComando = datosC;
    int crc = 0;
    int TramaEnteros = 0;
    byte[] trama = {(byte)0,(byte)0};
    if (datosComando[0]==1 && datosComando[1] ==0){
        trama[0] = (byte)0;
        trama[1] = (byte)208;
    }
    if (datosComando[0]==1 && datosComando[1] ==1){
        trama[0] = (byte)0;
        trama[1] = (byte)176;
    }
    if (datosComando[0]==1 && datosComando[1] ==2){
        trama[0] = (byte)0;
        trama[1] = (byte)192;
    }
    if (datosComando[0]==2 && datosComando[1] ==0){
        trama[0] = (byte)1;
        trama[1] = (byte)16;
    }
    if (datosComando[0]==2 && datosComando[1] ==1){
        trama[0] = (byte)1;
        trama[1] = (byte)48;
    }
    if (datosComando[0]==2 && datosComando[1] ==2){
        trama[0] = (byte)1;
        trama[1] = (byte)32;
    }
    if (datosComando[0]==3 && datosComando[1] ==0){
        trama[0] = (byte)1;
        trama[1] = (byte)224;
    }
    if (datosComando[0]==3 && datosComando[1] ==1){
        trama[0] = (byte)1;
        trama[1] = (byte)240;
    }
    if(secuencia ==1){
        trama[0] = (byte) (trama[0] | (byte) 32);
    }
    TramaEnteros = this.arregloBytesInt(trama);
    crc = this.obtenerCRCInt(trama);
    TramaEnteros = TramaEnteros + crc;
    System.out.println("Trama enviada con crc (entero):  "+TramaEnteros);
     System.out.println("Trama enviada con crc (binario): "+Integer.toBinaryString(TramaEnteros));
    System.out.println("");
    this.log.anadirString("Tx: ");
    this.log.anadirString(Integer.toBinaryString(TramaEnteros)+",");
```

```
        this.log.anadirString(Integer.toString(TramaEnteros)+"\n");
        this.logAux.anadirString("Tx: ");
        this.logAux.anadirString(Integer.toBinaryString(TramaEnteros)+",");
        this.logAux.anadirString(Integer.toString(TramaEnteros)+"\n");
        trama = this.intArregloBytes(TramaEnteros);
        return trama;
    }
       public byte[] tramaColor (byte[] valorColor,int sec){
        byte color = valorColor[0];
        int crc = 0;
        int TramaEnteros = 0;
        byte[] trama = {(byte)0,(byte)0};
        switch (color){
            case 10:
                trama[1] = (byte)160;
            break;
            case 15:
                trama[1] = (byte)240;
            break;
            default:
         }
             if(sec ==1){
                 trama[0] = (byte) (trama[0] | (byte) 32);
             }
             TramaEnteros = this.arregloBytesInt(trama);
             crc = this.obtenerCRCInt(trama);
             TramaEnteros = TramaEnteros + crc;
             System.out.println("Tx color (int):" + "\n");
             System.out.println(TramaEnteros + "\n");
             System.out.println("Tx color (bin):" + "\n");
             System.out.println(Integer.toBinaryString(TramaEnteros) + "\n");
             this.log.anadirString("Tx: ");
             this.log.anadirString(Integer.toBinaryString(TramaEnteros)+",");
             this.log.anadirString(Integer.toString(TramaEnteros)+"\n");
             this.logAux.anadirString("Tx: ");
             this.logAux.anadirString(Integer.toBinaryString(TramaEnteros)+",");
             this.logAux.anadirString(Integer.toString(TramaEnteros)+"\n");
             trama = this.intArregloBytes(TramaEnteros);
      return trama;
    }
     public byte[] tramaFinalizacion (){
        int crc = 0;
        int TramaEnteros = 0;
        byte[] trama = {(byte)16,(byte)0};
        TramaEnteros = this.arregloBytesInt(trama);
        crc = this.obtenerCRCInt(trama);
        TramaEnteros = TramaEnteros + crc;
        trama = this.intArregloBytes(TramaEnteros);
        return trama;
    }
   public byte [] obtenerParamTramaMinLong (byte[]trama){
        int TramaRecividaEntero = this.arregloBytesInt(trama);
        int tipoTrama = 0;
        int transmisor = 0;
        int sec= 0;
        int claseControl = 0;
        int ack = 0;
        byte [] parametrosTrama1 = {(byte)0, (byte)0,(byte)0};
```

```
        tipoTrama = (TramaRecividaEntero >>> 7)&1 ;
        transmisor = (TramaRecividaEntero >>> 6)&1 ;
        if (tipoTrama == 1 && transmisor ==1){
            sec = (TramaRecividaEntero >>> 5)&1;
            claseControl = (TramaRecividaEntero >>> 3)&3;
            ack = (TramaRecividaEntero >>> 2)&1;
        }
        parametrosTrama1[0] = (byte)sec;
        parametrosTrama1[1] = (byte)claseControl;
        parametrosTrama1[2] = (byte)ack;
        int acuseRecivido = this.arregloBytesInt(trama);
        if (ack ==0){
            System.out.println("ACK recibido (entero) : "+acuseRecivido);
            System.out.println("ACK recibido (binario): "+ Integer.toBinaryString(acuseRecivido));
            System.out.println("");
            this.log.anadirString("Rx: ACK ");
            this.log.anadirString(Integer.toBinaryString(acuseRecivido)+",");
            this.log.anadirString(Integer.toString(acuseRecivido)+"\n");
            this.logAux.anadirString("Rx: ACK ");
            this.logAux.anadirString(Integer.toBinaryString(acuseRecivido)+",");
            this.logAux.anadirString(Integer.toString(acuseRecivido)+"\n");

        } else {
            System.out.println("NACK recibido (entero): "+acuseRecivido);
            System.out.println("NACK recibido (binario): "+ Integer.toBinaryString(acuseRecivido));
            System.out.println("");
            this.log.anadirString("Rx: NACK ");
            this.log.anadirString(Integer.toBinaryString(acuseRecivido)+",");
            this.log.anadirString(Integer.toString(acuseRecivido)+"\n");
            this.logAux.anadirString("Rx: NACK ");
            this.logAux.anadirString(Integer.toBinaryString(acuseRecivido)+",");
            this.logAux.anadirString(Integer.toString(acuseRecivido)+"\n");
        }
        return parametrosTrama1;
    }
    public byte[] obtenerParamTramaMaxLong (byte[]trama){
        int TramaRecividaEntero = this.arregloBytesInt(trama);
        int tipoTrama = 0;
        int transmisor = 0;
        int sec= 0;
        int color = 0;
        int fin = 0;
        int ack = 0;
        int tipoTramaDatos = 0;
        byte[] parametrosTrama2 = {(byte)0,(byte)0,(byte)0,(byte)0, (byte)0};
        tipoTrama = (TramaRecividaEntero >>> 15)&1 ;
        int verificarCRC = this.obtenerCRCInt(trama);
        if (verificarCRC == 0){
            tipoTrama = (TramaRecividaEntero >>> 15)&1 ;
            if(tipoTrama==0){
                transmisor = (TramaRecividaEntero >>> 14)&1;
                if (transmisor==1){
                sec = (TramaRecividaEntero >>> 13)&1;
                tipoTramaDatos = (TramaRecividaEntero >>> 12)&1;
                switch (tipoTramaDatos){
                    case 0:
                        color = (TramaRecividaEntero >>> 4)&15;
                    break;
```

```
                case 1:
                    fin = 1;
                    break;
                default:
             }
            }
          }
       }else {
          sec = (TramaRecividaEntero >>> 13)&1;
          ack = 1;
       }
       parametrosTrama2[0] = (byte)fin;
       parametrosTrama2[1] = (byte)color;
       parametrosTrama2[2] =  (byte)sec;
       parametrosTrama2[3] =  (byte)ack;
       return parametrosTrama2;
   }
}
```

C.10. Paquete protocoloRAP.servicios.conexion

C.10.1. Clase ConexionBTclt

```
package protocoloRAP.servicios.conexion;

import AppRobot.midRobot.utileria.interfaces.*;
import protocoloRAP.canalRFCOMM.*;
import javax.microedition.io.*;
import javax.bluetooth.*;
import java.util.*;
import java.io.*;

public final class ConexionBTclt extends Thread implements
       DiscoveryListener,InterfazParametrosBluetooth {
    private int CODIGO_INQUIRY_COMPLETED;
    private boolean busquedaTerminada;
    private StreamConnection strcon;
    private CanalRFCOMM canalRFCOMM;
    private boolean nxtEncontrado;
    private Vector equiposRemotos;
    private boolean conexionNXT;
    private String nombreNXT;
    private String URL;
    private int canal;
  public ConexionBTclt (boolean iniciarCliente){
      this.canal = 1;
      this.nombreNXT = "NXT";
      this.equiposRemotos = new Vector();
      if(iniciarCliente)
          this.start();
  }
  public ConexionBTclt (String nombreNXT, int canal,
                        boolean iniciarThread){
      this.canal = canal;
      this.nombreNXT = nombreNXT;
      this.equiposRemotos = new Vector();
      if(iniciarThread)
          this.start();
```

```
}
 public void run() {
    try {
         this.iniciarCliente();
     } catch (Exception ex) {
     }
}
 private void iniciarCliente() throws BluetoothStateException, InterruptedException, IOException {

     this.busquedaTerminada = false;
     this.nxtEncontrado = false;
     this.conexionNXT = false;
     LocalDevice equipo = LocalDevice.getLocalDevice();
     DiscoveryAgent discoveryAgent = equipo.getDiscoveryAgent();
     equipo.setDiscoverable(DiscoveryAgent.GIAC);
     discoveryAgent.startInquiry(DiscoveryAgent.GIAC, this);
     synchronized(this) {
        this.wait();
     }
     if(CODIGO_INQUIRY_COMPLETED == INQUIRY_COMPLETED) {
     for(int i = 0; i < equiposRemotos.size(); i++) {
           RemoteDevice rd = (RemoteDevice)equiposRemotos.elementAt(i);
              if (rd.getFriendlyName(false).equals(
                           this.obtenerNombreNXT())) {
                 this.URL = this.obtenerURL(rd, this.canal);
                 this.nxtEncontrado = true;
                 break;
              }
     }
              if(this.nxtEncontrado){
                 this.strcon = (StreamConnection)Connector.open(URL);
                 this.canalRFCOMM = new CanalRFCOMM
                          ((StreamConnection)this.strcon);
                 this.conexionNXT = true;
              }
     synchronized(this) {
                this.busquedaTerminada = true;
                this.notifyAll();
           }
     }
}
 public boolean conexionEstablecida() throws InterruptedException {
    int i = 0;
     while(true){
           synchronized(this) {
              this.wait();
           }
      i = i + 1;
         if(i==2){
            break;
         }
     }
     if((this.nxtEncontrado && this.conexionNXT
          && this.busquedaTerminada)== true){
             return true;
     } else {
             return false;
     }
```

```
}
  private String obtenerURL(RemoteDevice rd, int canal) {
     StringBuffer url = new StringBuffer(PROTOCOLO_BLUETOOTH);

     url.append(rd.getBluetoothAddress());
     if(canal == 1){
        url.append(CANAL_UNO);
     }else if(canal == 2){
        url.append(CANAL_DOS);
     }else if(canal == 3){
        url.append(CANAL_TRES);
     }else if(canal == 22){
        url.append(CANAL_SIMULACION_NOKIA);
     }else{
     }
     url.append(AUTENTICACION);
     url.append(ENCRIPCION);
     url.append(MASTER);
     return url.toString();
  }
  public String obtenerNombreNXT(){
     return this.nombreNXT;
  }
  public synchronized void establecerNombreNXT(String nombreNXT) {
     this.nombreNXT = nombreNXT;
  }
  public synchronized void establecerCanalComunicacion(int canal) {
     this.canal = canal;
  }
    public CanalRFCOMM obtenerCanalRFCOMM(){
       return this.canalRFCOMM;
    }
    public void deviceDiscovered(RemoteDevice equipo, DeviceClass dc) {
       this.equiposRemotos.addElement(equipo);
    }
    public void servicesDiscovered(int arg0, ServiceRecord[] arg1) {
    }
    public void serviceSearchCompleted(int arg0, int arg1) {
    }
    public void inquiryCompleted(int dt) {
       CODIGO_INQUIRY_COMPLETED  = dt;
       synchronized(this) {
          this.notifyAll();
       }
    }
}
```

C.11. Paquete protocoloRAP.servicios.transmision

C.11.1. Clase TxRx

```
package protocoloRAP.servicios.transmision;

import protocoloRAP.servicios.OperacionesTramas.*;
import AppRobot.midRobot.utileria.interfaces.*;
import AppRobot.midRobot.utileria.pantallas.*;
import AppRobot.midRobot.utileria.*;
import protocoloRAP.canalRFCOMM.*;
```

```
import protocoloRAP.*;
import java.io.*;

public final class TxRx extends Thread implements InterfazMetodosHilos,
       InterfazMensajes{
    private CanalRFCOMM canalRFCOMM;
    private byte[] tramaRecibida;
    private boolean ejecutar;
    private int s;
    private EsperaServidor esperaServidor;
    private ControlRobot controlRobot;
    private Alerta colorRojo,colorAzul,errorTramaColor;
    private boolean colorEstablecidoRojo;
    private Log log,logAux;

    public TxRx (CanalRFCOMM canalRFCOMM,ControlRobot controlRobot,
          EsperaServidor esperaServidor,Alerta colorRojo,Alerta colorAzul,
          Alerta errorTramaColor,Log log,Log logAux,boolean iniciarThread) {
       this.controlRobot = controlRobot;
       this.esperaServidor = esperaServidor;
       this.colorAzul = colorAzul;
       this.colorRojo = colorRojo;
       this.errorTramaColor = errorTramaColor;
       this.canalRFCOMM = canalRFCOMM;
       this.tramaRecibida = null;
       this.ejecutar = true;
       this.colorEstablecidoRojo = true;
       this.s = 0;
       this.log = log;
       this.logAux = logAux;
       if(iniciarThread)
         this.start();
    }
    public void run(){
       this.iniciarRx();
    }
      private void iniciarRx() {
       while(this.ejecutar != false) {
           synchronized (this) {
            try {
               this.wait(1);
                  this.administradorRx();
              } catch (Exception ex) {
              }
             }
          }
    }
    private synchronized void txTrama(byte[] trama) throws IOException {
       this.canalRFCOMM.transmitirTrama(trama);
    }

 private synchronized byte[] rxTemporizadorAcuse()
          throws InterruptedException{
       boolean estado = false;
       while(this.tramaRecibida == null){
          this.wait(TxRx.VALOR_TEMPORIZADOR);
          if(this.tramaRecibida == null ){
             estado = false;
```

```
            break;
        }else{
            estado = true;
            break;
        }
    }
    if (estado == true) {
        byte[] tramaRecibidaC = this.tramaRecibida;
        this.tramaRecibida = null;
        this.notifyAll();
        return tramaRecibidaC;
    } else {
        this.esperaServidor.mostrarFormulario();
        byte[] tramaRecibidaC = {TxRx.SIN_RESPUESTA};
        this.notifyAll();
        return tramaRecibidaC;
    }
}
public byte controlFlujo(byte[] trama,int repeticiones) throws
        IOException, InterruptedException {
    System.out.println("Se transmitio trama");
    byte[] tramaRec = {TxRx.SIN_RESPUESTA};
    byte ack = 0;
    this.txTrama(trama);
    for(int i = 0; i < repeticiones;i++) {
        tramaRec = this.rxTemporizadorAcuse();
            if(tramaRec[0] != TxRx.SIN_RESPUESTA){
                System.out.println("Se recibio trama");
                OperacionesTramaclt op = Protocolo.obtenerSAP().
                        obtenerOperacionesTrama();
                byte[] parametrosTrama = op.
                     obtenerParamTramaMinLong(tramaRec);
                if (this.verificarParametrosAcuse(parametrosTrama)== true){
                    if (verificarParametrosLongMin(trama,
                        parametrosTrama)== true){
                        ack = 1;
                        break;
                    }else{
                        ack = 0;
                        if (i == repeticiones - 2){
                            this.txTrama(trama);
                            System.out.println("retransmitiendo ...");
                            this.log.anadirString("rtx ...\n");
                            this.logAux.anadirString("rtx ...\n");
                        }else{

                             break;
                        }
                    }
                }else {
                    ack = 2;
                    if (i == repeticiones - 2){
                        this.txTrama(trama);
                        System.out.println("retransmitiendo ...");
                        this.log.anadirString("rtx ...\n");
                        this.logAux.anadirString("rtx ...\n");
                    }else{
                        break;
```

```
                    }
                }
            }else{
                System.out.println("No recibio trama");
                if (i == repeticiones - 2){
                    this.txTrama(trama);
                  System.out.println("retransmitiendo ...");
                  this.log.anadirString("rtx ...\n");
                  this.logAux.anadirString("rtx ...\n");
                }else{
                    break;
                }
            }
        }
        if(ack == 1) {
            this.controlRobot.mostrarFormulario();
            byte ackTransferencia = TxRx.ACK_TRANS_TRAMA;
            return ackTransferencia;
        }else if(ack == 2){
            byte nackTransferencia = TxRx.NACK_TRANS_TRAMA;
            return nackTransferencia;
        }else{
            return TxRx.SIN_RESPUESTA;
        }
 }
private synchronized void notificarAcuseRecibido (byte[] trama) throws InterruptedException {
        while(this.tramaRecibida != null) {
            this.wait();
        }
        this.tramaRecibida = trama;
        this.notifyAll();
    }
   public synchronized void establecerSecuenciaEnviada(int s){
      this.s = s;
   }
   private boolean chequearBuffer(int tamanioTrama) throws IOException{
      int datosDisponibles = this.canalRFCOMM.obtenerStreamEntrada().
              available();
       if (datosDisponibles!= tamanioTrama) {
              return false;
      }else{
              return true;
      }
    }
   private void administradorRx() throws IOException, InterruptedException{
      if(this.chequearBuffer(1)){
        byte[] tramaMinL = this.canalRFCOMM.recibirTrama();
        this.notificarAcuseRecibido(tramaMinL);
       }else if(this.chequearBuffer(2)){
          byte[] tramaMaxL = this.canalRFCOMM.recibirTrama();
          OperacionesTramaclt op = Protocolo.obtenerSAP().
                          obtenerOperacionesTrama();
          byte[] respuesta = op.obtenerParamTramaMaxLong(tramaMaxL);
          byte color = respuesta[1];
          if (respuesta[3]==0){
              if (respuesta[0]==0){
                  if(color == TxRx.COLOR_ROJO){
                      if(this.colorEstablecidoRojo){
```

```
                    this.controlRobot.obtenerGaugeTomar().setValue(1);
                }
                this.colorRojo.mostrarAlerta();
            }else if(color == TxRx.COLOR_AZUL){
                if(this.colorEstablecidoRojo == false){
                    this.controlRobot.obtenerGaugeTomar().setValue(1);
                }
                this.colorAzul.mostrarAlerta();
            }else{
            }
        }else if (respuesta[0]==1){
        }
    }else {
        this.errorTramaColor.mostrarAlerta();
    }
  }else{
  }
}
private boolean verificarParametrosAcuse (byte[] parametros){
    boolean acuse = true;
    if (parametros[2]==(byte)0){
        acuse = true;
    }else {
        acuse = false;
    }
    return acuse;
}
private boolean verificarParametrosLongMin(byte[] tramaEnviada, byte[]parametros) {
  int verificarClase = 0;
  boolean verificar = true;
   int  secTrama = (tramaEnviada[0] >>> 5)&1;
   int claseTramaEnviada =  (tramaEnviada[0] >>> 4)&1;
   int tramaEnviadaEntero = Protocolo.obtenerSAP().
         obtenerOperacionesTrama().arregloBytesInt(tramaEnviada);
   if (claseTramaEnviada==0){
      int color = (tramaEnviadaEntero >>> 4)& 31;
      if(color == TxRx.COLOR_AZUL){
         verificarClase = 3;
      }else if(color == TxRx.COLOR_ROJO){
         verificarClase = 3;
      }else{
        verificarClase = 1;
      }
  }
  if (claseTramaEnviada==1){
     verificarClase = 2;
  }
   if ( secTrama ==parametros[0]&& parametros[1]== verificarClase){
      verificar = true;
   }else{
      verificar = false ;
   }
   return verificar;
}
public synchronized void establecerEstadoColorRojo(boolean estado){
    this.colorEstablecidoRojo = estado;
}
public synchronized void terminarHilo() {
```

```
        this.ejecutar = false;
        this.notifyAll();
    }
}
```

ANEXO D: PROGRAMA DESARROLLADO PARA EL BRAZO ROBOT *LEGO MINDSTORMS*

D.1. Paquete AppRobot.principal

D.1.1. Clase Principal

```
package AppRobot.principal;

import AppRobotServer.utileria.pantallas.Presentacion;

public class Principal {
            public static void main(String[] args) throws InterruptedException {
           Presentacion.iniciarPresentacion();
       }
}
```

D.2. Paquete AppRobotServer.archivo

D.2.1. Clase Archivo

```
package AppRobotServer.archivo;

import java.io.*;

public final class Archivo {
    private File fl;
    private String nombreArchivo;
    public Archivo(String nombreArchivo){
          this.nombreArchivo = nombreArchivo;
          this.fl = new File(this.nombreArchivo);
   }
    private byte[] extraerBytes() throws FileNotFoundException, IOException{
        InputStream is;
        is = new FileInputStream(this.fl);
        long longitudFl = this.fl.length();
        byte[] bytes = new byte[(int)longitudFl];
        int offset = 0;
        int numLeido = 0;
        while (offset < bytes.length
              && (numLeido=is.read(bytes, offset, bytes.length-offset)) >= 0) {
                    offset += numLeido;
        }
        is.close();
        return bytes;
    }
    public int leerArchivo(){
        int dato = 0;
        try {
           byte[] bytes = this.extraerBytes();
           dato = this.arregloBytesAInt(bytes);
        } catch (FileNotFoundException ex) {
        } catch (IOException ex) {
        }
        return dato;
```

```
    }
    public void escribirArchivo(int dato){
        try {
            OutputStream os = new FileOutputStream(this.fl, false);
            byte [] arreglo =  intArregloBytes (dato);
            os.write(arreglo);
            os.close();
        } catch (Exception ex) {
        }
    }
    public boolean existeArchivo(){
        return this.fl.exists();
    }
    public int arregloBytesAInt(byte[] arreglo){
        int valor = 0;
            for (int k = 0; k < arreglo.length; k++){
                valor = (valor << 8) + (arreglo[k] & 0xff);
            }
            return valor;
    }
    public byte[] intArregloBytes(int valor){
     byte[] resultado;
     byte b3 = (byte)(( valor >> 24) & 0xff) ;
     byte b2 = (byte)(( valor >> 16) & 0xff) ;
     byte b1 = (byte)(( valor >> 8 ) & 0xff) ;
     byte b0 = (byte)( valor  & 0xff) ;
     resultado = new byte[]{ b3 , b2 , b1 , b0 };
     return resultado;
    }
}
```

D.3. Paquete AppRobotServer.control.sensores

D.3.1. Clase Sensores

```
package AppRobotServer.control.sensores;

import AppRobotServer.utileria.interfaces.*;
import AppRobotServer.control.servos.*;
import lejos.nxt.*;

public final class Sensores extends Thread implements InterfazMensajes  {
    private boolean ejecutar,rojo;
    private TouchSensor ts;
    private LightSensor ls;
    private byte[] color;
    private Servos svs;
    public Sensores(boolean iniciarSensores){
        this.svs = new Servos();
        this.color = null;
        this.rojo = true;
        this.ts = new TouchSensor(SensorPort.S4);
        this.ls = new LightSensor(SensorPort.S3,false);
        this.ejecutar = true;
        if(iniciarSensores)
            this.start();
    }
    public void run(){
```

```
    this.iniciarSensores();
 }
 private void iniciarSensores(){
       while (this.ejecutar != false){
          synchronized(this){
             try {
                this.wait(1);
                this.identificarColor();
             } catch (InterruptedException ex) {
          }
         }
      }
 }
private synchronized void identificarColor(){
   if(this.ts.isPressed()){
      try {
         this.svs.pararAA();
         Sound.beepSequence();
         this.ls.setFloodlight(true);
         Thread.sleep(100);
         int valorLS = this.ls.readValue();
         Thread.sleep(100);
         this.ls.setFloodlight(false);
         this.svs.moverArriba();
         while(true){
            if(this.ts.isPressed()!= true){
               this.svs.pararAA();
               break;
            }else{
               continue;
            }
         }
          if(valorLS <= 55 && valorLS >= 40){
             byte[] colorRojo = {Sensores.COLOR_ROJO};
             this.establecerColor(colorRojo);
             if(this.rojo){
                this.svs.cerrarMano();
             }
          }else if(valorLS <= 35 && valorLS >= 25) {
             byte[] colorAzul = {Sensores.COLOR_AZUL};
             this.establecerColor(colorAzul);
             if(this.rojo == false){
                this.svs.cerrarMano();
             }
          }else {
          }
      } catch (InterruptedException ex) {
      }
   }else{
   }
}
public synchronized void establecerColor(byte[] color){
   this.color = color;
}
public Servos obtenerServos(){
   return this.svs;
}
public synchronized byte[] obtenerColor(){
```

```
        return this.color;
    }
    public synchronized void establecerColorMano(boolean rojo){
        this.rojo = rojo;
    }
    public synchronized void terminarHilo() {
        this.ejecutar = false;
        this.notifyAll();
    }
}
```

D.4. Paquete AppRobotServer.control.servos

D.4.1. Clase Servos

```
package AppRobotServer.control.servos;

import AppRobotServer.utileria.interfaces.*;
import protocoloRAP.singleton.*;
import lejos.nxt.*;

public final class Servos implements InterfazMensajes {
    private static final int VELOCIDAD_MOTOR_A = 720;
    private static final int VELOCIDAD_MOTOR_B = 720;
    private static final int VELOCIDAD_MOTOR_C = 360;
    private boolean abierta;
    public Servos(){
        this.abierta = true;
        Motor.A.resetTachoCount();
        Motor.A.setSpeed(VELOCIDAD_MOTOR_A);
        Motor.B.resetTachoCount();
        Motor.B.setSpeed(VELOCIDAD_MOTOR_B);
        Motor.C.setSpeed(VELOCIDAD_MOTOR_C);
    }
    public void moverIzquierda(){
        Protocolo.obtenerSAP().obtenerTacometro()
            .establerDireccionMA(Servos.IZQUIERDA);
          Motor.A.backward();
    }
    public void moverDerecha(){
          Protocolo.obtenerSAP().obtenerTacometro()
              .establerDireccionMA(Servos.DERECHA);
          Motor.A.forward();
    }
    public void pararID(){
        Protocolo.obtenerSAP().obtenerTacometro()
              .establerDireccionMA(Servos.PARADO);
        Motor.A.stop();
    }
    public void moverArriba(){
        Protocolo.obtenerSAP().obtenerTacometro()
              .establerDireccionMB(Servos.ARRIBA);
        Motor.B.forward();
    }
    public void moverAbajo(){
        Protocolo.obtenerSAP().obtenerTacometro()
              .establerDireccionMB(Servos.ABAJO);
        Motor.B.backward();
```

```
    }
    public void pararAA(){
        Protocolo.obtenerSAP().obtenerTacometro()
                .establerDireccionMB(Servos.PARADO);
        Motor.B.stop();
    }
    public void abrirMano(){
        try {
            for(int i = 0; i < 4 ; i++){
                Motor.C.backward();
                Thread.sleep(50);
                Motor.C.stop();
            }
        } catch (InterruptedException ex) {
        }
        this.abierta = true;
    }
    public void cerrarMano(){
        try {
            for(int i = 0; i < 4 ; i++){
                Motor.C.forward();
                Thread.sleep(50);
                Motor.C.stop();
            }
        } catch (InterruptedException ex) {
        }
       this.abierta = false;
    }
    public synchronized boolean obtenerEstadoMano(){
        return this.abierta;
    }
}
```

D.5. Paquete AppRobotServer.utileria.interfaces

D.5.1. Interfaz InterfazMensajes

```
package AppRobotServer.utileria.interfaces;

public interface InterfazMensajes {
    public static final byte P_ESTABLECIMIENTO_CONEXION  = 1;
    public static final byte P_TRANSFERENCIA_DATOS = 2;
    public static final byte P_FINALIZACION_CONEXION  = 3;
    public static final boolean ACK_ESTABLECIMIENTO_CONEXION  = true;
    public static final boolean NACK_ESTABLECIMIENTO_CONEXION  = false;
    public static final boolean ACK_FINALIZACION_CONEXION  = true;
    public static final boolean ERROR = false;
    public static final byte ACK_TRANS_TRAMA = 4;
    public static final byte NACK_TRANS_TRAMA = 5;
    public static final boolean ACK_TRANS_TRAMA_SAP = true;
    public static final boolean NACK_TRANS_TRAMA_SAP = false;
    public static final byte COLOR_ROJO = 10;
    public static final byte COLOR_AZUL = 15;
    public static final String NOMBRE_ARCHIVO_A = "motorA.txt";
    public static final String NOMBRE_ARCHIVO_B = "motorB.txt";
    public static final byte IZQUIERDA = 6;
    public static final byte PARADO = 7;
    public static final byte DERECHA = 8;
```

```
    public static final byte ARRIBA = 9;
    public static final byte ABAJO = 11;
}
```

D.5.2. Interfaz MetodosHilos

```
package AppRobotServer.utileria.interfaces;


public interface MetodosHilos {
    public abstract void terminarHilo();
}
```

D.6. Paquete AppRobotServer.utileria.pantallas

D.6.1. Clase MenuProtocolo

```
package AppRobotServer.utileria.pantallas;

import AppRobotServer.utileria.interfaces.*;
import protocoloRAP.singleton.*;
import lejos.util.*;
import lejos.nxt.*;

public final class MenuProtocolo implements InterfazMensajes {
  public static void iniciarMenuProtocolo() throws InterruptedException{
            String[] opciones = {"Servidor","Salir"};
            TextMenu menu = new TextMenu (opciones,0,"MENU");
            int a = menu.select();
            switch(a){
               case 0:
                 Protocolo.obtenerInstanciaProtocolo();
                 LCD.clearDisplay();
                 Protocolo.obtenerSAP().
                 peticionServicio(MenuProtocolo.P_ESTABLECIMIENTO_CONEXION);
                 boolean respuestaSAP = Protocolo.obtenerSAP().
                         confirmacionServicio();
                  if(respuestaSAP){
                     LCD.clearDisplay();
                     LCD.drawString("Cliente", 0, 1);
                     LCD.drawString("  conectado !!!", 0, 2);
                  }else{
                  }
                  break;
               case 1:
                  System.exit(0);
                  break;
               default:
                  System.exit(0);
            }
    }
}
```

D.6.2. Clase Presentacion

```
package AppRobotServer.utileria.pantallas;
```

```
import lejos.nxt.*;

public final class Presentacion {
   public static void iniciarPresentacion() throws InterruptedException{
            LCD.drawString("PROYECTO", 4, 0);
            LCD.drawString("DE", 7, 1);
            LCD.drawString("TITULACION", 3, 2);
            LCD.drawString("Elsa Roldan M.", 1, 4);
            LCD.drawString("Marco Almeida P.", 1, 5);
            LCD.drawString(">Presione ENTER", 1, 7);
            int opcion = Button.waitForPress();
            switch(opcion){
               case 1:
                  LCD.clearDisplay();
                  MenuProtocolo.iniciarMenuProtocolo();
                  break;
               default:
            }
            Button.waitForPress();
   }
}
```

D.7. Paquete protocoloRAP.SAP

D.7.1. Clase PuntoAccesoServicio

```
package protocoloRAP.SAP;

import protocoloRAP.servicios.OperacionesTramas.*;
import protocoloRAP.servicios.escuchaSensores.*;
import protocoloRAP.servicios.escuchaBotones.*;
import protocoloRAP.servicios.transmision.*;
import AppRobotServer.utileria.interfaces.*;
import protocoloRAP.servicios.tacometro.*;
import AppRobotServer.control.sensores.*;
import protocoloRAP.servicios.conexion.*;
import AppRobotServer.control.servos.*;
import protocoloRAP.canalRFCOMM.*;
import AppRobotServer.archivo.*;
import lejos.nxt.*;
import java.io.*;

public final class PuntoAccesoServicio extends Thread implements
      InterfazMensajes{
   private CanalRFCOMM canalRFCOMM;
   private ConexionBTsvr con;
   private boolean ejecutar;
   private byte[] servicio;
   private Sensores sns;
   private Servos svs;
   private TxRx txrx;
   private OperacionesTramasvr oT;
   private byte[] tramaEnviar;
   private EscuchaSensores escuchaSensores;
   private Tacometro tacometro;
   private Archivo archivoMotorA,archivoMotorB;
   private EscuchaBotones escuchaBotones;
   public PuntoAccesoServicio(boolean iniciarSAP){
```

```
        this.txrx = null;
        this.sns = null;
        this.svs = null;
        this.canalRFCOMM = null;
        this.tramaEnviar = null;
        this.escuchaSensores = null;
        this.tacometro = null;
        this.archivoMotorA = new Archivo (PuntoAccesoServicio.NOMBRE_ARCHIVO_A);
        this.archivoMotorB = new Archivo (PuntoAccesoServicio.NOMBRE_ARCHIVO_B);
        this.oT = new OperacionesTramasvr();
        this.ejecutar = true;
        this.con = new ConexionBTsvr(false);
        if(iniciarSAP)
            this.start();
    }

    public void run(){
        this.iniciarSAP();
    }
    private void iniciarSAP() {
      while(this.ejecutar != false) {
              synchronized (this) {
                 try {
                    this.wait();
                 } catch (InterruptedException ex) {
                 }
              }
         }
     }
    public synchronized void peticionServicio(byte mensaje){
        while( this.servicio != null){
            try {
               this.wait();
            } catch (InterruptedException ex) {
            }
        }
            byte[] mensajeC = {mensaje};
            this.servicio = mensajeC;
            this.notifyAll();
        }
public synchronized boolean confirmacionServicio() throws InterruptedException{
      while( this.servicio == null){
            this.wait();
        }
        boolean respuesta = this.determinacionServicio(this.servicio);
        this.servicio = null;
        return respuesta;
    }
     private boolean determinacionServicio(byte[] peticion) {
        switch(peticion[0]){
            case PuntoAccesoServicio.P_ESTABLECIMIENTO_CONEXION:
               this.con.start();
               try{
                  boolean estadoCon = this.con.conexionEstablecida();
                  if(estadoCon){
                     this.canalRFCOMM = this.con.obtenerCanalRFCOMM();
                     this.txrx = new TxRx (this.canalRFCOMM,true);
                     this.sns = this.txrx.obtenerSensores();
```

```
            this.escuchaSensores = new EscuchaSensores(this.sns,true);
            this.svs = this.txrx.obtenerServos();
            this.tacometro = new Tacometro(true);
            this.escuchaBotones = new EscuchaBotones();
            return PuntoAccesoServicio.ACK_ESTABLECIMIENTO_CONEXION;
        }else{
            return PuntoAccesoServicio.NACK_ESTABLECIMIENTO_CONEXION;
        }
    }catch(Exception ex){
            return PuntoAccesoServicio.NACK_ESTABLECIMIENTO_CONEXION;
    }
case PuntoAccesoServicio.P_TRANSFERENCIA_DATOS:
    byte[] trama = this.tramaEnviar;
    try {
        this.txrx.txTrama(trama);
    } catch (IOException ex) {
        return PuntoAccesoServicio.NACK_TRANS_TRAMA_SAP;
    }
    return PuntoAccesoServicio.ACK_TRANS_TRAMA_SAP;
case PuntoAccesoServicio.P_FINALIZACION_CONEXION:
    this.svs.pararAA();
    this.svs.pararID();
    boolean estadoMano = this.svs.obtenerEstadoMano();
    if(estadoMano == false){
        this.svs.abrirMano();
    }else{
    }
    int tacoA = this.tacometro.obtenerPosicionMA();
    int tacoB = this.tacometro.obtenerPosicionMB();
    if (tacoA < 0){
        this.svs.moverDerecha();
        while(true){
            tacoA = this.tacometro.obtenerPosicionMA();
            if(tacoA >= 0){
                break;
            }
        }
        this.svs.pararID();
    }else{
        this.svs.moverIzquierda();
        while(true){
            tacoA = this.tacometro.obtenerPosicionMA();
            if(tacoA <= 0){
                break;
            }
        }
        this.svs.pararID();
    }
    if (tacoB < 0){
        this.svs.moverArriba();
        while(true){
            tacoB = this.tacometro.obtenerPosicionMB();
            if(tacoB >= 0){
                break;
            }
        }
        this.svs.pararAA();
    }
```

```
            boolean existeArchivoA = this.archivoMotorA.existeArchivo();
            boolean existeArchivoB = this.archivoMotorB.existeArchivo();
            if(existeArchivoA){
               this.archivoMotorA.escribirArchivo(tacoA);
            }else{
            }
            if(existeArchivoB){
               this.archivoMotorB.escribirArchivo(tacoB);
            }else{
            }
            this.escuchaBotones.establecerFinConexion(true);
            this.finalizarConexion();
            return PuntoAccesoServicio.ACK_FINALIZACION_CONEXION;
         default:
            this.svs.pararAA();
            this.svs.pararID();
            LCD.clearDisplay();
            LCD.drawString("Error", 0, 0);
            this.escuchaBotones.establecerFinConexion(true);
            this.finalizarConexion();
            return PuntoAccesoServicio.ERROR;
      }
   }
   private void finalizarConexion(){
      if(this.archivoMotorA != null){
         this.archivoMotorA = null;
      }
      if(this.archivoMotorB != null){
         this.archivoMotorB = null;
      }
      if(this.tacometro.isAlive()){
         this.tacometro.terminarHilo();
         this.tacometro = null;
      }else{
         this.tacometro = null;
      }
      if(this.escuchaSensores.isAlive()){
         this.escuchaSensores.terminarHilo();
         this.escuchaSensores = null;
      }else{
         this.escuchaSensores = null;
      }
      if(this.sns.isAlive()){
         this.sns.terminarHilo();
         this.sns = null;
      }else{
         this.sns = null;
      }
      if(this.svs != null){
         this.svs = null;
      }
      if(this.oT != null){
         this.oT = null;
      }
      if(this.txrx != null){
         if(this.txrx.isAlive()){
           this.txrx.terminarHilo();
```

```
            this.txrx = null;
          }else{
            this.txrx = null;
          }
        }
        if(this.canalRFCOMM != null){
          this.canalRFCOMM.desconectar();
          this.canalRFCOMM = null;
        }
        if(this.con != null){
          this.con = null;
        }
        this.terminarHilo();
        System.gc();
    }
    public OperacionesTramasvr obtenerOperacionesTrama(){
        return this.oT;
    }
    public synchronized void establecerTramaEnviar(byte[] tramaEnviar){
        this.tramaEnviar = tramaEnviar;
    }
    public synchronized Archivo obtenerArchivoA(){
        return this.archivoMotorA;
    }
    public synchronized Archivo obtenerArchivoB(){
        return this.archivoMotorB;
    }
    public Tacometro obtenerTacometro(){
        return this.tacometro;
    }
    public synchronized void terminarHilo() {
        this.ejecutar = false;
        this.notifyAll();
    }
}
```

D.8. Paquete protocoloRAP.canalRFCOMM

D.8.1. Clase CanalRFCOMM

```
package protocoloRAP.canalRFCOMM;

import javax.microedition.io.*;
import java.io.*;

public final class CanalRFCOMM {
    private StreamConnection strcon;
    private DataInputStream  streamEntrada;
    private DataOutputStream streamSalida;
    public CanalRFCOMM (StreamConnection strcon) {
        this.strcon = strcon;
        this.streamEntrada = this.strcon.openDataInputStream();
        this.streamSalida = this.strcon.openDataOutputStream();
    }
     public byte[] recibirTrama()throws IOException {
        if(this.streamEntrada.available() == 0){
            return null;
        } else {
```

```
            byte tramaRecibida[] = new byte[this.streamEntrada.available()];
            this.streamEntrada.read(tramaRecibida);
            return tramaRecibida;
        }
    }
    public void enviarTrama (byte[] trama) {
        final DataOutputStream dout = this.streamSalida;
        final byte[] datos = trama;
        Runnable env = new Runnable() {
            public void run() {
                try {
                    dout.write(datos);
                    dout.flush();
                } catch (IOException ex) {
                }
            }
        };
        Thread tenv = new Thread(env);
        tenv.start();
    }
    public DataInputStream obtenerStreamEntrada(){
        return this.streamEntrada;
    }
    public DataOutputStream obtenerStreamSalida(){
        return this.streamSalida;
    }
    public StreamConnection obtenerStreamConexion(){
        return this.strcon;
    }
    public void desconectar() {
        try{
            if(this.streamEntrada != null){
                this.streamEntrada.close();
            }
        }catch(Exception e) {
        }
        try {
            if(this.streamSalida != null)
                this.streamSalida.close();
        }catch(Exception e) {
        }
        try {
            if( this.strcon != null)
                this.strcon.close();
        }catch(Exception e) { }
            this.streamEntrada = null;
            this.streamSalida = null;
            this.strcon = null;
    }
}
```

D.9. Paquete protocoloRAP.servicios.OperacionesTramas

D.9.1. Clase OperacionesTramasvr

```
package protocoloRAP.servicios.OperacionesTramas;

public class OperacionesTramasvr {
```

```
    public OperacionesTramasvr(){
    }
     private  int obtenerCRCInt(byte [] arreglo){
          int crc = 0;
          int i = 0;
          int datos = 0;
          int datosInvertidos= 0;
          int poly = 25;
          int opXor = 0;
          datos = this.arregloBytesAInt(arreglo);
          datosInvertidos = this.obtenerNumero(datos,16);
          if (datosInvertidos%2==0){
             opXor = (int)(datosInvertidos^0);
          }else {
             opXor = (int)(datosInvertidos^poly);
          }
          for (i = 1;i<12;i++){
             crc= (int)(opXor>>>1);
             if (crc % 2 == 0){
                opXor = (int)(crc^0);
             }else {
                opXor = (int)(crc^poly);
                crc = opXor;
           }
          }
          crc= (int)(opXor>>>1);
          crc = obtenerNumero(crc,4);
     return crc;
}
    private  int arregloBytesAInt(byte[] arreglo){
       int valor = 0;
          for (int k = 0; k < arreglo.length; k++){
             valor = (valor << 8) + (arreglo[k] & 0xff);
          }
          return valor;
    }

    private int obtenerNumero(int valor, int bits){
          int[] tabla = {1,2,4,8,16,32,64,128,256,512,
                    1024,2048,4096,8192,16384,32768};
          int numero = valor;
          int nBits = bits;
          int numeroInvertido =0;
           for (int j = nBits-1; j>=0;j--){
             if (numero % 2 != 0){
               numeroInvertido = numeroInvertido + tabla[j];
             }
             numero = numero>>>1;
           }
          return numeroInvertido;
     }
    private  byte[] intArregloBytes(int valor){
     byte[] resultado;
     byte b1 = (byte)(( valor >> 8 ) & 0xff) ;
     byte b0 = (byte)( valor  & 0xff) ;
     resultado = new byte[]{ b1 , b0 };
     return resultado;
    }
```

```
public byte[] tramaColor (byte[] valorColor,int sec){
    byte color = valorColor[0];
    int crc = 0;
    int TramaEnteros = 0;
    byte[] trama = {(byte)64,(byte)0};
    switch (color){
        case 10:
            trama[1] = (byte)160;
        break;
        case 15:
            trama[1] = (byte)240;
        break;
        default:
     }
         if(sec ==1){
            trama[0] = (byte) (trama[0] | (byte) 32);
         }
         TramaEnteros = this.arregloBytesAInt(trama);
         crc = this.obtenerCRCInt(trama);
         TramaEnteros = TramaEnteros + crc;
         System.out.println("Tx color (int):" + "\n");
         System.out.println(TramaEnteros + "\n");
         System.out.println("Tx color (bin):" + "\n");
         System.out.println(Integer.toBinaryString(TramaEnteros) + "\n");
         trama = this.intArregloBytes(TramaEnteros);
  return trama;
}
   public byte[] tramaFinalizacion(){
    int crc = 0;
    int TramaEnteros = 0;
    byte[] trama = {(byte)80,(byte)0};
    TramaEnteros = this.arregloBytesAInt(trama);
    crc = this.obtenerCRCInt(trama);
    TramaEnteros = TramaEnteros + crc;
    System.out.println("Tx Fin (int):" + "\n");
    System.out.println(TramaEnteros + "\n");
    System.out.println("Tx Fin (bin):" + "\n");
    System.out.println(Integer.toBinaryString(TramaEnteros) + "\n");
    trama = this.intArregloBytes(TramaEnteros);
  return trama;
}
   public byte[] tramaControl (int sec, int claseControl){
    int TramaEnteros = 0;
    byte[] trama = {(byte)0};
    switch (claseControl){
        case 0:
            trama[0] = (byte)196;
        break;
        case 1:
            trama[0] = (byte)200;
        break;
        case 2:
            trama[0] = (byte)208;
        break;
        case 3:
            trama[0] = (byte)216;
        break;
        default:
```

```
        }
            if (claseControl!=0){
                if(sec ==1){
                    trama[0] = (byte) (trama[0] | (byte) 32);
                }
            }
            TramaEnteros = this.arregloBytesAInt(trama);
        return trama;
      }
    public byte[] obtenerParamTramaMaxLong (byte[]trama){
        int TramaRecividaEntero = this.arregloBytesAInt(trama);
        System.out.println("Rx (int):" + "\n" );
        System.out.println(TramaRecividaEntero + "\n");
        System.out.println("Rx (bin):" + "\n");
        System.out.println(Integer.toBinaryString(TramaRecividaEntero) + "\n");
        int tipoTrama = 0;
        int transmisor = 0;
        int sec= 0;
        int dato = 0;
        int comando = 0;
        int color = 0;
        int tipoTramaDatos = 0;
        byte[] tramaAck = {(byte)0};
        byte[] parametrosTrama2 = {(byte)0,(byte)0,(byte)0,(byte)0, (byte)0,(byte)0};
        int fin = 0;
        int ack = 0;
        tipoTrama = (TramaRecividaEntero >>> 15)&1 ;
        int verificarCRC = obtenerCRCInt(trama);
        if (verificarCRC == 0){
            tipoTrama = (TramaRecividaEntero >>> 15)&1 ;
            if(tipoTrama==0){
                transmisor = (TramaRecividaEntero >>> 14)&1;
                if (transmisor==0){
                    sec = (TramaRecividaEntero >>> 13)&1;
                tipoTramaDatos = (TramaRecividaEntero >>> 12)&1;
                    switch (tipoTramaDatos){
                        case 0:
                            dato = (TramaRecividaEntero >>> 4)&31;
                            if (dato == 10){
                                color = dato;
                                tramaAck = this.tramaControl(sec,3);
                            } else if (dato == 15){
                                color = dato;
                                tramaAck = this.tramaControl(sec,3);
                            } else {
                                comando = dato;
                                tramaAck = this.tramaControl(sec,1);
                            }
                        break;
                        case 1:
                            fin = 1;
                            tramaAck = this.tramaControl(sec,2);
                            break;
                        default:
                    }
                   }
            }
        }else {
```

```
        sec = (TramaRecividaEntero >>> 13)&1;
        tramaAck = this.tramaControl(sec,0);
        ack = 1;
      }
      int tramaAckInt = this.arregloBytesAInt(tramaAck);
      parametrosTrama2[0] = (byte)fin;
      parametrosTrama2[1] = (byte)comando;
      parametrosTrama2[2] =  tramaAck[0];
      parametrosTrama2[3] =  (byte)sec;
      parametrosTrama2[4] =  (byte)ack;
      parametrosTrama2[5] =  (byte)color;
      byte [] acuseB = {parametrosTrama2[2]};
      int acuse = this.arregloBytesAInt(acuseB);
      if (ack ==0){
         System.out.println("ACK Tx (int):" + "\n");
         System.out.println(acuse + "\n");
         System.out.println("ACK Tx (bin):" + "\n");
         System.out.println(Integer.toBinaryString(acuse)+ "\n");
      }else{
         System.out.println("NACK Tx (int):" + "\n");
         System.out.println(acuse + "\n");
         System.out.println("NACK Tx (bin):" + "\n");
         System.out.println(Integer.toBinaryString(acuse)+ "\n");
      }
      return parametrosTrama2;
   }
}
```

D.10. Paquete protocoloRAP.servicios.conexion

D.10.1. Clase ConexionBTsvr

```
package protocoloRAP.servicios.conexion;

import protocoloRAP.canalRFCOMM.*;
import javax.microedition.io.*;
import lejos.nxt.comm.*;
import lejos.nxt.*;

public final class ConexionBTsvr extends Thread {
    private boolean inicioConexion;
    private CanalRFCOMM canalRFCOMM;
public ConexionBTsvr(boolean iniciarServidor){
        if(iniciarServidor)
         this.start();
    }
public void run(){
       this.iniciarServidor();
    }
    private void iniciarServidor() {
       try {
          LCD.drawString("Esperando ", 0, 1);
          LCD.drawString("  conexion ...", 0, 2);
          BTConnection btcon = Bluetooth.waitForConnection();
          btcon.setIOMode(NXTConnection.RAW);
          this.canalRFCOMM = new CanalRFCOMM
                ((StreamConnection) btcon);
          synchronized (this) {
```

```
            this.inicioConexion = true;
            notify();
          }
        } catch (Exception ex) {
        }
    }
    public boolean conexionEstablecida() throws InterruptedException{
        int i = 0;
          while(true){
                synchronized(this) {
                this.wait();
          }
           i = i + 1;
           if(i==1){
              break;
           }
     }
        if(this.inicioConexion) {
          return true;
        }else{
          return false;
        }
    }
    public CanalRFCOMM obtenerCanalRFCOMM(){
        return this.canalRFCOMM;
    }
}
```

D.11. Paquete protocoloRAP.servicios.escuchaBotones

D.11.1. Clase EscuchaBotones

```
package protocoloRAP.servicios.escuchaBotones;

import AppRobotServer.utileria.interfaces.*;
import protocoloRAP.singleton.*;
import lejos.nxt.*;

public final class EscuchaBotones implements InterfazMensajes,ButtonListener {
    private boolean finConexion;
    public EscuchaBotones(){
        this.finConexion = false;
        Button.ENTER.addButtonListener(this);
        Button.ESCAPE.addButtonListener(this);
    }
    public void buttonPressed(Button btn) {
        if(btn == Button.ENTER){
            Protocolo.obtenerSAP().peticionServicio
                (EscuchaBotones.P_FINALIZACION_CONEXION);
            try {
                boolean respuesta = Protocolo.obtenerSAP().confirmacionServicio();
                if(respuesta){
                    this.finConexion = true;
                    LCD.clearDisplay();
                    LCD.drawString("Conexion",0,0);
                    LCD.drawString("finalizada !!!",2,1);
                }else{
                    LCD.clearDisplay();
```

```
                LCD.drawString("Error !!!!",0,0);
              }
            } catch (InterruptedException ex) {
            }
        }else if(btn == Button.ESCAPE){
            if(this.finConexion){
               System.exit(0);
            }else{
            }
        }else{
        }
    }

    public void establecerFinConexion(boolean finConexion){
        this.finConexion = finConexion;
    }
    public void buttonReleased(Button btn) {
    }
}
```

D.12. Paquete protocoloRAP.servicios.escuchaSensores

D.12.1. Clase EscuchaSensores

```
package protocoloRAP.servicios.escuchaSensores;

import AppRobotServer.utileria.interfaces.*;
import AppRobotServer.control.sensores.*;
import protocoloRAP.singleton.*;
import lejos.nxt.*;

public class EscuchaSensores extends Thread implements InterfazMensajes {
    private Sensores sns;
    private boolean ejecutar;
    public EscuchaSensores(Sensores sns,boolean iniciarThread){
        this.sns = sns;
        this.ejecutar = true;
        if(iniciarThread)
          this.start();
}
    public void run(){
        this.iniciarEscuchaSensores();
    }
    private void iniciarEscuchaSensores() {
        while(this.ejecutar != false) {
            synchronized (this) {
                try {
                    this.wait(1);
                    byte[] color = this.sns.obtenerColor();
                        if(color != null){
                          byte[] tramaEnviar = Protocolo.obtenerSAP().
                              obtenerOperacionesTrama().tramaColor(color,0);
                          Protocolo.obtenerSAP().establecerTramaEnviar
                                (tramaEnviar);
                          Protocolo.obtenerSAP().peticionServicio
                                (EscuchaSensores.P_TRANSFERENCIA_DATOS);
                          boolean respuestaSAP = Protocolo.obtenerSAP().
                            confirmacionServicio();
```

```
                    if(respuestaSAP){
                        color = null;
                        this.sns.establecerColor(color);
                    }else{
                        LCD.clearDisplay();
                        LCD.drawString("No se tx el color",0,0);
                        color = null;
                        this.sns.establecerColor(color);
                    }
                    color = null;
                    this.sns.establecerColor(color);
                }else{
                }
            } catch (InterruptedException ex) {
            }
        }
    }
  }
  public synchronized void terminarHilo() {
    this.ejecutar = false;
    this.notifyAll();
  }
}
```

D.13. Paquete protocoloRAP.servicios.tacometro

D.13.1. Clase Tacometro

```
package protocoloRAP.servicios.tacometro;

import AppRobotServer.utileria.interfaces.*;
import protocoloRAP.singleton.*;
import lejos.nxt.*;

public final class Tacometro extends Thread implements InterfazMensajes {
  private boolean ejecutar;
  private int contadorA,contadorB,valorArchivoA,
      valorArchivoB,posicionA,posicionB;
  private byte direccionMA, direccionMB;
  public Tacometro(boolean iniciarThread){
    this.ejecutar = true;
    Motor.A.resetTachoCount();
    Motor.B.resetTachoCount();
    this.contadorA = 0;
    this.contadorB = 0;
    this.posicionA = 0;
    this.posicionB = 0;
    this.valorArchivoA = Protocolo.obtenerSAP().
                    obtenerArchivoA().leerArchivo();
    this.valorArchivoB = Protocolo.obtenerSAP().
                    obtenerArchivoB().leerArchivo();
    this.direccionMA = Tacometro.PARADO;
    this.direccionMB = Tacometro.PARADO;
    if(iniciarThread)
      this.start();
  }
  public void run(){
    this.iniciarTacometro();
```

```
}
 private void iniciarTacometro() {
     while(this.ejecutar != false) {
       synchronized (this) {
         try {
               this.wait(1);
               this.contadorA = Motor.A.getTachoCount();
               this.contadorB = Motor.B.getTachoCount();
               this.posicionA = this.contadorA + this.valorArchivoA;
               this.posicionB = this.contadorB + this.valorArchivoB;
               switch (this.direccionMA){
                  case Tacometro.IZQUIERDA:
                     if(this.posicionA <= -11662){
                        Motor.A.stop();
                     }
                     break;
                  case Tacometro.PARADO:
                     break;
                  case Tacometro.DERECHA:
                    if(this.posicionA >= 11662){
                      Motor.A.stop();
                     }
                     break;
                  default:
                }
                switch (this.direccionMB){
                  case Tacometro.ARRIBA:
                     if(this.posicionB >= 0){
                        Motor.B.stop();
                     }
                     break;
                  case Tacometro.PARADO:
                     break;
                  case Tacometro.ABAJO:
                    if(this.posicionB <= -9569){
                      Motor.B.stop();
                     }
                     break;
                  default:
                }
         } catch (Exception ex) {
         }
       }
     }
 }
 public synchronized void establerDireccionMA(byte direccionMA){
    this.direccionMA = direccionMA;
 }
 public synchronized void establerDireccionMB(byte direccionMB){
    this.direccionMB = direccionMB;
 }
 public synchronized int obtenerPosicionMA(){
    return this.posicionA;
 }
 public synchronized int obtenerPosicionMB(){
    return this.posicionB;
 }
public synchronized void terminarHilo() {
```

```
        this.ejecutar = false;
        this.notifyAll();
    }
}
```

D.14. Paquete protocoloRAP.servicios.transmision

D.14.1. Clase TxRx

```
package protocoloRAP.servicios.transmision;

import AppRobotServer.utileria.interfaces.*;
import AppRobotServer.control.sensores.*;
import AppRobotServer.control.servos.*;
import protocoloRAP.canalRFCOMM.*;
import protocoloRAP.singleton.*;
import lejos.nxt.*;
import java.io.*;

public final class TxRx extends Thread implements InterfazMensajes {
    private CanalRFCOMM canalRFCOMM;
    private byte[] tramaRecibida;
    private boolean ejecutar;
    private Sensores sns;
    private Servos svs;
    public TxRx (CanalRFCOMM canalRFCOMM,boolean iniciarThread) {
        this.sns = new Sensores(true);
        this.svs = this.sns.obtenerServos();
        this.canalRFCOMM = canalRFCOMM;
        this.ejecutar = true;
        if(iniciarThread)
          this.start();
    }
    public void run(){
        this.iniciarRx();
    }
     private void iniciarRx() {
          while(this.ejecutar != false) {
            synchronized (this) {
                try {
                    this.wait(1);
                    this.administradorRx();
                } catch (InterruptedException ex) {
                }catch (IOException ex) {
                }
            }
          }
     }
    public synchronized void txTrama(byte[] trama) throws IOException {
        this.canalRFCOMM.enviarTrama(trama);
    }
    public synchronized boolean chequearBuffer(int tamanioTrama)
                                        throws IOException{
        int datosDisponibles = this.canalRFCOMM.
               obtenerStreamEntrada().available();
              if (datosDisponibles != tamanioTrama) {
                   return false;
              }else{
```

```
                return true;
            }
}
private void ejecutarComando(byte valorComando) throws IOException{
  byte comando = valorComando;
   switch (comando){
        case (byte)11:
          this.svs.pararID();
          break;
        case (byte)12:
          this.svs.moverDerecha();
          break;
        case (byte)13:
          this.svs.moverIzquierda();
          break;
        case (byte)17:
          this.svs.moverArriba();
          break;
        case (byte)18:
           this.svs.moverAbajo();
           break;
         case (byte)19:
           this.svs.pararAA();
           break;
         case (byte)30:
           this.svs.abrirMano();
           break;
         case (byte)31:
           this.svs.cerrarMano();
           break;
         default:
   }
}
public Sensores obtenerSensores(){
    return this.sns;
}
public Servos obtenerServos(){
    return this.svs;
}
private void administradorRx() throws IOException, InterruptedException{
      if(this.chequearBuffer(2)){
          this.tramaRecibida = this.canalRFCOMM.recibirTrama();
          byte[] parametros = Protocolo.obtenerSAP().
          obtenerOperacionesTrama().obtenerParamTramaMaxLong(tramaRecibida);
          byte[] acuseEnviar = {parametros[2]};
          Protocolo.obtenerSAP().establecerTramaEnviar(acuseEnviar);
          Protocolo.obtenerSAP().peticionServicio
                    (TxRx.P_TRANSFERENCIA_DATOS);
          boolean respuestaSAP = Protocolo.obtenerSAP().
                      confirmacionServicio();
          if(respuestaSAP == TxRx.ACK_TRANS_TRAMA_SAP){
            if(parametros[4]== 0){
               switch(parametros[0]){
                   case (byte)0:
                    if(parametros[5]!= 0){
                       byte color = parametros[5];
                       if(color == TxRx.COLOR_ROJO){
                          this.sns.establecerColorMano(true);
```

```
                    }else if(color == TxRx.COLOR_AZUL){
                        this.sns.establecerColorMano(false);
                    }else{
                    }
                }else{
                    byte comando = parametros[1];
                    this.ejecutarComando(comando);
                }
                break;
                case (byte)1:
                    Protocolo.obtenerSAP().peticionServicio
                        (TxRx.P_FINALIZACION_CONEXION);
                    boolean respuesta = Protocolo.obtenerSAP().
                            confirmacionServicio();
                    if (respuesta){
                        LCD.clearDisplay();
                        LCD.drawString("Conexion",0,0);
                        LCD.drawString("finalizada !!!",2,1);
                    }
                break;
                default:
            }
        }else {
        }
      }else{
            Protocolo.obtenerSAP().peticionServicio
            (TxRx.P_FINALIZACION_CONEXION);
      }
     }
  }
  public synchronized void terminarHilo() {
     this.ejecutar = false;
     this.notifyAll();
  }
}
```

D.15. Paquete protocoloRAP.singleton

D.15.1. Clase Protocolo

```
package protocoloRAP.singleton;

import protocoloRAP.SAP.*;

public final class Protocolo {
  private static final Protocolo protocoSingleton = new Protocolo();
  private static PuntoAccesoServicio sap;
  private Protocolo(){
     Protocolo.sap = new PuntoAccesoServicio(true);
  }
  public static Protocolo obtenerInstanciaProtocolo() {
     return protocoSingleton;
  }
  public static PuntoAccesoServicio obtenerSAP() {
     return Protocolo.sap;
  }
}
```

ANEXO E: INSTALACIÓN DE LAS HERRAMIENTAS DE SOFTWARE UTILIZADAS EN EL PROYECTO

E.1. CONFIGURACIÓN DEL AMBIENTE *JAVA*

- Descargar e instalar el *Java Developer's Kit (JDK)* de la empresa *ORACLE* que se encuentra disponible en el siguiente *url*:

  http://www.oracle.com/index.html

  La versión recomendada del *JDK* es 1.6.0 actualización 22 o superior.

- Se debe crear una nueva variable de entorno que se denominará *JAVA_HOME* (Panel de control -> Sistema -> Opciones avanzadas->Variables de entorno). Como valor de esta variable se debe establecer el *path* donde se encuentra instalado el *JDK.* Un valor típico se muestra a continuación:

  C:\Archivos de programa\Java\jdk1.6.0_22

  En la Figura E-1 se muestra la creación de la variable de entorno.

- Agregar el *path* del compilador de *Java* como valor a la variable de entorno *PATH*. Un valor típico se muestra a continuación:

  C:\Archivos de programa\Java\jdk1.6.0_22\bin

- Reiniciar el sistema para que los cambios tengan efecto

- Descargar e instalar la versión 6.7.1 de *NetBeans* que contenga el paquete *J2ME*. El instalador se encuentra disponible en el *url*:

  http://netbeans.org/

  En la Figura E-2 se muestra la pantalla de inicio de instalación del *IDE Netbeans* 6.7.1

- Reiniciar el sistema para que los cambios tengan efecto

Figura E-1 Creación de la variable de entorno *Java_Home*

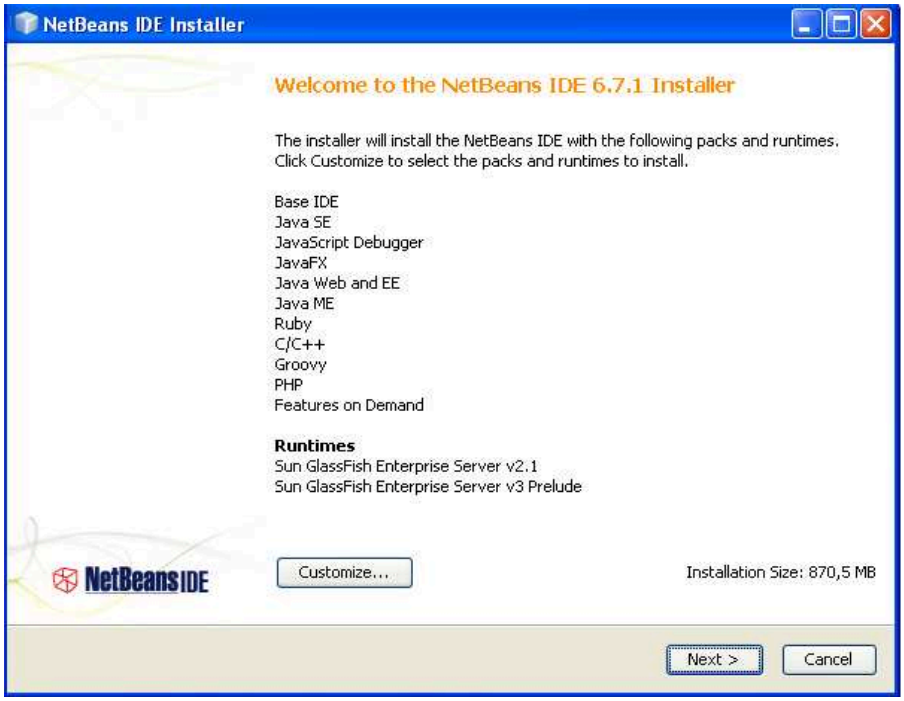


Figura E-2 Pantalla de inicio de instalación del *IDE Netbeans 6.7.1*

E.2. CONFIGURACIÓN DEL AMBIENTE PARA *JSPIN*

- Descargar e instalar el compilador de *C mingw* dentro de la carpeta por defecto (*C:\mingw*). El instalador se encuentra disponible en el siguiente *url:*

  http://mingw.org/

  En la Figura E-3 se muestra la pantalla de inicio de instalación del compilador de *C mingw*.

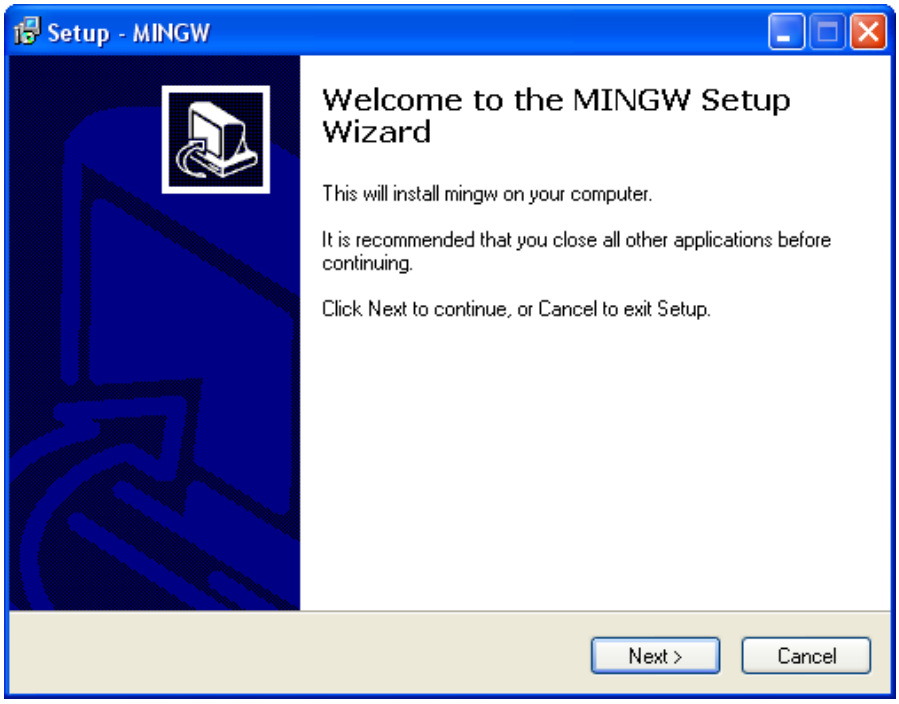


Figura E-3 Pantalla de inicio de instalación del compilador de *C mingw*

- Descargar e instalar *JSPIN* 4.7 en el mismo *path* donde se instaló el compilador de *C mingw.* El instalador se encuentra disponible en el siguiente *url*:

  http://stwww.weizmann.ac.il/g-cs/benari/jspin/

  En la Figura E-4 se muestra la pantalla de inicio de instalación del *software JSPIN 4.7.*

- Reiniciar el sistema para que los cambios tengan efecto

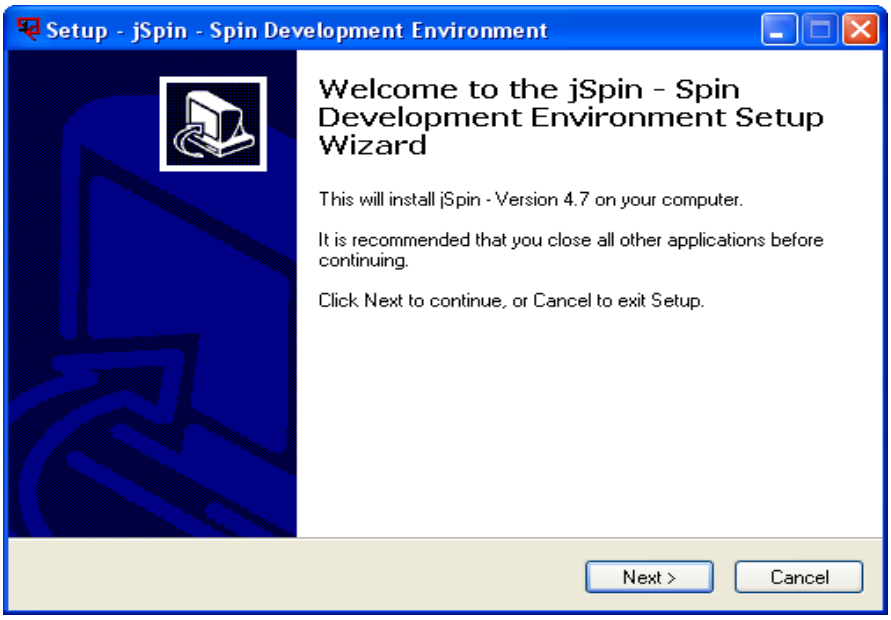


Figura E-4 Pantalla de inicio de instalación del *software JSPIN 4.7.*

E.3. CONFIGURACIÓN DEL AMBIENTE PARA *BLUETOOTH*

- En el mercado existen una gran variedad de dispositivos *Bluetooth.* Para este proyecto se eligió el dispositivo *DBT*-122 el mismo que es un dispositivo *Bluetooth* versión 2 de la marca *Linksys* que trae consigo un *cd rom* que contiene sus respectivos *drivers* y *software* para su instalación. En la Figura E-5 se muestra la pantalla de inicio de instalación del *software* del dispositivo *Bluetooth DBT*-122.

- Reiniciar el sistema para que los cambios tengan efecto

Figura E-5 Pantalla de inicio de instalación del *software* del dispositivo *Bluetooth DBT*-122

E.4. CONFIGURACIÓN DEL AMBIENTE PARA EL *ROBOT LEGO MINDSTORMS*

- El brazo robot adquirido pertenece a la compañía *Lego* y por tanto, el *kit* contiene *cd rom* con respectivos *drivers* y *software* para su instalación.

Figura E-6 Pantalla de inicio de instalación del *software Lego Mindstorms NXT*

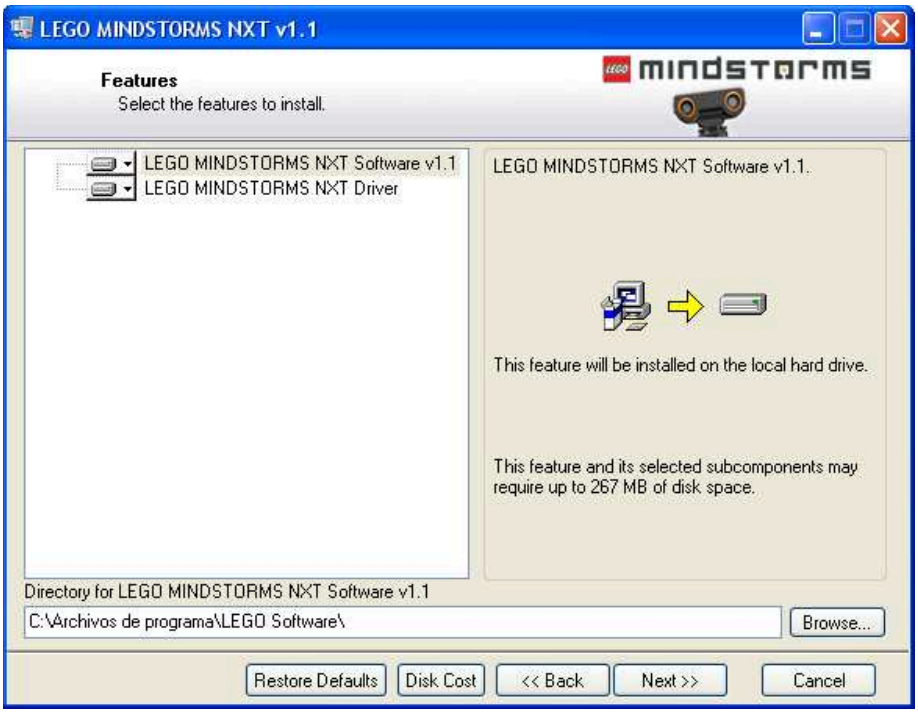


Figura E-7 Pantalla que muestra los componentes del *software Lego Mindstorms* que serán instalados

- Reiniciar el sistema para que los cambios tengan efecto

E.5. CONFIGURACIÓN DEL AMBIENTE PARA *leJOS*

- Descargar e instalar *leJOS* 0.85. El instalador se encuentra disponible en el siguiente *url:*

  http://lejos.sourceforge.net/

  En la Figura E-8 se muestra la pantalla que indica el *path* donde se instalará el *software leJOS*.

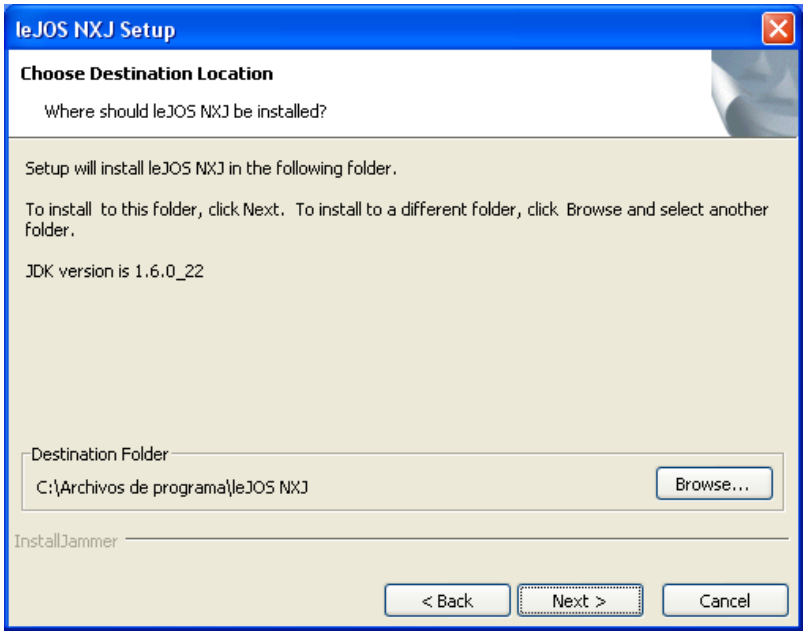


Figura E-8 Pantalla que indica el *path* donde se instalará el *software leJOS*

- Es importante anotar el *path* donde se va a instalar la carpeta correspondiente a los proyectos de *leJOS*, debido a que en esa carpeta se encontrará el *plugin* para desarrollar programas con el *software NetBeans.* En la Figura E-9 se muestra la pantalla que indica dicho *path*.

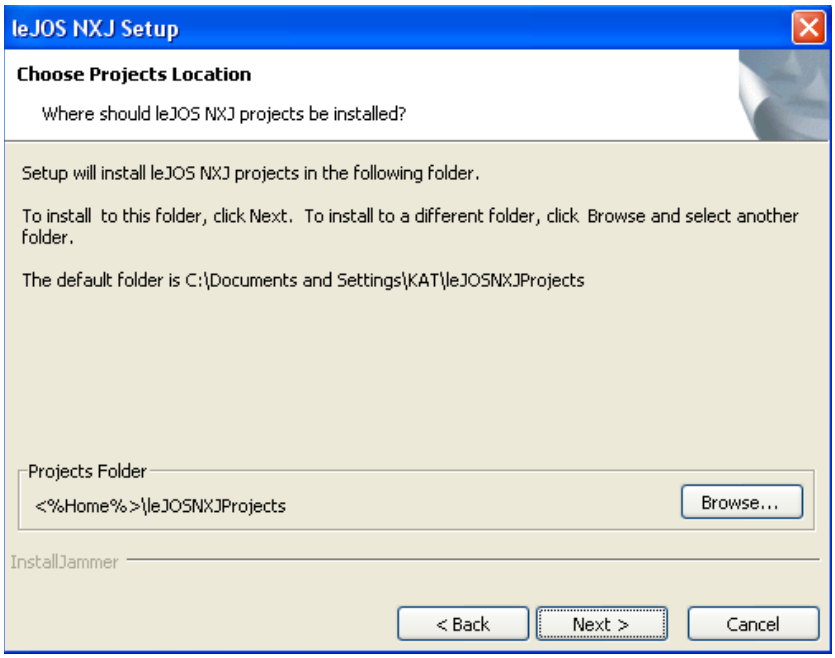


Figura E-9 Pantalla que indica el *path* donde se instalará la carpeta correspondiente a los proyectos de *leJOS*

- Una vez finalizada la instalación se mostraran las pantallas que se muestran en la Figura E-10 y la Figura E-11

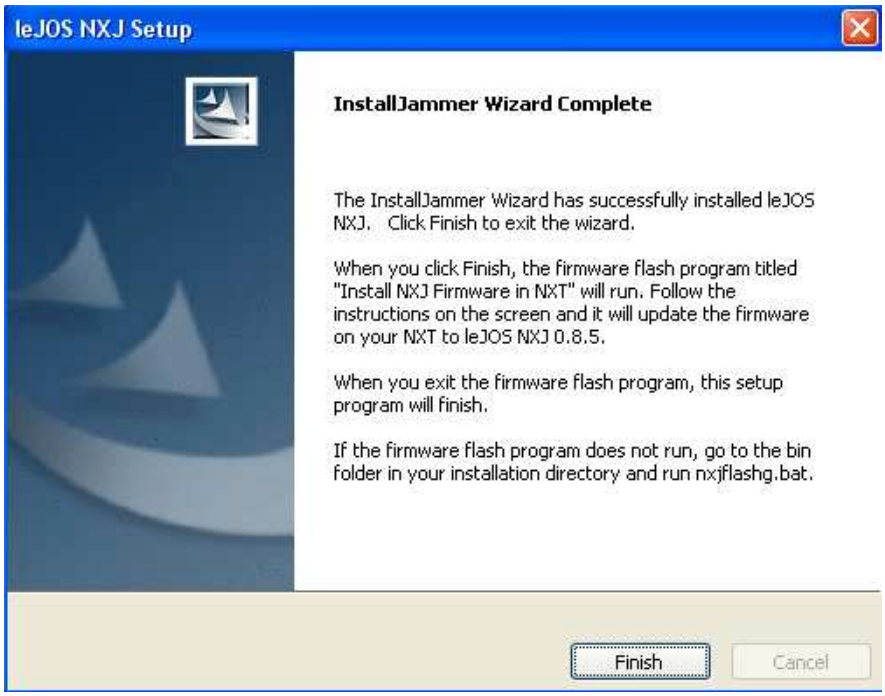


Figura E-10 Pantalla de finalización de la instalación del *software leJOS* en el *PC*

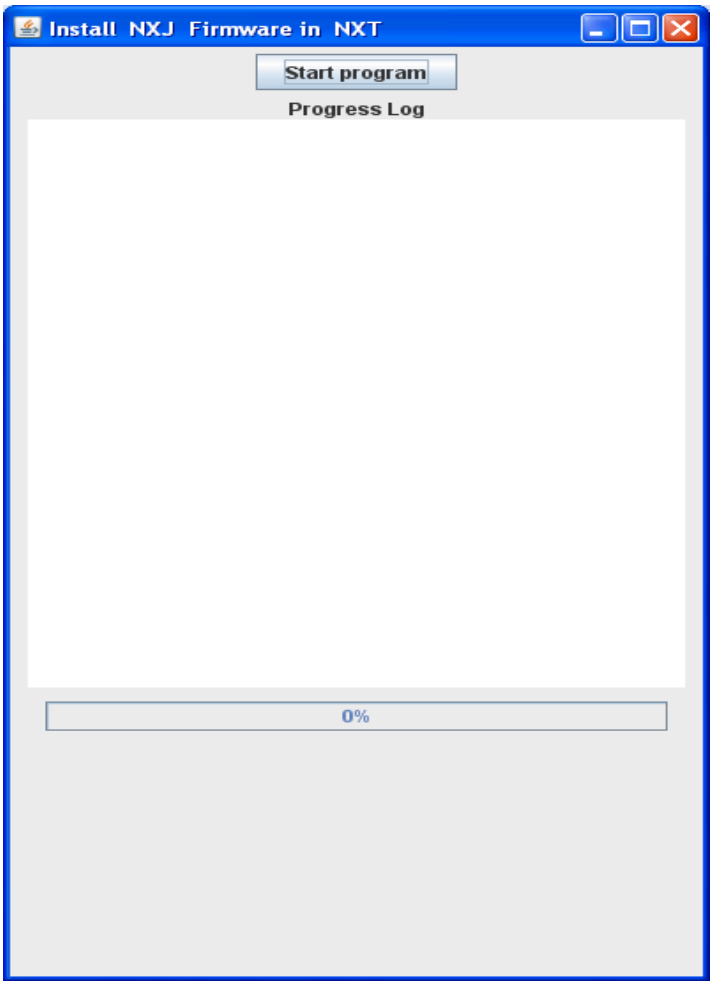


Figura E-11 Pantalla para instalar el firmware *leJOS* en el brazo robot

- Una vez instalado *leJOS* en el *PC*, este permite a los usuarios reemplazar el *firmware* original del brazo robot. Para esto se debe tener conectado el brazo robot al *PC* ya sea utilizando *Bluetooth* o *usb.* Para dar inicio a la transferencia del *firmware leJOS* al brazo robot se debe hacer *clic* en el botón *Start Program* de la pantalla *Install NXJ Firmware in NXT* (Figura E-11)

- Adicionalmente para reemplazar el firmware original del brazo robot por el *firmware leJOS* se tiene el comando *nxjflash* que puede ser usado a través de la línea de comandos. Este comando abre la pantalla de la Figura E-11.

E.6. CONFIGURACIÓN DEL AMBIENTE PARA *NOKIA*

- Descargar e instalar el *software Nokia Pc Suite* para el modelo *Nokia* 5220 que se encuentra disponible en el siguiente *url*:

  www.nokia.com/pcsuite

  En la Figura E-12 se muestra la pantalla de inicio de instalación del *software Nokia Pc Suite*

Figura E-12 Pantalla de inicio de instalación del *software Nokia Pc Suite*

- Descargar e instalar el *Software Developer kit* de la serie 40 de *Nokia* (emulador *Nokia*) que se encuentra disponible en el siguiente *url*:

  www.forum.nokia.com

  En la Figura E-13 se muestra la pantalla de inicio de instalación del *software*.

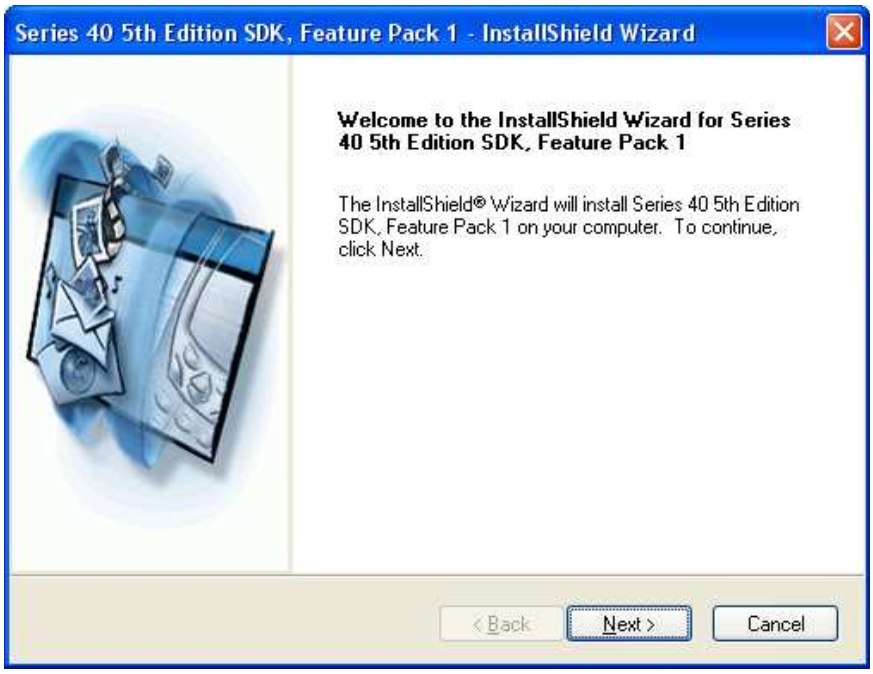


Figura E-13 Pantalla de inicio de instalación del *Software Developer kit* de las series 40 de *Nokia*

- Descargar e instalar el *Software Developer kit* de la serie 40 de *Nokia* (emulador *Nokia*) para los modelos 6112 que se encuentra en el *url* anterior. En la Figura E-14 se muestra la pantalla de inicio de instalación del *software.*

Figura E-14 Pantalla de inicio de instalación del *software Developer kit* de las series 40 de *Nokia* para los modelos 6112

- Reiniciar el sistema para que los cambios tengan efecto

- Abrir los emuladores y en la opción *Help* elija la opción *Register now* para registrarlos

E.7. AGREGAR EL *PLUGIN* DE *leJOS* EN EL *IDE NETBEANS*

- Abrir el menú *tools -> plugins* que se encuentra en la parte superior del *IDE.* En la Figura E-15 se muestra la pantalla *Plugins*

- En la pestaña *Downloaded* hacemos *clic* en *add plugins* y buscamos el *path* donde se instalaron los proyectos de ejemplo para *leJOS* (*C:\Documents and Settings\KAT\leJOSNXJProjects\NXJPlugin\build*). Dentro de ese *path* hay una carpeta denominada *NXJPlugin* donde se encuentra el *plugin* para *Netbeans*

  En la Figura E-16 se muestra la pestaña *Downloaded* de la pantalla *Plugins*

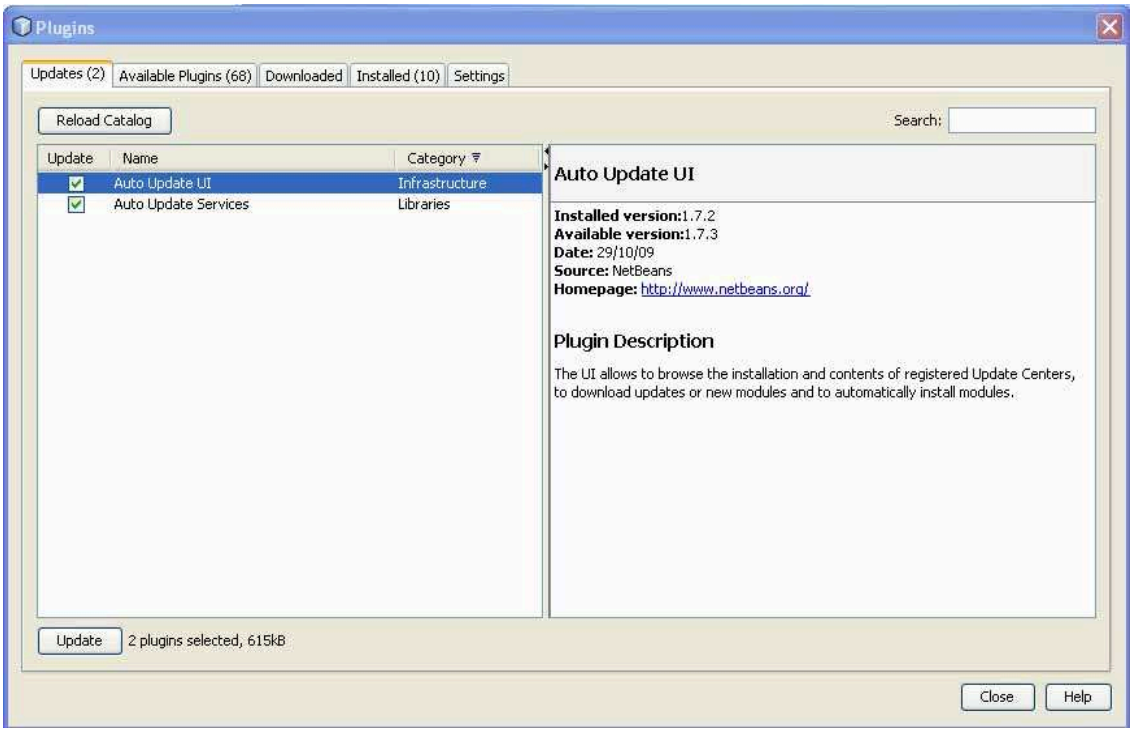


Figura E-15 Pantalla *Plugins* del *IDE Netbeans*

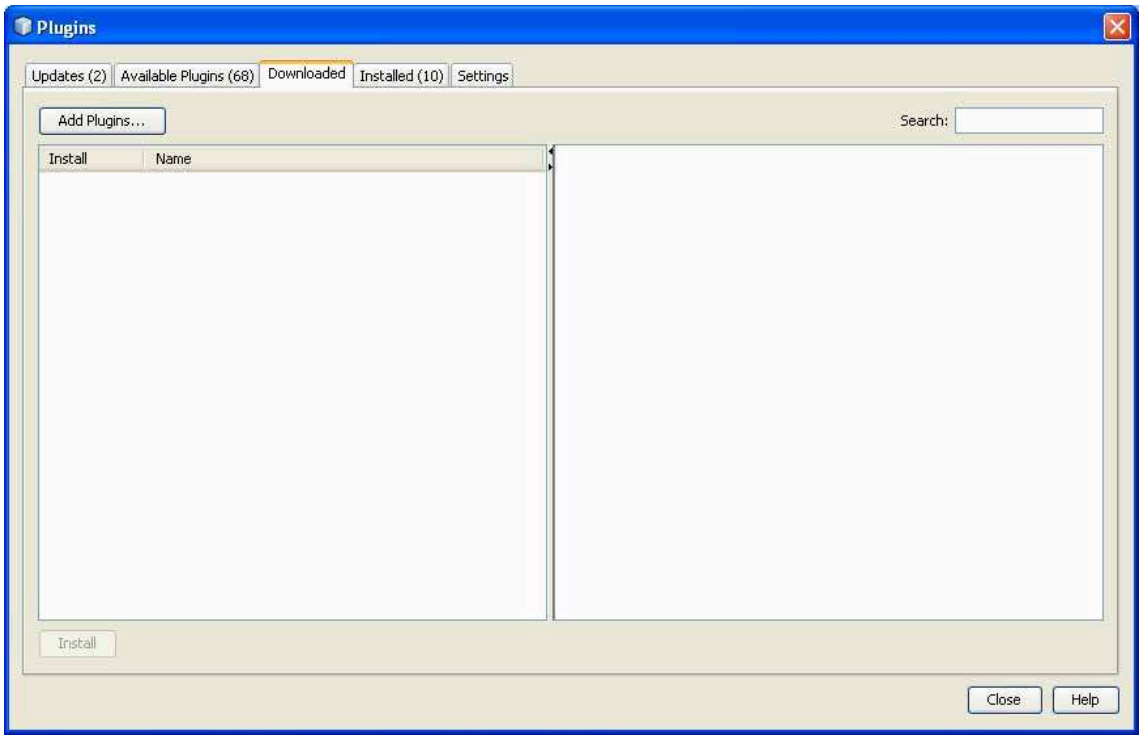


Figura E-16 Pestaña *Downloaded* de la pantalla *Plugins* del *IDE Netbeans*

- Si se realizaron correctamente los pasos anteriores se mostrara la pantalla de la Figura E-17

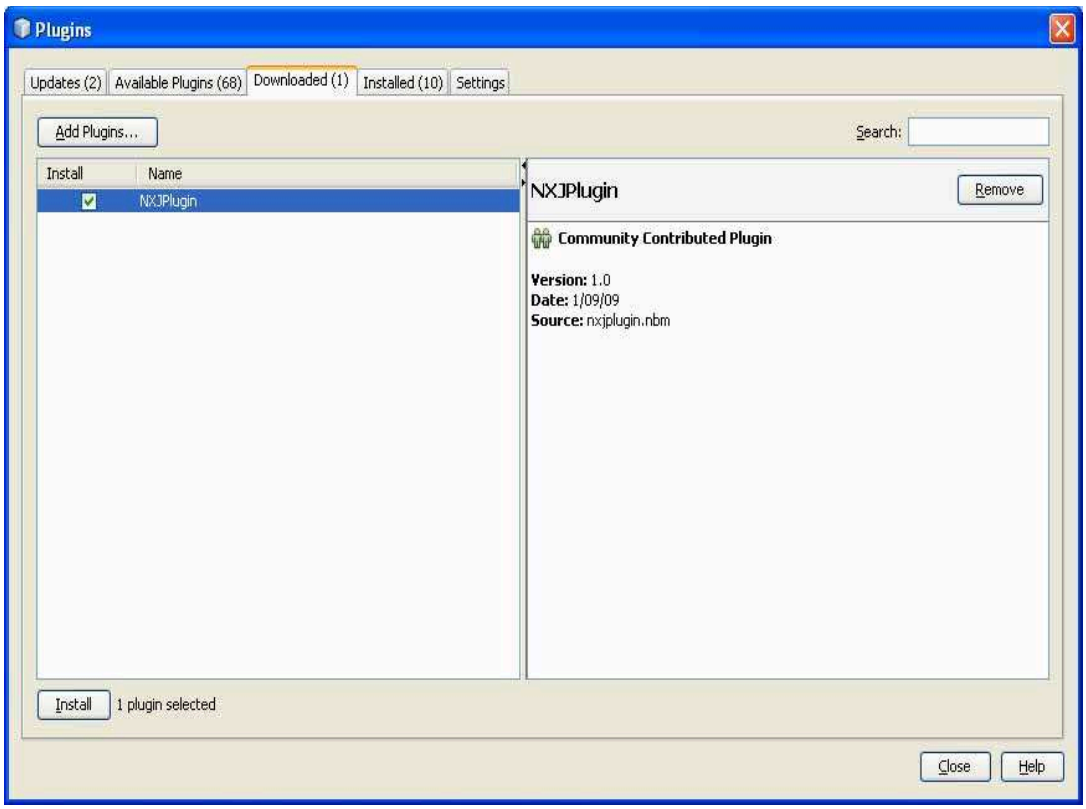


Figura E-17 *Plugin leJOS* instalado en el *IDE Netbeans*

E.8. AGREGAR LOS EMULADORES *NOKIA* EN EL *IDE NETBEANS*

- Para agregar los emuladores *Nokia* al *IDE Netbeans*, se debe primero crear un proyecto en blanco del tipo *Mobile Class Library J2ME* como se muestra en la Figura E-18.

- Inmediatamente *Netbeans* activara el modulo *J2ME* como se muestra en la Figura E-19.

- Una vez creado el proyecto dirigirse al menú superior a *Tools -> Java Platforms* como se muestra en la Figura E-20

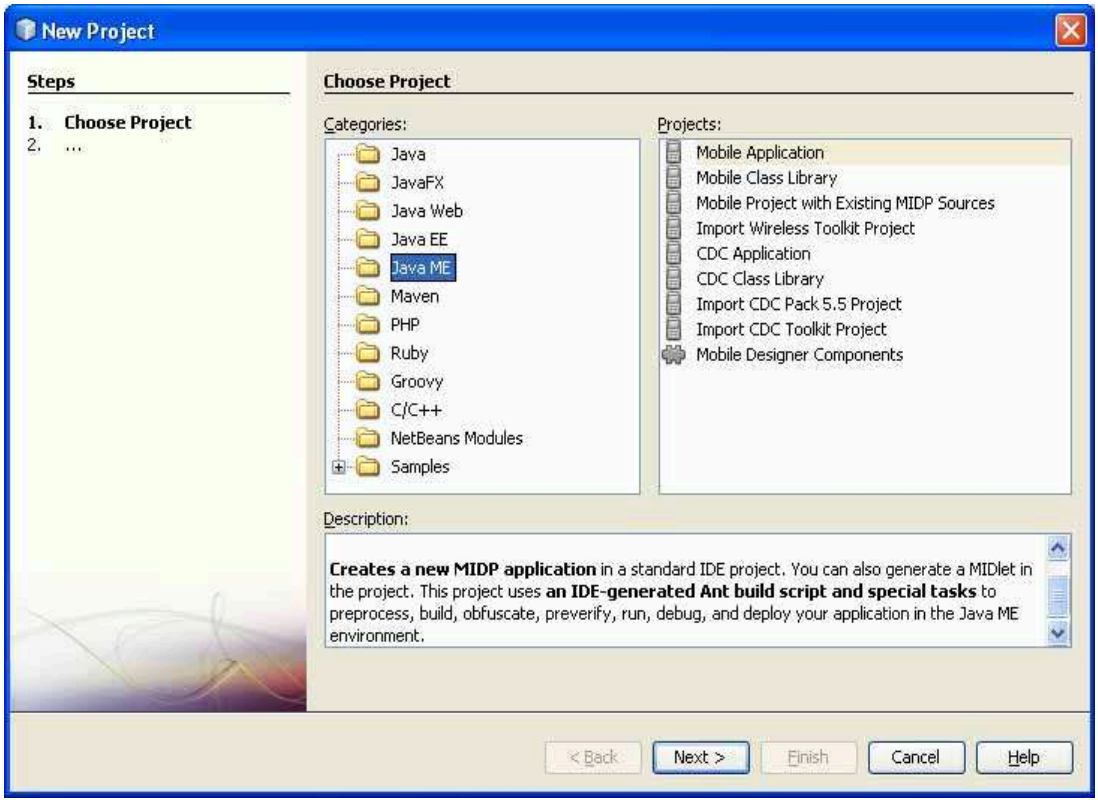


Figura E-18 Creación de un proyecto del tipo *Mobile Class Library J2ME* en el *IDE Netbeans*

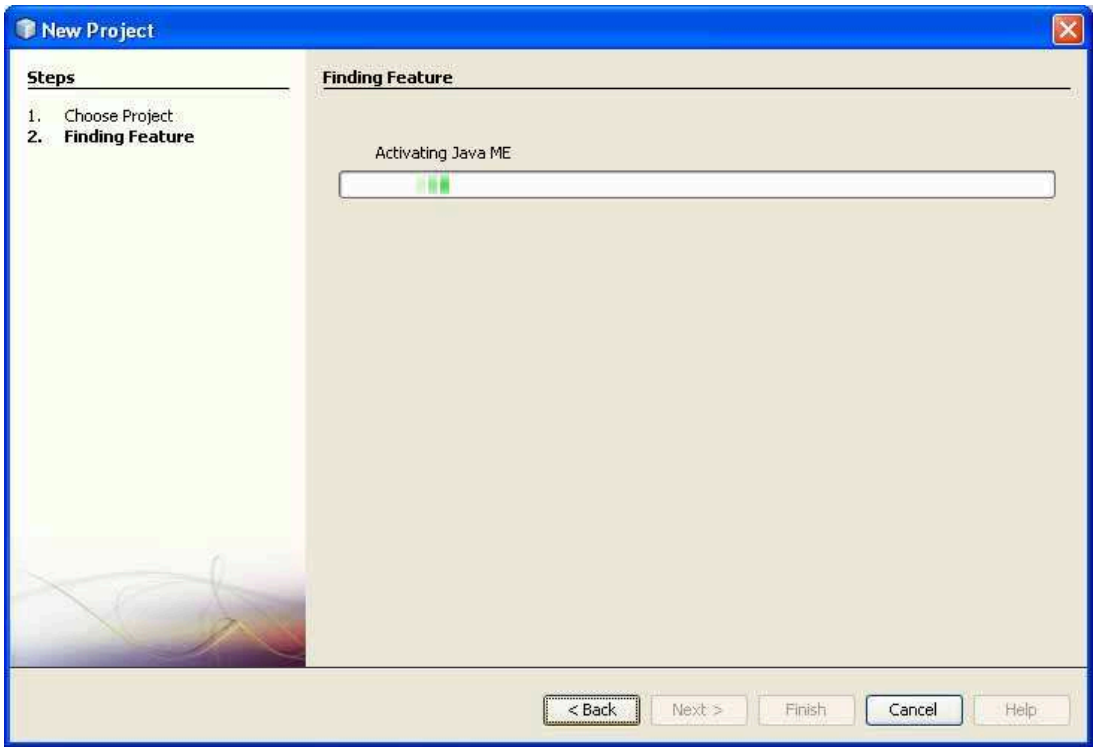


Figura E-19 Activación del modulo *J2ME* en el IDE Netbeans

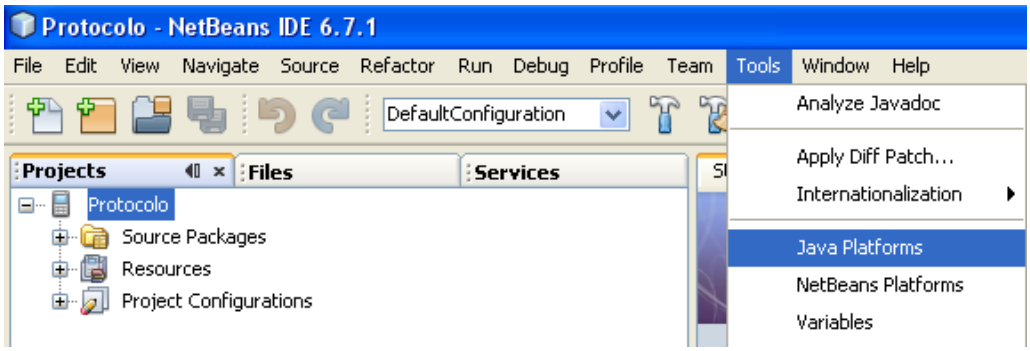


Figura E-20 Menú *Tools -> Java Platforms* del *IDE Netbeans*

- Se abrirá la pantalla *Java Platform Manager* que se muestra en la Figura E21. Se debe elegir *Add Platform* y se mostrará la pantalla de la Figura E-22 donde se debe escoger la opción *Java ME MIDP Platform Emulator.*

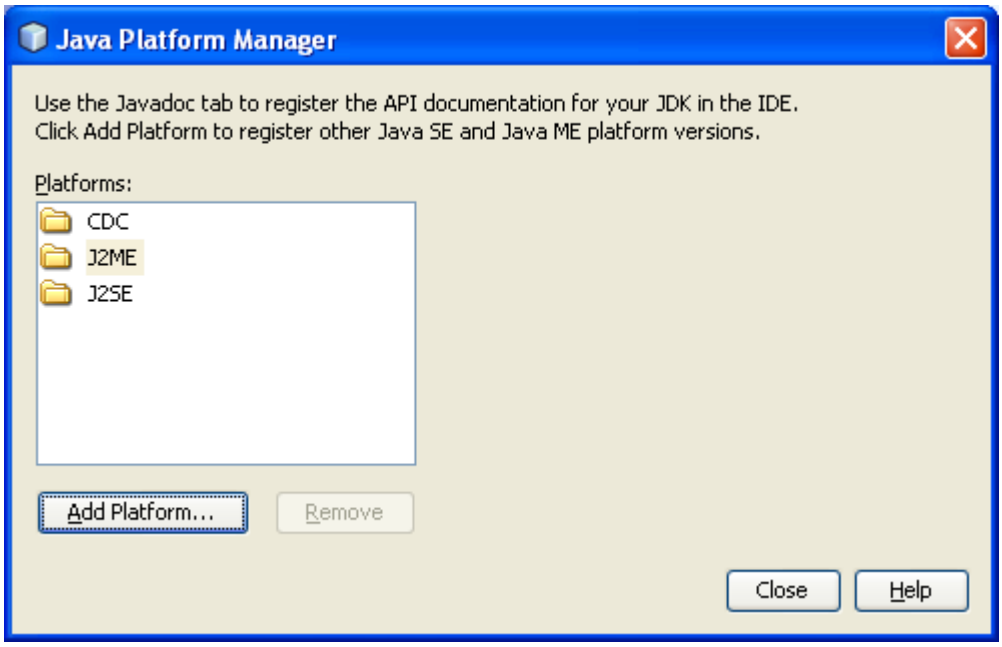


Figura E-21 Pantalla *Java Platform Manager* del *IDE Netbeans*

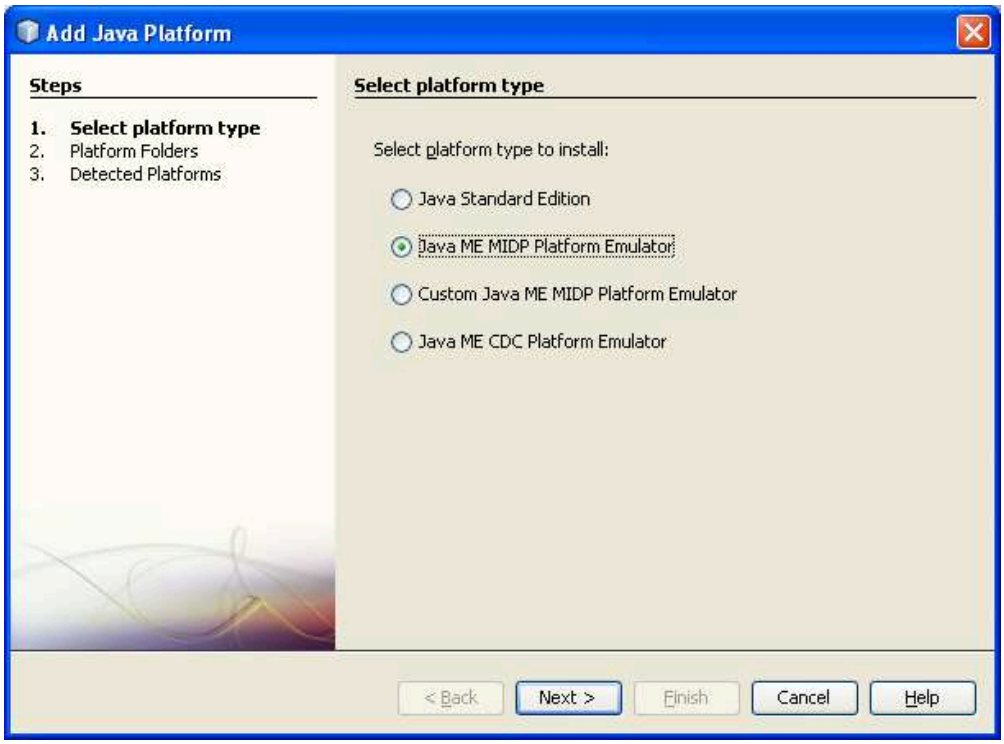


Figura E-22 Pantalla *Add Java Platform* del *IDE Netbeans*

- *Netbeans* realiza una búsqueda automática de las plataformas existentes en el *PC* y le muestra en pantalla como se ve en la Figura E-23.

Figura E-23 Plataformas existentes en el *PC*

E.9. CREACIÓN DE UN PROYECTO *leJOS* UTILIZANDO EL *IDE NETBEANS*

- Para crear un proyecto *leJOS* se debe abrir el menú *File -> New Project* que se encuentra en la parte superior del *IDE*. Se debe elegir la categoría *Samples,* tipo de proyecto *NXJproject* como se muestra en la Figura E-24.

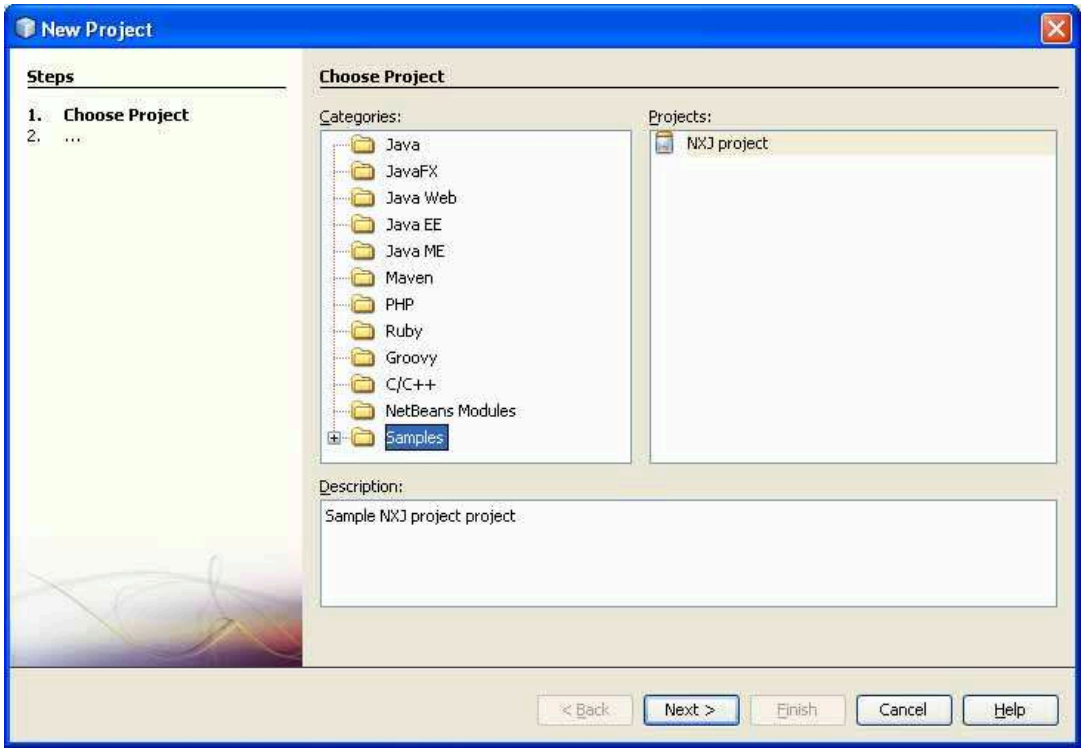


Figura E-24 Creación de un proyecto *leJOS* en el *IDE Netbeans*

- A continuación se pedirá el nombre que se desea dar al proyecto y se lo creará. En la Figura E-25 se muestra el proyecto creado.

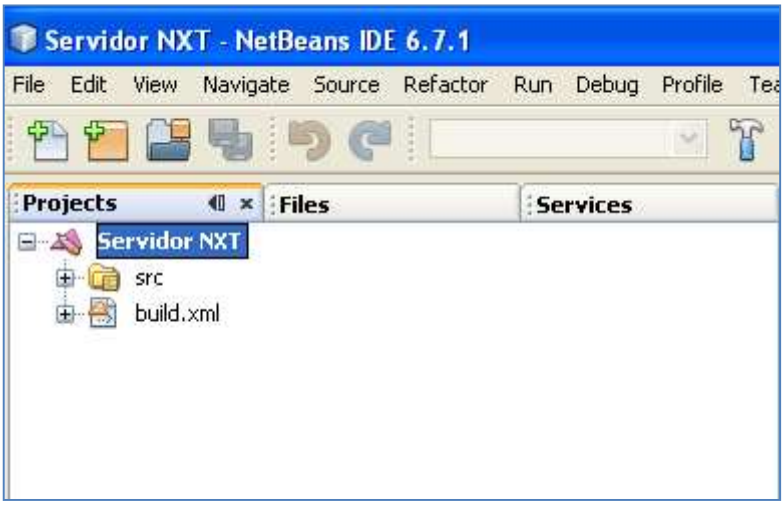


Figura E-25 Proyecto *leJOS* creado

- Una vez creado el proyecto, el *IDE Netbeans* genera los archivos que se muestran en la Figura E-26 en la carpeta del proyecto.

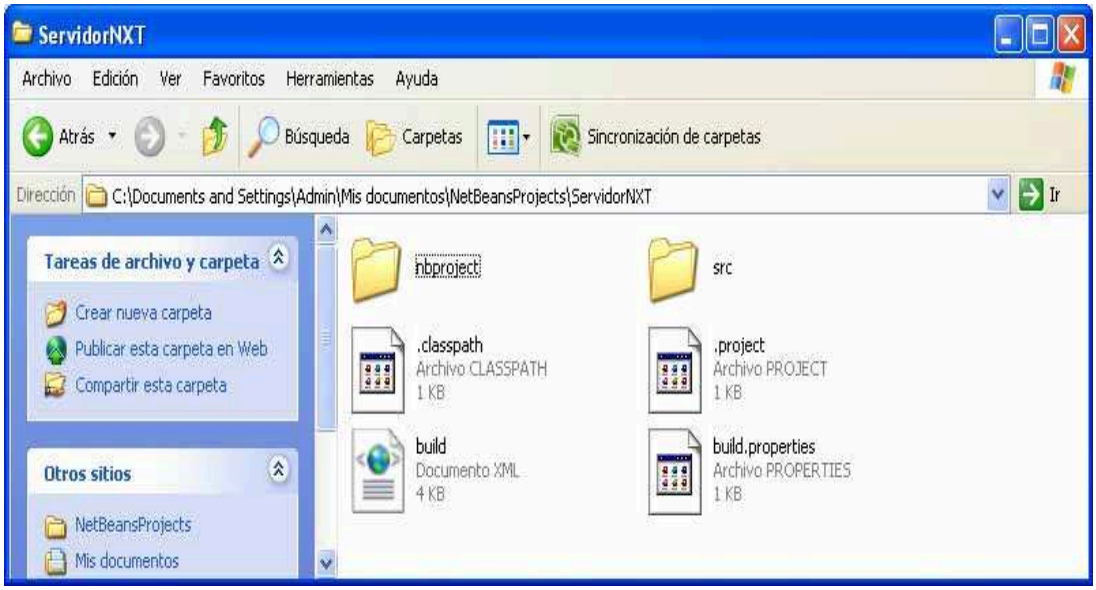


Figura E-26 Archivos del proyecto *leJOS*

- En el archivo *build.properties* se debe cambiar el valor de la variable *nxj.home=../snapshot* por *nxj.home=${env.NXJ_HOME}* quedando el archivo como se muestra en la Figura E-27. De esta manera el proyecto se compilará adecuadamente.

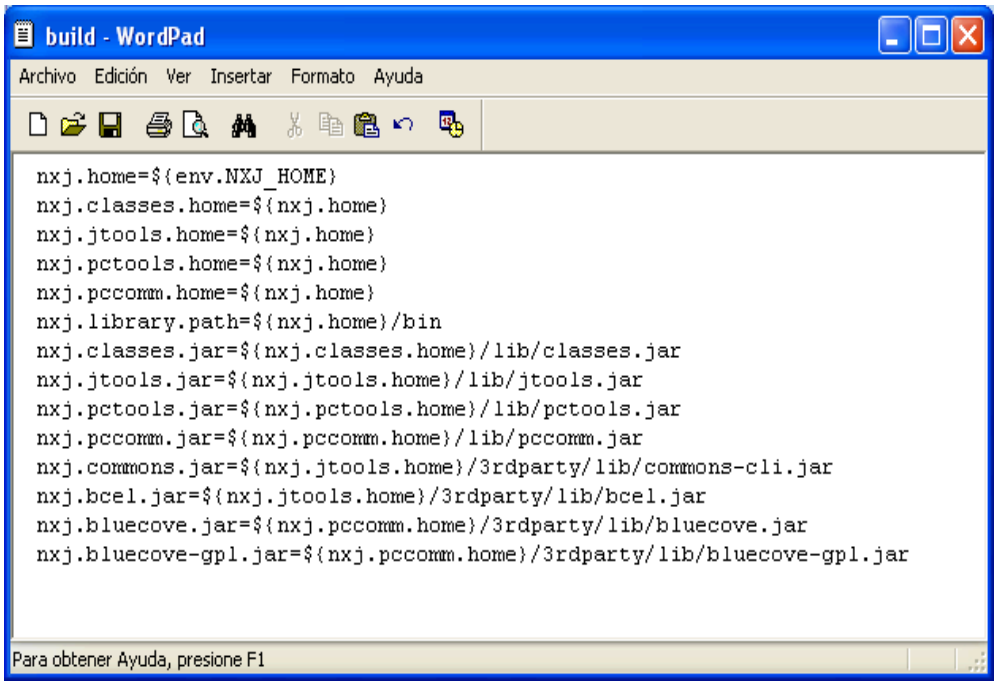


Figura E-27 Archivo *build.properties*

ANEXO F: MANUAL DE USUARIO

F.1. INSTALACIÓN DE LAS APLIACACIÓN EN LOS TELÉFONOS CELULARES Y EL BRAZO ROBOT *LEGO MINDSTORMS*

F.1.1. INSTALACIÓN DE LA APLICACIÓN EN EL TELEFONO CELULAR *SONY ERICSSON W580*

El instalador de la aplicación para los teléfonos celulares se encuentra en el archivo *Brazo_Robot.jar* que se muestra en la Figura F-1

Figura F-1 Instalador de la Aplicación para los teléfonos celulares

Para instalar la aplicación en el teléfono celular *Sony Ericsson w580* se conecta al *PC* ya sea a través de *USB* o *Bluetooth.* Se debe pasar el archivo *Brazo_Robot.jar* a la carpeta *Other* que se encuentra en el teléfono como se muestra en la Figura F-2. Una vez transferido el archivo se lo ejecuta desde el teléfono celular.

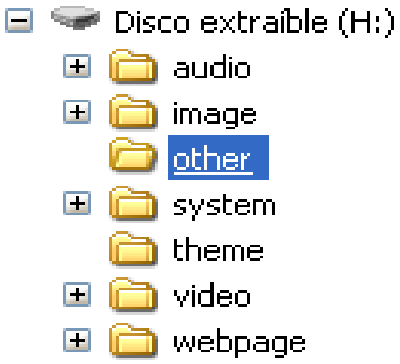


Figura F-2 Carpetas del teléfono *Sony Ericsson w580*

F.1.2. INSTALACIÓN DE LA APLICACIÓN EN EL TELEFONO CELULAR *NOKIA 5220*

Para instalar la aplicación en el teléfono celular *Nokia 5220*, se puede transferir directamente al archivo *Brazo_Robot.jar* desde el *IDE Netbeans* ya que tiene una conexión directa con el *software Nokia PC Suite*. Para esto se debe abrir el proyecto desde el *IDE Netbeans*, al hacer *clic* derecho sobre el nombre del proyecto se desplegarán varias opciones como se muestra en la Figura F-3, se escoge la opción *Properties* y se mostrará la pantalla de configuración como se muestra en la Figura F-4.

En la pantalla de configuración en la categoría *Deploying* se elige como *Deployment Method* el *Nokia Terminal connected via PC Suite* como se muestra en la Figura F-4.

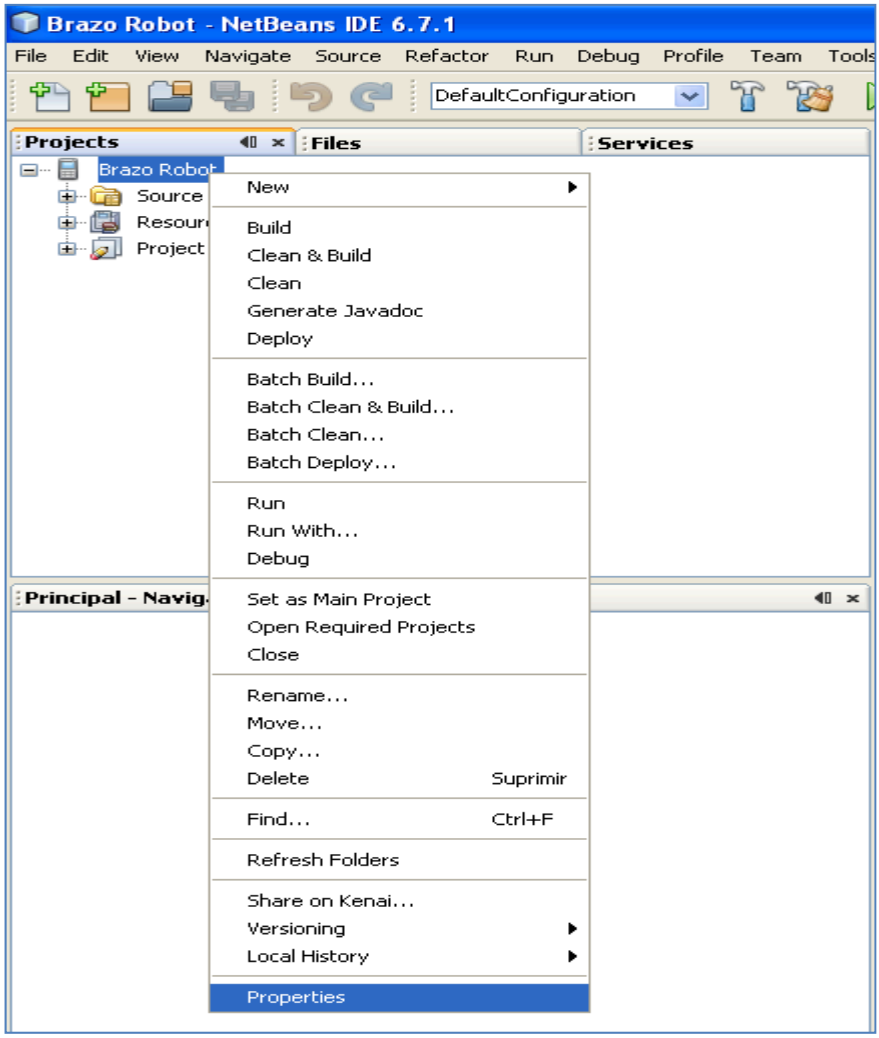


Figura F-3 Opciones del proyecto creado en el *IDE Netbeans*

En el momento que se desea transferir la aplicación desde el *IDE Netbeans*, una vez que se ha conectado el teléfono celular con el *PC,* ya sea a través de *USB* o *Bluetooth*, en las opciones del proyecto que se muestran en la Figura F-3 se escoge *Deploy*. La aplicación será transferida directamente al teléfono celular.

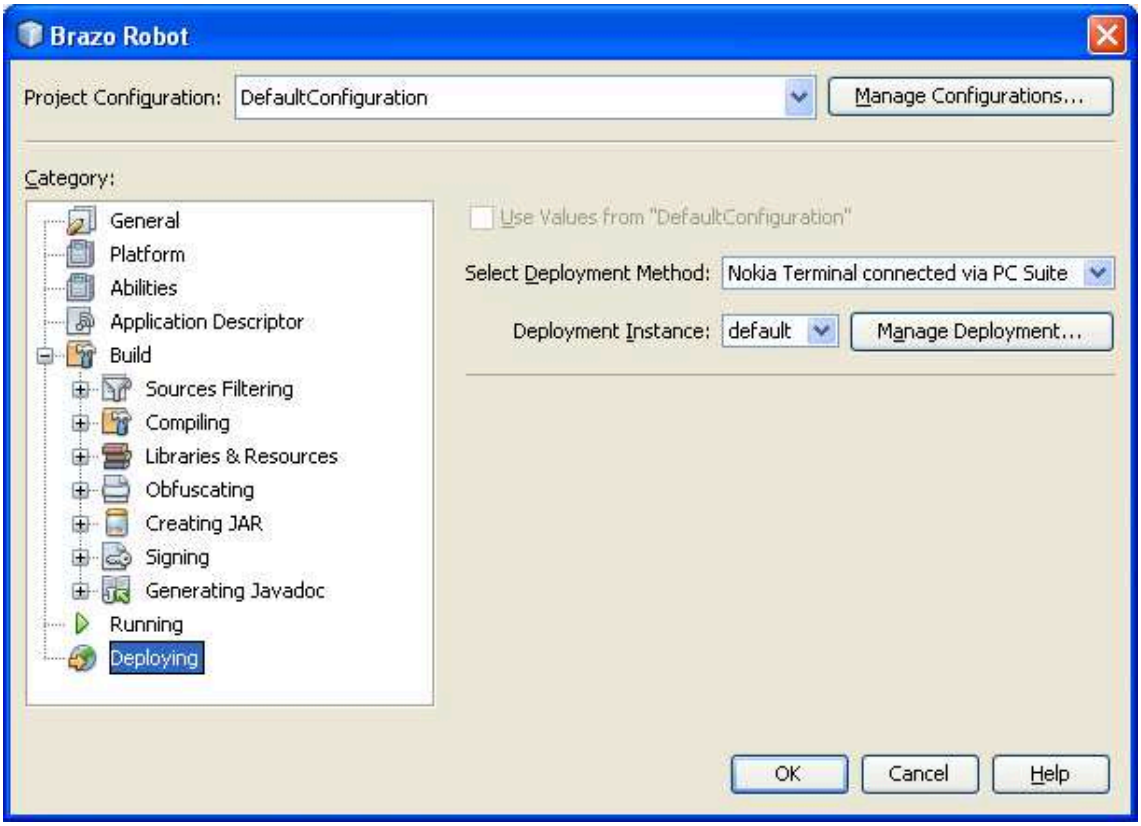


Figura F-4 Pantalla de configuración del las propiedades del proyecto creado en el *IDE Netbeans*

F.1.3. INSTALACIÓN DE LA APLICACIÓN EN EL BRAZO ROBOT *LEGO MINDSTORMS*

Para instalar la aplicación en el brazo robot se lo hace mediante el *IDE Netbeans*. Se debe abrir el proyecto mediante la opción *File -> Open Proyect* como se muestra en la Figura F-5.

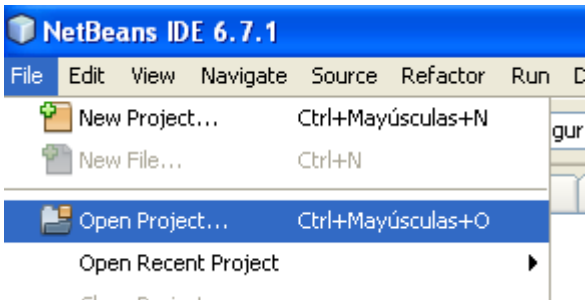


Figura F-5 Opción para abrir un proyecto desde el *IDE Netbeans*

A continuación se abrirá una pantalla que permitirá buscar la ubicación del proyecto que se desea abrir, en nuestro caso el proyecto que se debe abrir es el proyecto *ServidorNXT* como se muestra en la Figura F-6.

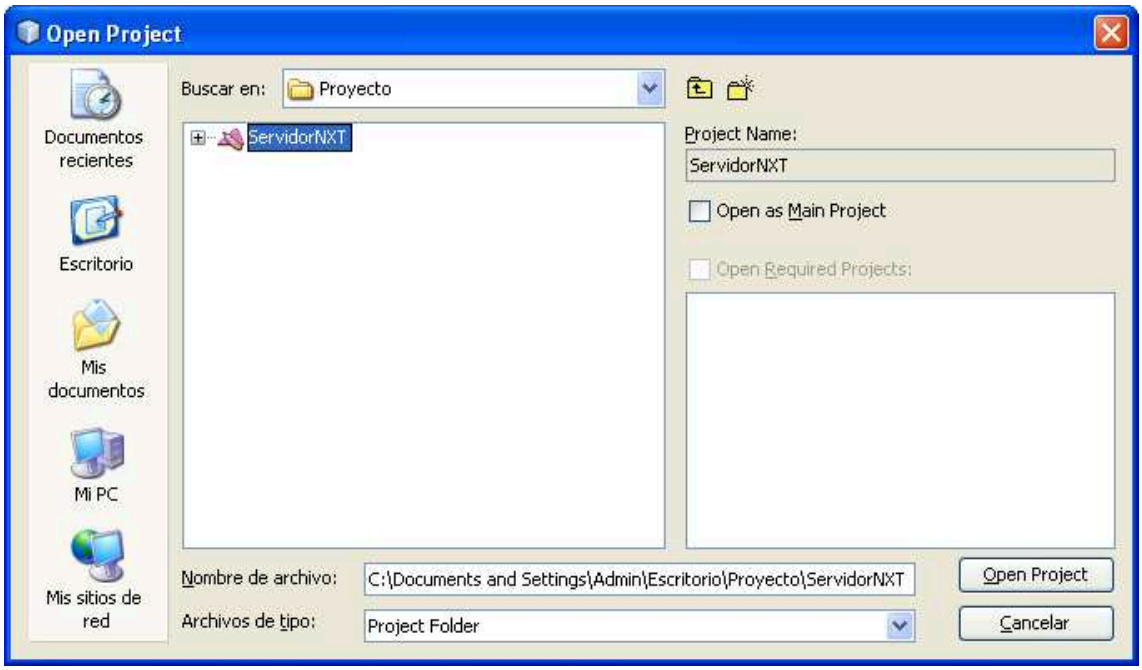


Figura F-6 Selección del proyecto que se desea abrir en el *IDE Netbeans*

Una vez abierto el proyecto se debe conectar el Brazo Robot al *PC* mediante *Bluetooth*. Para transferir el proyecto se debe hacer *clic* sobre el icono  (*Run Project*) que se encuentra en la parte superior del *IDE,* o se puede hacer clic derecho sobre el nombre del proyecto y elegir la opción *Run* como se muestra en la Figura F-7.

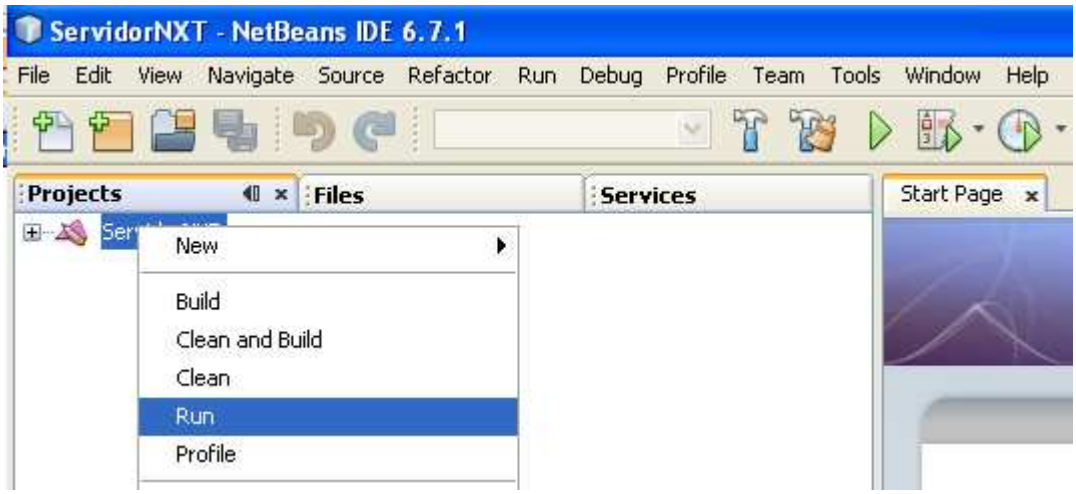


Figura F-7 Opción para transferir un proyecto desde el *IDE Netbeans* al Brazo Robot

F.2. USO DE LA APLICACIÓN

Antes de iniciar la aplicación se debe conocer la función que realiza cada botón del brazo robot. En la Figura F-8 se muestra la función de cada botón del *NXT brick*.

Figura F-8 Botones del *NXT Brick*

El primer paso consiste en iniciar el programa servidor que se encuentra instalado en el brazo robot.

Para encender el brazo robot se debe presionar el botón *ENTER*. Inmediatamente aparecerá en pantalla el menú que corresponde al sistema operativo *leJOS*, como se muestra en la Figura F-9.

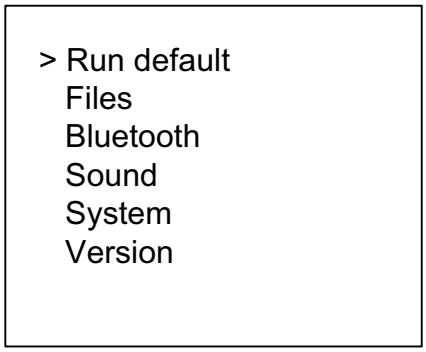


Figura F-9 Menú del sistema operativo *leJOS*

Luego se debe seleccionar *Files* > *ServidorNXT.nxj* como se muestra en la Figura F-10. Automáticamente aparecerá la pantalla *Presentación* con la caratula del proyecto como se muestra en la Figura F-11. Para continuar se debe presionar *ENTER*.

Figura F-10 Selección del archivo

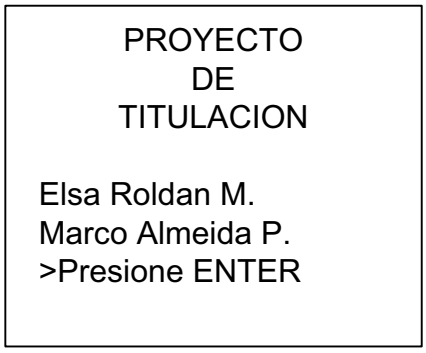


Figura F-11 Pantalla *Presentación* del Brazo Robot

En la Pantalla *MENU* que se muestra en la Figura F-12 existen dos opciones: *Servidor* y *Salir*. Si se elige la opción *Salir* la aplicación retornará al menú del sistema operativo, caso contrario si se elige la opción *Servidor*, entonces el *NXT Brick* estará preparado recibir una conexión entrante y se mostrará la pantalla *Espera Conexión* como se muestra en la Figura F-13.

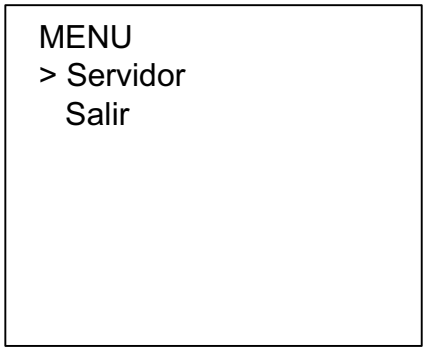


Figura F-12 Pantalla *MENU* en el Brazo Robot.

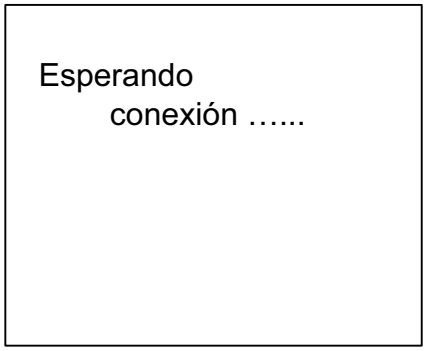


Figura F-13 Pantalla *Espera Conexión* en el Brazo Robot

Una vez que el brazo robot se encuentre esperando conexiones entrantes, se procede a iniciar la aplicación cliente que se encuentra instalada en los teléfonos celulares. Para este manual de usuario se utilizará el teléfono celular *Nokia* 5220.

La aplicación del cliente se encuentra en el *path Menú* > *Aplicaciones* > *Colección*. La aplicación se denomina *Brazo Robot*.

Cuando se inicia la aplicación se despliega la pantalla *Presentación* del teléfono celular como se indica en la Figura F-14.

Figura F-14 Pantalla Presentación del teléfono celular

En la pantalla Presentación del teléfono celular se tiene dos botones *Options* e Ingresar.

Al elegir el botón *Options* se desplegará otros dos botones: *Salir* e *Info* como se muestra en la Figura F-15. El botón *Salir* cierra la aplicación y el botón *Info* da una breve descripción de la aplicación como se muestra en la Figura F-16.

Figura F-15 Botón *Options* de la pantalla *Presentación*

Figura F-16 Botón *Info* de la pantalla *Presentación*

Si se elige el botón *Ingresar*, se despliega la pantalla *Búsqueda* como se muestra en la Figura F-17. Esta pantalla tiene por objetivo permitir al usuario ingresar el nombre *Bluetooth* del brazo robot para iniciar la búsqueda del mismo. La búsqueda se inicia presionando el botón *Buscar*.

Figura F-17 Pantalla *Búsqueda*

Mientras el teléfono celular realiza el proceso de búsqueda se despliega la pantalla *Espera* como se muestra en la Figura F-18.

Figura F-18 Pantalla *Espera*

Si el teléfono celular encontró el brazo robot, establece la conexión y se desplegará la pantalla *Control Robot* como se muestra en la Figura F-19. En el brazo robot se despliega la pantalla *Información conexión* como se muestra en la Figura F-20.

Figura F-19 Pantalla Control Robot.

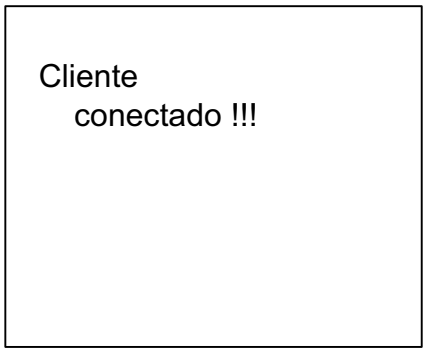


Figura F-20 Pantalla *Información conexión* en el Brazo Robot

La pantalla *Control Robot* permite al usuario manipular los movimientos del robot. Por ejemplo, si el usuario elige la opción izquierda el brazo robot se moverá de manera continua hacia la izquierda. Para parar el movimiento que esté ejecutando el usuario deberá ubicar el cursor en el centro.

En esta pantalla se tiene dos botones *Options* y *Fin*. Al elegir el botón *Options* se desplegarán otros dos botones: *Log* y *Config* como se muestra en la Figura F-21.

Figura F-21 Botón *Options* de la pantalla *Control Brazo Robot*

Si se elige el botón *Log* se muestra un historial de todos los comandos que se han transmitido y los respectivos acuses de recibo tanto en formato binario como en decimal. Esta pantalla se muestra en la Figura F-22

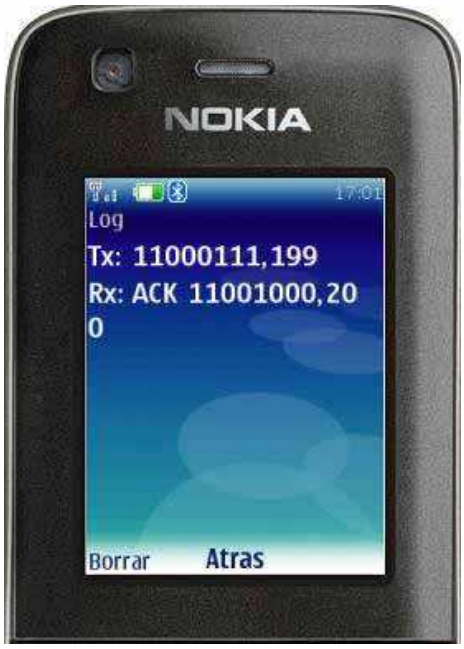


Figura F-22 Pantalla *Log* del teléfono celular

De igual manera en el brazo robot se mostrará la pantalla *Log* con la misma información como se muestra en la Figura F-23.

Rx (int):
199
Rx (bin):
11000111
ACK Tx (int):
200
ACK Tx (bin):
11001000

Figura F-23 Pantalla Log del Brazo Robot

Si se elige el botón *Config* de la pantalla *Control Brazo Robot* se desplegará la pantalla *Configuración Color* como se muestra en la Figura F-24. Esta pantalla permite al usuario indicarle al brazo robot que color deberá tener la pelota para que la capture.

Figura F-24 Pantalla Configuración Color

En esta pantalla se tiene dos botones: *Options* y *Select*. El botón *Select* permite escoger entre las dos opciones color rojo o color azul. Una vez que se ha escogido el color se debe proceder a presionar el botón *Options,* el mismo que mostrará dos botones más: *Enviar* y *Atrás* como se muestra en la Figura F-25. Si se presiona el botón *Enviar* se guardará la configuración en el brazo robot, caso contrario si se presiona el botón *Atrás* la configuración no se guardara. El botón *Atrás* permite regresar a la pantalla *Control Robot*.

Figura F-25 Botones de la pantalla Configuración Color

El botón *Fin* de la pantalla *Control Robot* por otra parte permite finalizar la conexión y cerrar la aplicación como se muestra en la Figura F-26. En el brazo robot aparecerá el mensaje *Conexión Finalizada* como se muestra en la Figura F-27.

Figura F-26 Pantalla Fin de la conexión en el teléfono celular.

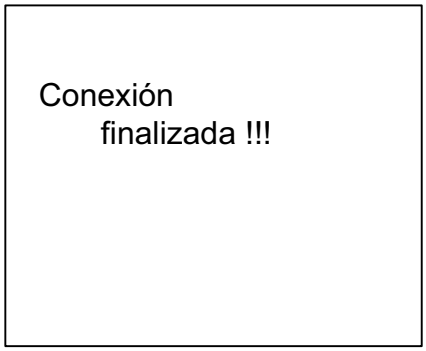


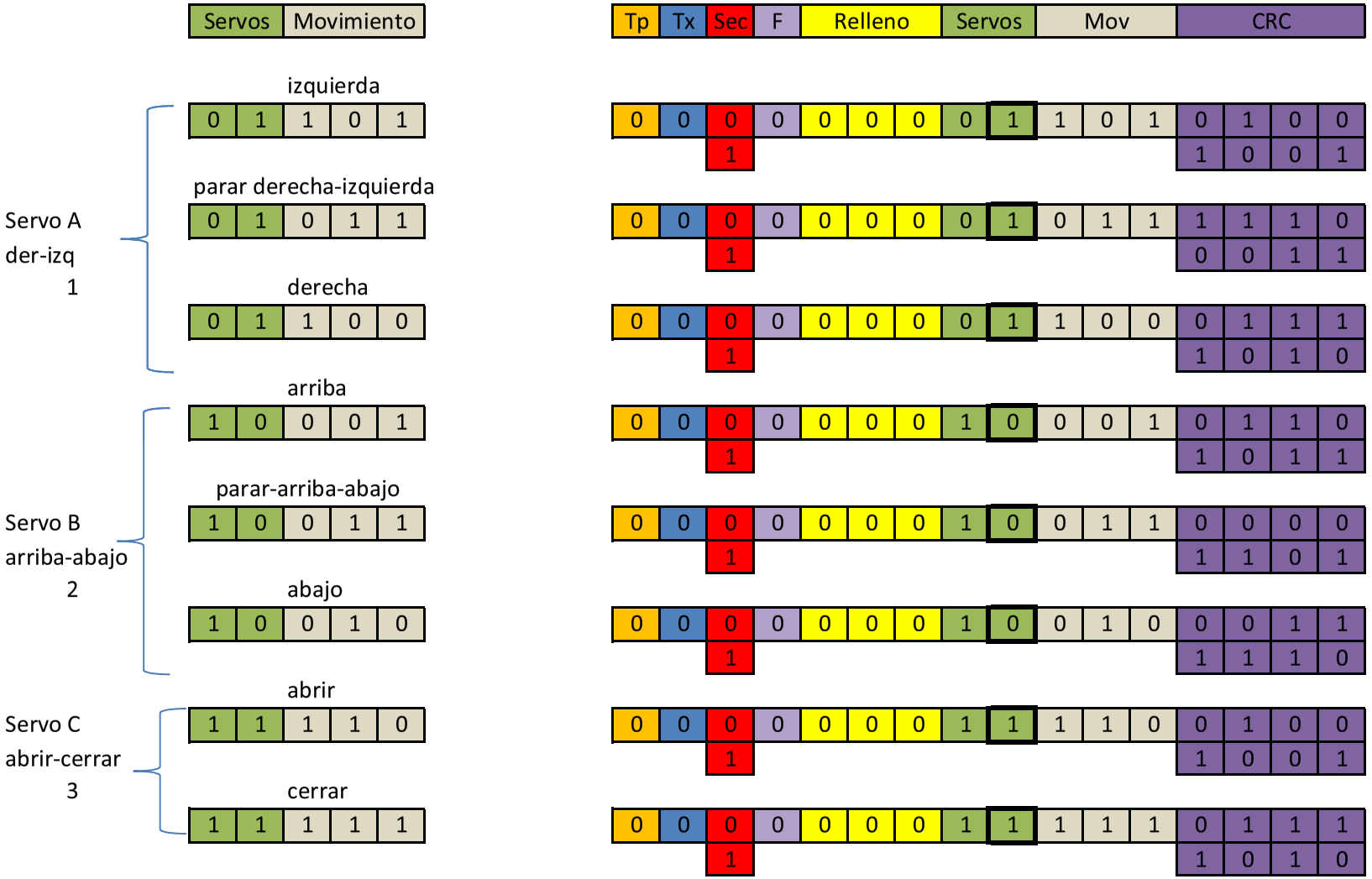
Figura F-27 Pantalla Fin de la conexión en el Brazo Robot

.

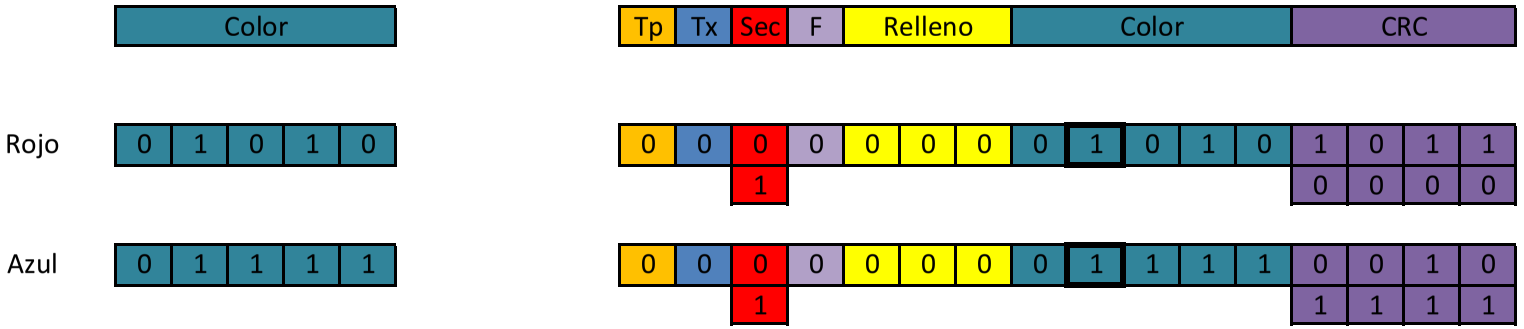
ANEXO G: MENSAJES DEL PROTOCOLO DE COMUNICACIÓN

G.1 MENSAJES DEL TELEFONO CELULAR

G.1.1 MENSAJES DE EJECUCIÓN DE COMANDO



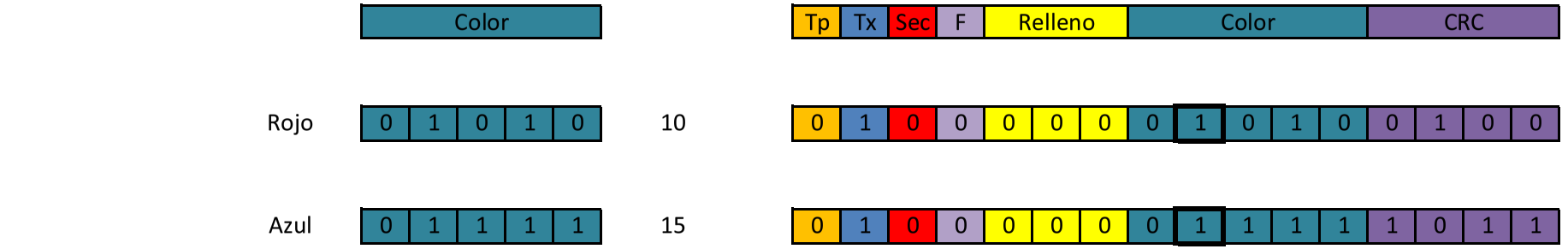
G.1.2 MENSAJES DE CONFIGURACIÓN COLOR



G.1.3 MENSAJES DE FIN DE CONEXIÓN

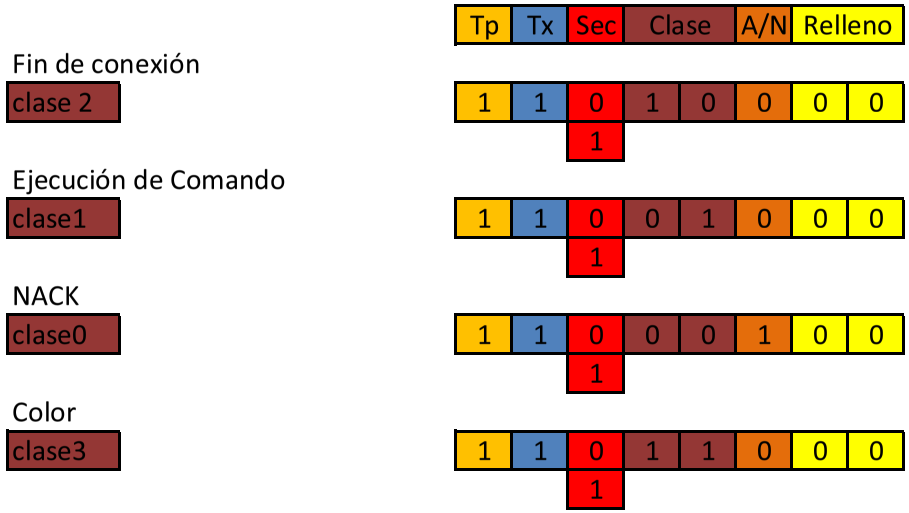
G.2 MENSAJES DEL BRAZO ROBOT

G.2.1 MENSAJES DE DETECCIÓN DE COLOR



G.2.2 MENSAJE DE FIN DE CONEXIÓN

G.2.3 MENSAJES DE CONTROL



ANEXO H
TABLA DE SALARIOS MÍNIMOS SECTORIALES: COMISIÓN SECTORIAL-TECNOLOGÍA

Comisión Sectorial No. 12
TECNOLOGÍA HARDWARE Y SOFTWARE (INCLUYE TIC´s) [72]

RAMA DE ACTIVIDAD	ESTRUCTURA OCUPACIONAL	CARGO Y/O FUNCIÓN	SALARIO MÍNIMO SECTORIAL
ENSAMBLAJE Y MANTENIMIENTO DE EQUIPOS DE COMPUTACIÓN Y ELECTRÓNICOS	B2	INGENIERO ELECTRÓNICO ESPECIALISTA EN MANTENIMIENTO	303,60
	D2	TÉCNICO EN ENSAMBLAJE Y MANTENIMIENTO DE EQUIPOS DE COMPUTACIÓN Y ELECTRÓNICOS	287,76
		TÉCNICO DE HELP DESK	
TÉCNICOS EN TELECOMUNICACIONES	A1	DIRECTOR DE TELECOMUNICACIONES O JEFE DE AREA.	337,36
	B1	SUPERVISOR GENERAL DE TELECOMUNICACIONES	324,83
		JEFE DE SISTEMAS, DESARROLLO Y TECNOLOGÍA	
		ARQUITECTO DE SOFTWARE	
	B2	DISEÑADOR DE SOFTWARE	303,60
		ADMINISTRADOR DE BASE DE DATOS	
	B3	ESPECIALISTA DE TELECOMUNICACIONES	300,96
		SUPERVISOR DE PLATAFORMAS O EQUIPO DE VOZ	
		SUPERVISOR DE PLATAFORMAS O EQUIPOS DE DATOS Y TELECOMUNICACIONES	
		ESPECIALISTA PROYECTISTA	
		TECNICO OPERADOR DE RADAR	
		TÉCNICO DE REDES DE DATOS	
	C1	PROGRAMADOR SENIOR	299,64
		ANÁLISTA DE SOFTWARE	
		TESTER DE SOFTWARE	
		PROGRAMADOR Y DISEÑADOR MULTIMEDIA	
		TÉCNICO EN MANTENIMIENTO DE SERVIDORES	
		OPERADOR DE CALL CENTER BILINGÜE	
		TECNICO INSTALADOR DE SERVICIOS AGREGADOS	
		OPERADOR DE LOCUTORIO, CYBER O CENTRO DE LLAMADAS	
		TÉCNICO DE FIBRA ÓPTICA Y EMPALMADO	
	C2	TÉCNICO EN MANTENIMIENTO DE COMPUTADORAS	294,36
		OPERADOR DE COMPUTACION	
		TECNICO DE CENTRALES TELEFONICAS	
		TECNICO DE TRANSMISIONES	

Comisión Sectorial No. 12
TECNOLOGÍA HARDWARE Y SOFTWARE (INCLUYE TIC´s) [72]

RAMA DE ACTIVIDAD	ESTRUCTURA OCUPACIONAL	CARGO Y/O FUNCIÓN	SALARIO MÍNIMO SECTORIAL
TÉCNICOS EN TELECOMUNICACIONES	C2	TECNICOS CONETORIZADORES DE EQUIPO	294,36
		AYUDANTE DE INSTALACION	
	D1	GUARDA ALMACEN O BODEGUERO	291,72
	D2	PROGRAMADOR JUNIOR	287,76
		TECNICO DE PLANTA EXTERNA, CABLISTA Y/O INSTALADOR	
		ASISTENTE DE TELECOMUNICACIONES	
	E1	MENSAJERO	287,68

Se debe considerar las siguientes disposiciones legales:

1. El Art. 3 del Acuerdo Ministerial 255 publicado en el suplemento del Registro Oficial No.358 del 8 de enero del 2011, señala: “En caso de que en las estructuras ocupacionales de las Comisiones Sectoriales, en una o varias ramas de actividad no se encuentren contemplados los cargos y/o funciones, estos deberán aplicar el salario básico unificado, debiendo el respectivo empleador notificar al Ministerio de Relaciones Laborales el o los cargos no contemplados, hasta el 31 de julio de cada año, a efectos de ser previstos en las reuniones de las comisiones sectoriales del año 2011”

2. Los salarios mínimos sectoriales determinados en el Acuerdo Ministerial 255 corresponden a los valores mínimos que el empleador debe pagar al trabajador conforme lo determina el Art. 81 del Código del Trabajo, independientemente de que el empleador pudiere pagar un valor mayor y/o valores que se hubieran negociado en contratos colectivos.