INSTITUTO POLITÉCNICO NACIONAL
CENTRO DE INVESTIGACIÓN EN COMPUTACIÓN
No. 1 **Serie:** Roja **Fecha:** Mayo 2007

Implementación en Java del Framework del Estándar de Hardware y Software

Daniel Aaron Torrescano Arias[1]
Juan Luis Díaz de León Santiago[2]

RESUMEN

En este documento se presentan el desarrollo e implementación completos del framework y su extensión de acuerdo a lo estipulado en el Estándar de Hardware y Software, para el Control Modular de Prototipos Industriales[2] utilizando el lenguaje de programación Java de Sun Microsystems y cuya funcionalidad es la de generar aplicaciones de alto nivel que involucran hardware y software.

Palabras clave: Software, Framework, Módulo, Estándar, Comunicación Serial, Protocolo, Pruebas

[1]Estudiante del Centro de Investigación en Computación.
[2]Profesor Titular del Centro de Investigación en Computación.

ADVERTENCIA

"Este reporte contiene información desarrollada por el Centro de Investigación en Computación del Instituto Politécnico Nacional a partir de datos y documentos con derechos de propiedad y por lo tanto su uso queda restringido a las aplicaciones que explícitamente se convenga.

La aplicación no convenida exime al Centro de su responsabilidad técnica y da lugar a las consecuencias legales que para tal efecto se determinen.

Información adicional sobre este reporte podrá obtenerse recurriendo a la Unidad de Publicaciones y Reportes Técnicos del Centro de Investigación en Computación del I.P.N. Av. Juan de Dios Bátiz s/n, teléfono 5729-63-00 ext. 56403, 56608 y 56610"

AGRADECIMIENTOS

Agradezco a mis compañeros del grupo de trabajo con los cuales he aprendido más de lo que alguna vez imaginé, a mi familia por el gran apoyo que siempre me han brindado, a mi asesor por dar rienda suelta a mi imaginación y al I.P.N. por otorgarme una casa que apreciar, respetar y querer. Este trabajo es apoyado y financiado por el Consejo Nacional de Ciencia y Tecnología (CONACYT) de México bajo registro 191878.

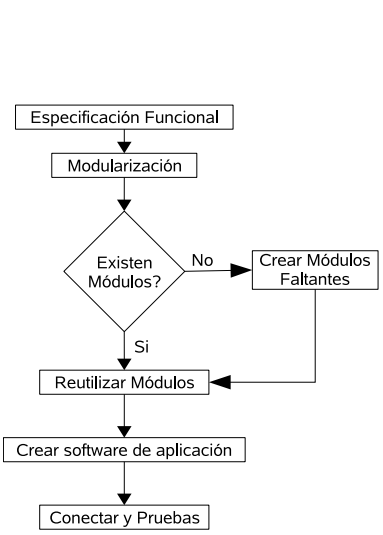
Índice

1. Introducción

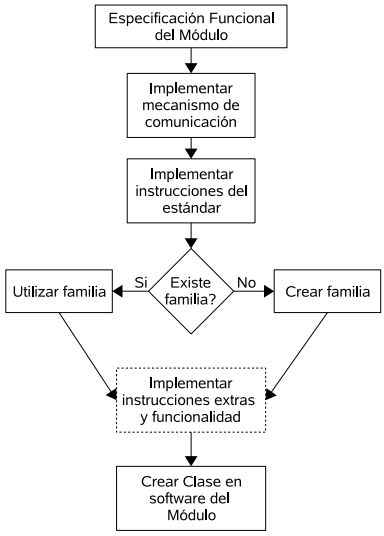
En [2] se presenta un Estándar de Hardware y Software para el Control Modular de Prototipos Industriales con el cual es posible modularizar funcionalmente un problema de hardware y software conjunto. Para desarrollar las aplicaciones de alto nivel son necesarios: los módulos (hardware y software) y un framework, es decir, un conjunto de clases abstractas y concretas que proporcionan el marco de trabajo y utilerías necesarias para la extensión del mismo o para el desarrollo de aplicaciones. En este trabajo se presentan el diseño y la implementación de dicho framework, así como una extensión que provee funcionalidad real al framework. Este trabajo fue realizado en LaTeX 2_ε y con el paquete informestecnicoscic [1].

2. Antecedentes

Existen distintos modelos de creación de prototipos, algunos de ellos mostrados por Rosentiel et al. en [6], sin embargo, el modelo que se implementa en este módulo es el propuesto en [2], que consta de un mecanismo general (figura 1b) y un submecanismo (figura 1a).



(a) Metodología para la creación de prototipos.



(b) Sub-Mecanismo de creación de Módulos.

Figura 1: Modelo de creación de prototipos.

En comparación con las metodologías de creación rápida de prototipos expuestas por Rosentiel et al. en [6], esta metodología permite que el diseñador pase de la especificación funcional y la modularización a las pruebas en un tiempo muy corto, de hecho, es posible aplicar cada avance del sistema integrado con el hardware físicamente presente. Sin embargo, es necesario contar con los módulos definidos por dicha modularización. Los retrasos en el

desarrollo pueden no ser necesarios en la medida en que la disponibilidad de módulos creados sea mayor.

El paradigma de programación orientado a objetos proporciona mecanismos muy importantes para la realización de sistemas de cómputo, como lo son:

Herencia Mecanismo mediante el cual unos elementos más específicos incorporan la estructura y el comportamiento definidos por otros elementos más generales.

Polimorfismo Indica una operación cuya implementación (un método o una máquina de estados disparada por un evento de llamada) puede ser proporcionada por una clase descendiente.

El uso común de lenguajes de diseño hizo necesario el consenso de diagramas en una herramienta popularmente utilizada para el diseño orientado a objetos. En [5] se establecen algunas realciones importantes entre las clases como son:

Asociación Es una descripción de una conexión entre instancias de clases.

Composición Es un tipo de asociación que representa una relación todo-parte

Agregación Es una forma más fuerte de asociación, en la cual el compuesto es el responsable único de gestionar sus partes

Dependencia Es una relación entre dos elementos de un modelo

Generalización Una relación entre una descripción más general y una variedad más específica de la general, usada para herencia.

Algunos de los diagramas utilizados en UML se presentan en la figura 2.

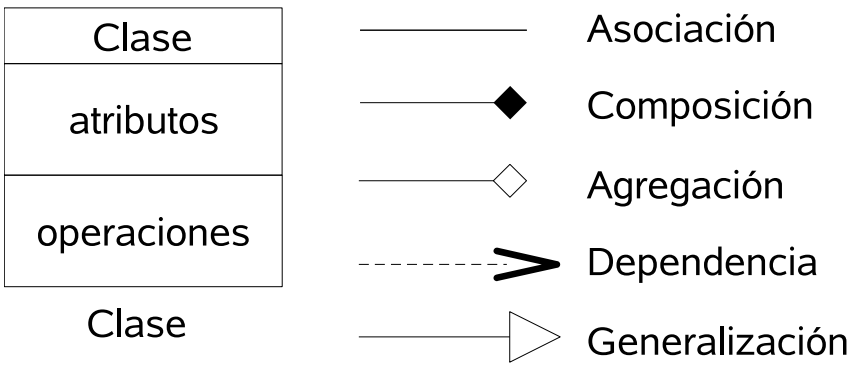


Figura 2: Diagramas en UML.

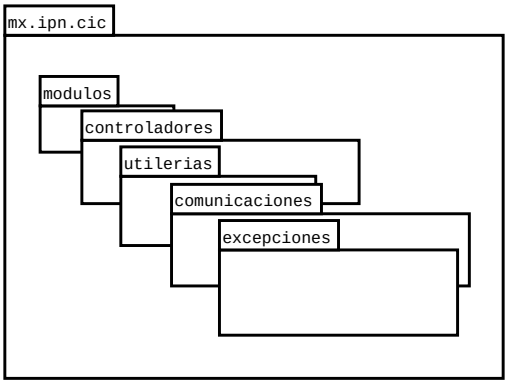


Figura 3: Paquetes del framework estándar.

3. Diseño del Framework

El framework esta dividido en cinco paquetes mostrados en la figura 3. Dicha división cumple con la separación de áreas dentro del estándar, la descripción de cada paquete es mostrada en la tabla 1.

Cuadro 1: Descripción de los paquetes del framework.

Paquete	Descripción
mx.ipn.cic.modulos	Dentro de este paquete se encuentra la clase base de cualquier clase de software que abstraiga la funcionalidad de un módulo de hardware.
mx.ipn.cic.controladores	En este paquete se almacena la clase base abstracta de cualquier tipo de control para el estandar, de la misma forma se presenta la clase abstracta base para cualquier constructor de controles y de contenedores, y la clase abstracta base para cualquier tabla que deba registrar el estado de los módulos. Aunado a ello, el paquete contiene una interfaz para implementar el mecanismo de recepción de mensajes.
mx.ipn.cic.utilerias	Varias clases utilizadas dentro del framework con propósitos diversos han sido incluidas en este paquete.
mx.ipn.cic.comunicaciones	En este paquete se encuentran las clases con las cuales es posible generar diversos mecanismos de comunicación que sean soportados por el estándar. Entre las clases que contiene se encuentran una clase abstracta que es la base de cualquier mecanismo de comunicación y un evento que sirve para notificar la llegada de un mensaje.
mx.ipn.cic.excepciones	Este paquete contiene el conjunto de excepciones o errores, ya sean en tiempo de compilación o en tiempo de ejecución.

3.1. Paquete mx.ipn.cic.modulos

El diagrama de clases de dicho paquete es mostrado en la figura 4. La única clase contenida en el paquete es la clase abstracta módulo. La razón es simple, cada uno de los módulos de hardware que sean desarrollados deben heredar forzosamente de dicha clase para poder ser considerados módulos dentro del framework y en ella se tienen una gran cantidad de métodos protegidos que pueden ser utilizados por sus clases hijas o derivadas para cumplir con todo el mecanismo de la Capa de Interfaz.

Así, por ejemplo, cuando el desarrollador de una clase ModuloPrueba hereda de Modulo, es posible utilizar todas y cada una de las instrucciones definidas por la especificación funcional del sistema y modularizadas en dicho hardware. Para conseguir esto, se deben utilizar los métodos de la clase base (por ejemplo, doo()) para la creación del mensaje que contiene la instrucción de bajo nivel. El envío y la recepción de los mismos es transparente para el desarrollador.

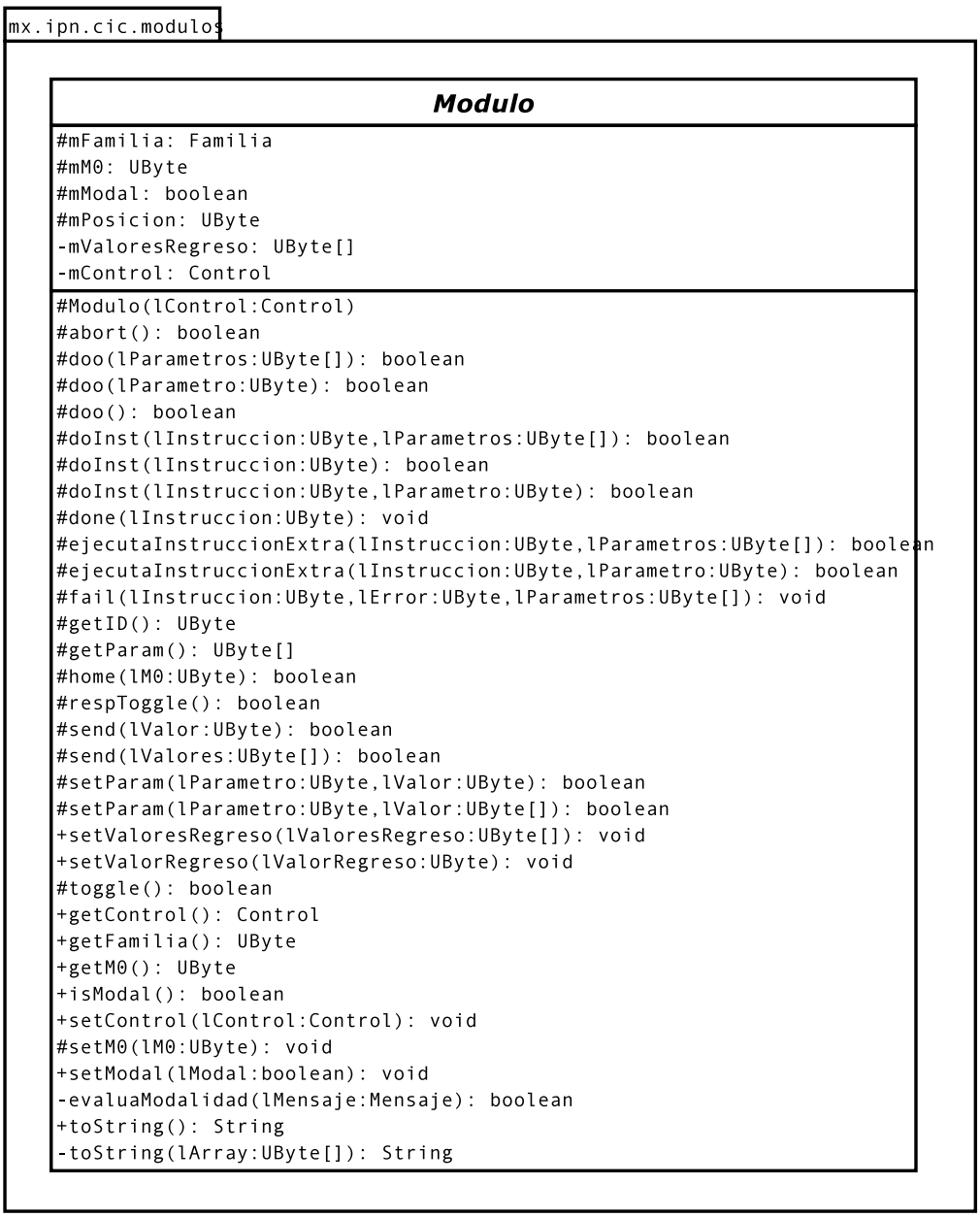


Figura 4: Paquete mx.ipn.cic.modulos del framework.

Hay un concepto implementado llamado modalidad. Se refiere a la capacidad que tiene el usuario final del módulo para decidir si requiere que una instrucción se ejecute en hardware y esperar por la respuesta del mismo (modal) o enviar la instrucción al módulo y no esperar la respuesta del mismo para continuar con la ejecución de su sistema. A continuación se esquematiza este concepto mediante un par de algoritmos ejecutando el mismo pseudo-código:

1. ...

2. Si M1.doo() == verdadero, continúa, de lo contrario, termina.
3. M2.doo();
4. Si M1.doo() == verdadero, continúa, de lo contrario, termina.
5. ...

Primero se analiza lo sucedido en caso de que M1 funcione de manera modal. En la línea 2 se invoca la instrucción doo del módulo y se marca en software como un módulo ocupado. Internamente, en el framework, se espera la respuesta del mismo, que puede ser una instrucción DONE o una instrucción FAIL. Si la respuesta es satisfactoria, la instrucción de la línea 2 retornará un valor verdadero y se podrán ejecutar las líneas siguientes, si la respuesta no llega en un tiempo máximo de espera determinado en el control, se dará por erronea la ejecución y se retornará falso.

En caso de que M1 funcione de manera no modal, la instrucción doo retorna un valor verdadero inmediatamente después de enviar la instrucción al módulo sin esperar respuesta, pero sigue siendo marcado como ocupado. La ejecución de la línea 3 se realiza con el módulo M1 posiblemente ocupado en la ejecución de la instrucción anterior. El hecho de no esperar un valor de respuesta, no quiere decir que dicho valor no llegue, al momento de que esto suceda, el módulo se marca como disponible nuevamente. Este último punto afecta la ejecución de la línea 4, ya que si no ha contestado el módulo o su tiempo de respuesta no ha sido superado, la ejecución no se realiza y se retorna un valor de falso.

Aunado a este mecanismo, el desarrollador del módulo de hardware tiene la facultad de implementar sus instrucciones, dentro del mismo, de manera no modal por defecto, esto es, al recibir la instrucción y antes de comenzar la ejecución de la misma, responder a la Capa de Interfaz con un mensaje DONE (o un FAIL) y con ello, el desarrollador de la Capa de Aplicación no tiene manera de saber en qué instante se ha terminado una función dentro del hardware. Este tipo de instrucciones deben ser bien documentadas.

3.2. Paquete mx.ipn.cic.controladores

El paquete y las clases que contiene son mostradas en la figura 5. La clase abstracta Control define e implementa algunos de los métodos mostrados con la finalidad de proveer al diseñador del sistema o prototipo de la posibilidad de variar el mecanismo de control que se debe seguir en la ejecución del mismo. Cuando alguno de los mecanismos de control programados en el framework no satisfagan los requisitos del prototipo, se debe crear una clase que herede de la clase Control y se deben implementar las instrucciones de la misma para ocultar las de la clase padre y con ello definir la metodología deseada. Al implementar la interfaz mx.ipn.cic.comunicaciones.MensajeRecibidoListener en la clase Control, se debe implementar el método mensajeRecibido, que será invocado por el mecanismo de comunicación para notificar la existencia de un mensaje en espera. La clase abstracta ReferenciasModulos especifica una serie de datos mínimos que debe contener una tabla de referencias de los módulos para poder llevar el control del prototipo.

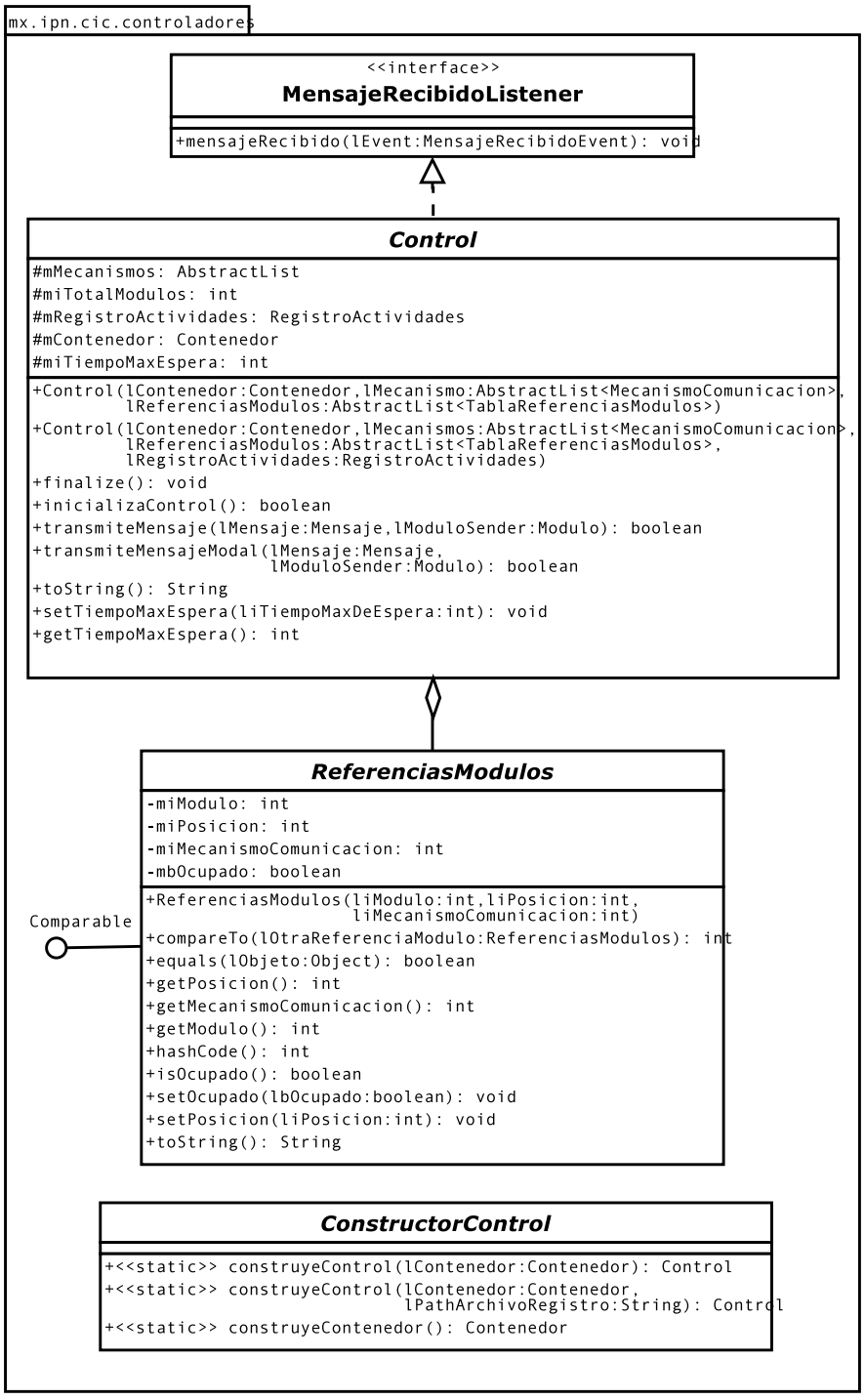


Figura 5: Paquete mx.ipn.cic.controladores del framework.

Por último, se presenta una clase abstracta llamada ConstructorControl, con la cual es posible ocultar todo el procedimiento de creación y administración del control y del contene-

dor (sección 3.3). Este mecanismo permite que el diseñador proporcione una clase con la cual crear un tipo de control y de contenedor mediante una llamada a un método y no mediante una llamada a un constructor, es decir, es posible controlar las variables del constructor del control e inclusive llevar a cabo algunas acciones durante dicho proceso de "creación". Esta clase se basa es un patrón de diseño de software orientado a objetos llamado Constructor o Bilder [4].

Cada una de estas clases abstractas deben ser heredadas para crear clases concretas.

3.3. Paquete mx.ipn.cic.utilerias

En este paquete se encuentran contenidas todas las clases que son necesarias para el framework y cuya funcionalidad es independiente de algún otro paquete. La clase UByte encapsula un dato de tipo short para delimitarlo a un tipo de dato byte sin signo con la capacidad de definir algunas funcionalidades y proporcionando un objeto en lugar de un dato de tipo primitivo. Esta clase sigue el principio de envoltura o wrapper definida en [3].

La clase Familia es una estructura que contiene básicamente dos elementos, una clase (o tipo) de módulo y un fabricante del mismo. Ambos datos, al ser concatenados, generan el ID o familia del módulo. Con esta clase se administran dichos valores para crear la familia o se procesa una familia para obtener el fabricante y el tipo.

La clase Mensaje encapsula la información especificada dentro de estándar por lo cual, en cada mensaje puede haber un UByte de destino, uno de tamaño, uno de instrucción y una cantidad definida de parámetros. Los métodos de dicha clase cumplen con los requerimientos de una clase de datos.

El registro de las actividades del sistema, específicamente de las operaciones de transmisión y recepción, puede ser realizada mediante la clase RegistroActividades la cual implementa un buffer de almacenamiento de tamaño variable que puede almacenar los registros en memoria hasta que se llene y después escribirlos en un archivo de salida, con la finalidad de reducir la escritura en disco duro por cada evento generado.

La clase Contenedor permite encapsular uno o varios mecanismos de control, con uno o varios módulos. De hecho, es necesaria su existencia para que el control tenga los módulos a su disposición en todo momento. La clase ContructorControl puede definir el tipo de colección en la que son almacenados, tanto los objetos de la clase Control como los de la clase Modulo, así como los de sus clases derivadas.

3.4. Paquete mx.ipn.cic.comunicaciones

Las clases contenidas en este paquete se muestran en la figura 7 y con ellas se especifican las normas a seguir para las clases que modelen los sistemas de comunicación entre los módulos Mn y el maestro M0 (en caso de estar implementado en una computadora personal).

La clase MecanismoComunicacion es abstracta y define la necesidad de contar con métodos para recibir y para transferir mensajes, a su vez, define que debe existir una cola para los mensajes de entrada, los mensajes de salida no requieren cola debido a que el estándar da una alta prioridad a las transmisiones y no continúa hasta terminar una transmisión por completo.

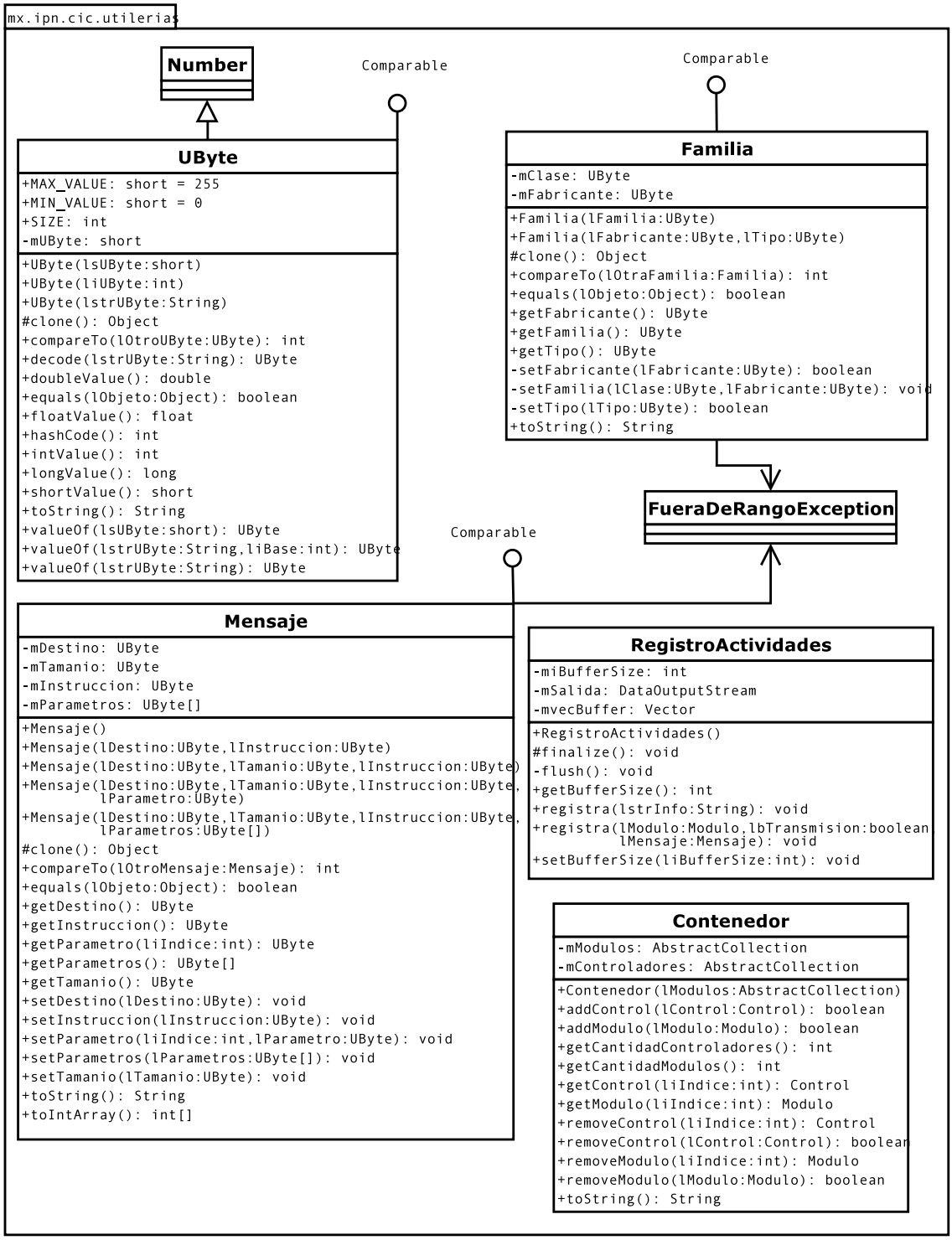


Figura 6: Paquete mx.ipn.cic.utilerias del framework.

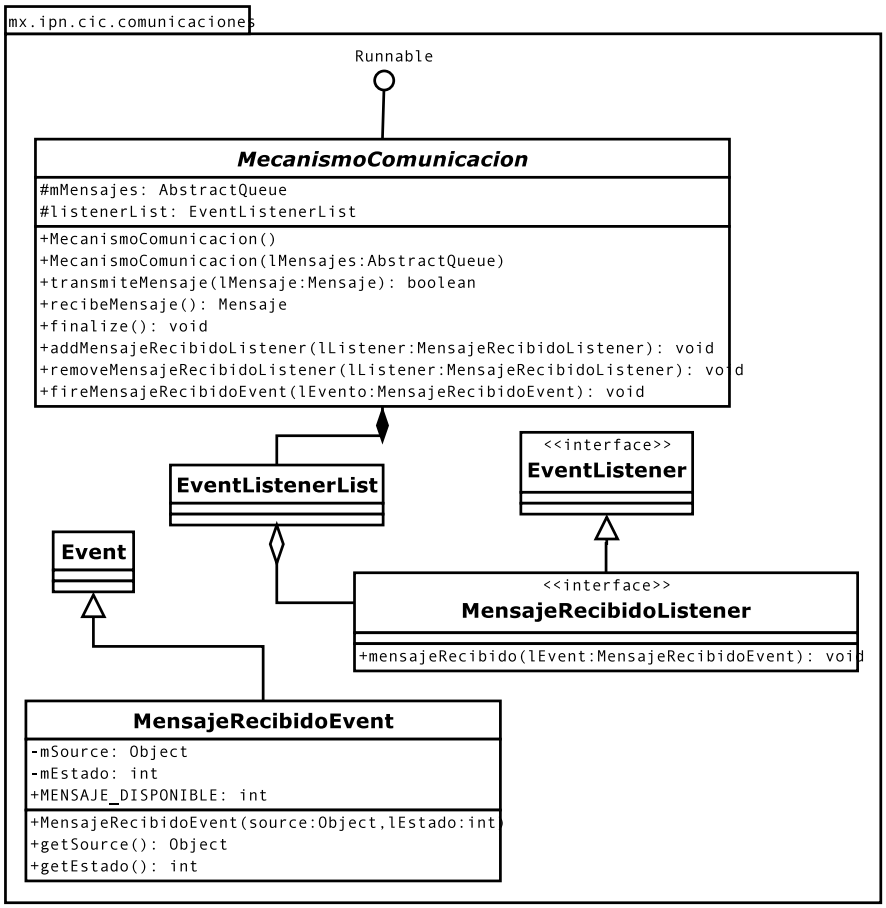


Figura 7: Paquete mx.ipn.cic.comunicaciones del framework.

Cualquier mecanismo de comunicación con el cual se desee trabajar debe ser implementado como clase derivada de ésta.

La clase MensajeRecibidoEvent y la interfaz (del inglés interface) MensajeRecibidoListener proporcionan un mecanismo para notificar la disponibilidad de un mensaje nuevo en el mecanismo de comunicación a aquellas clases que implementen la interfaz.

3.5. Paquete mx.ipn.cic.excepciones

Existen un par de excepciones que deben ser empleadas por otro conjunto de clases dentro del estándar, las cuales estan contenidas en este paquete y son modeladas en la figura 8.

ImposibleInicializarMedioException es lanzada al momento de intentar inicializar el mecanismo de comunicación, de hecho, debe ser arrojada por las clases derivadas de MecanismoComunicacion y validada dentro la clase que genera el mecanismo de comunicación (en general ConstructorControl). La excepción FueraDeRangoException se arroja como consecuencia de intentar utilizar una variable cuyo rango no sea válido.

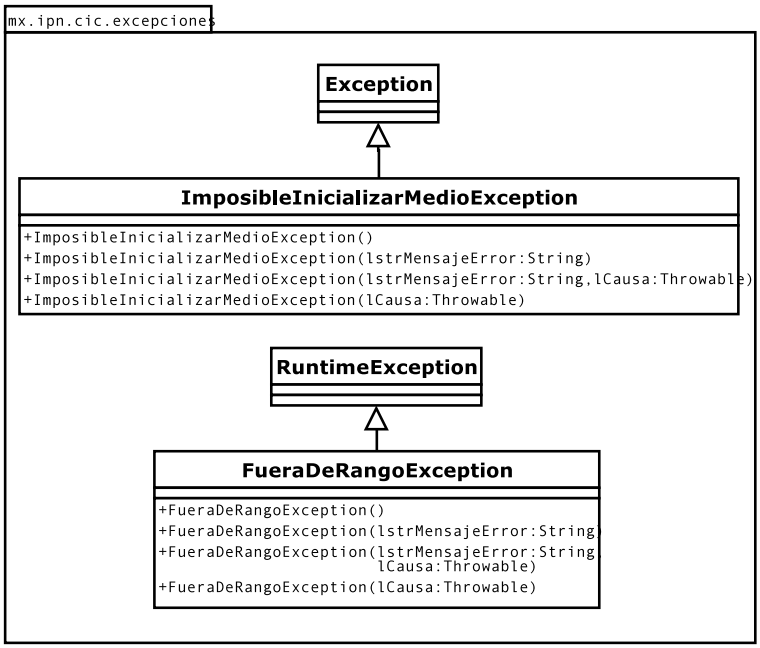


Figura 8: Paquete mx.ipn.cic.excepciones del framework.

3.6. Diagrama de Clases del Framework

El conjunto de clases y las relaciones que existen entre ellas se muestran en el diagrama UML de la figura 9, que omite los atributos y los métodos que ya han sido mostrados en las secciones anteriores.

Un contenedor está compuesto de entre uno y varios módulos y de entre uno y varios controles. El control implementa la interfaz para la notificación de mensajes recibidos, mantiene una referencia del contenedor en el cual se encuentra para poder utilizar el conjunto de módulos y con ella crear las referencias de los mismos en algún tipo de colección, a su vez contiene una clase con la cual lleva el registro de las actividades de transmisión y recepción de los módulos y por último se compone de entre uno y varios mecanismos de comunicación.

Por su parte el mecanismo de comunicación se compone de un arreglo de escuchadores a los cuales debe notificar la disponibilidad de nuevos mensajes mediante la creación de eventos de mensaje recibido y es posible que genere una excepción al intentar inicializar el medio de comunicación.

Tanto el mensaje como la familia del mismo pueden arrojar una excepción en caso de que los valores se encuentren fuera del rango establecido. La clase UByte es utilizada en todo el proceso, desde que llegan los bytes del mensaje y son encapsulados en objetos UByte, hasta la creación de los parámetros de invocación de los métodos de los módulos. Los mensajes son creados tanto en el mecanismo de comunicación como en cada módulo y complementados o utilizados por el control.

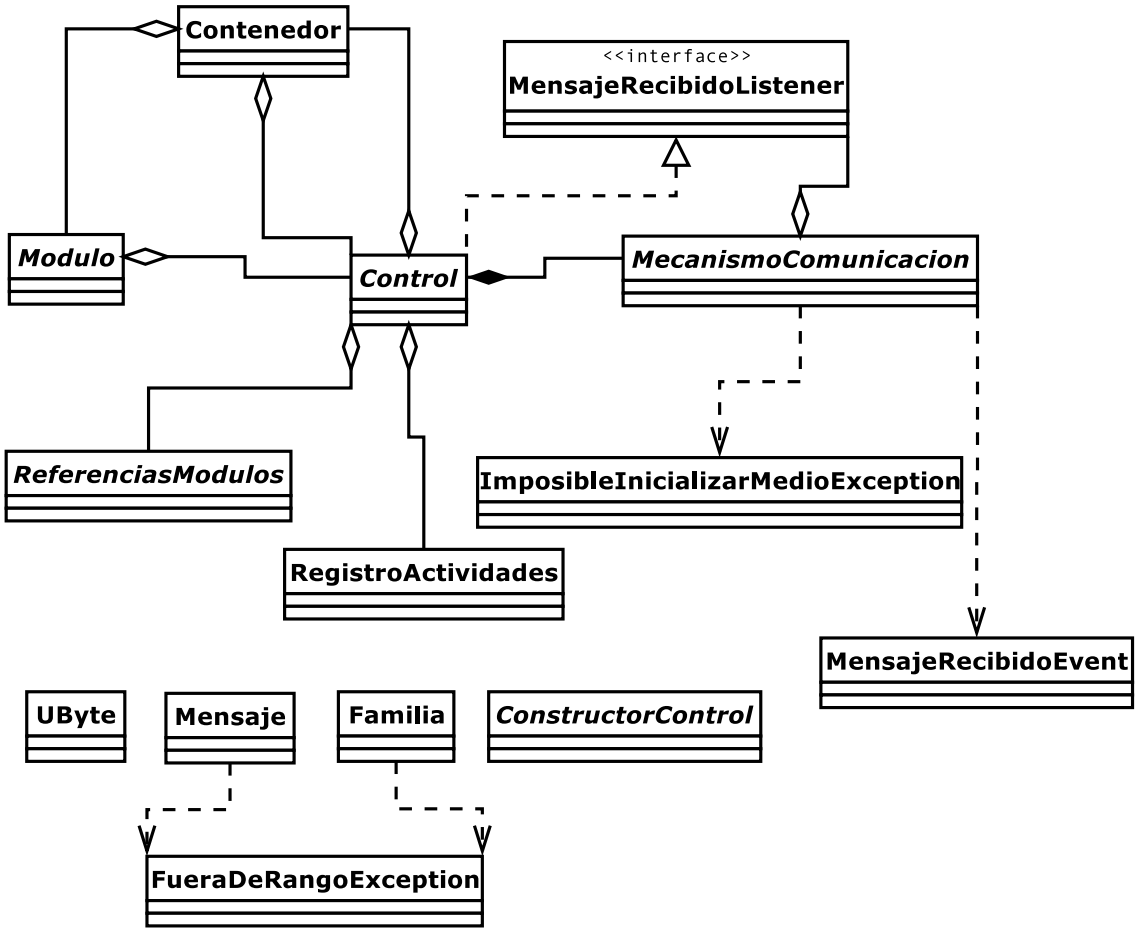


Figura 9: Diagrama de clases del framework.

4. Implementación Base

A continuación se lista el código fuente de cada una de las clases que integran el framework, los comentarios obedecen a las características planteadas por Java, para realizar la documentación mediante Javadoc y explican de manera clara la actividad realizada.

4.1. mx.cic.ipn.modulos

4.1.1. Clase mx.cic.ipn.modulos.Modulo

```
1 package mx.ipn.cic.modulos;
2 import mx.ipn.cic.controladores.Control;
3 import mx.ipn.cic.excepciones.FueraDeRangoException;
4 import mx.ipn.cic.utilerias.*;
5 /**Esta clase abstrae la funcionalidad basica de un modulo y es la clase
6  *padre o base de cualquier tipo de modulo.
7  *Un modulo de hardware debe proporcionar una subclase de esta para implementar
8  *las funcionalidades de alto nivel mediante las instrucciones base del estandar
9  *implementadas aqui.*/
```

```
10 public abstract class Modulo {
11         /**Es el controlador del modulo.*/
12         private Control mControl;
13         /**Familia a la cual pertenece el modulo.*/
14         protected Familia mFamilia;
15         /**Posicion relativa del modulo maestro (M0) a este.*/
16         protected UByte mM0;
17         /**Posicion del modulo en el estandar.*/
18         protected int miPosicion;
19         /**Arreglo para los valores de regreso de las instrucciones getID y
20         *getParam.*/
21         protected UByte mValoresRegreso[];
22         /**Define la funcionalidad del modulo. Modal significa que no habria
23          *respuesta para el modulo(software) hasta recibir una respuesta del
24          *hardware.*/
25         protected boolean mModal;
26         /**Crea una nueva instancia de la clase Modulo con el controlador
27          *<code>lControl</code>.
28          *@see mx.ipn.cic.controladores.Control
29          *@param lControl Es el control que maneja al modulo.*/
30         protected Modulo(Control lControl) {
31                 mControl=lControl;
32                 miPosicion=-1;
33         }
34         /**Instruccion de bajo nivel con la cual se detiene la ejecucion de alguna
35          *instruccion.
36          *Internamente tiene codigo <code>0x00</code>. El formato del
37          *mensaje es: <p>
38          *Destino:Tamanio:abort<p>
39          *Con el mensaje final:<p>
40          *destino:0:0
41          *@return <code>true</code> si la respuesta fue Done, <code>false</code> si
42          *              la respuesta fue Fail.*/
43         protected boolean abort(){
44                 Mensaje lMensaje= new Mensaje(null,new UByte(0),new UByte(0));
45                 return evaluaModalidad(lMensaje);
46         }
47         /**Instruccion de bajo nivel con la cual se ejecuta una actividad por
48          *defecto del modulo sin parametros.
49          * <p>
50          *Internamente tiene codigo <code>0x01</code>. El formato del mensaje es:<p>
51          *Destino:Tamanio:doo<p>
52          *Con el mensaje final:<p>
53          *destino:0:1
54          *@return <code>true</code> si la respuesta fue Done, <code>false</code> si
55          *              la respuesta fue Fail.*/
56         protected boolean doo(){
57                 Mensaje lMensaje= new Mensaje(null,new UByte(0),new UByte(1));
58                 return evaluaModalidad(lMensaje);
59         }
60         /**Instruccion de bajo nivel con la cual se ejecuta una actividad por
61          *defecto del modulo con un parametro.
62          *Internamente tiene codigo <code>0x01</code>. El formato del mensaje es:<p>
63          *Destino:Tamanio:doo:Parametro<p>
64          *Con el mensaje final:<p>
65          *destino:1:1:<code>lParametro</code>
66          *@param lParametro Es el parametro de la instruccion
67          *@return <code>true</code> si la respuesta fue Done, <code>false</code> si
68          *              la respuesta fue Fail.*/
69         protected boolean doo(UByte lParametro){
70                 Mensaje lMensaje= new Mensaje(null,new UByte(1),new UByte(1),
71                 lParametro);
72                 return evaluaModalidad(lMensaje);
73         }
74         /**Instruccion de bajo nivel con la cual se ejecuta una actividad por
```

```
75          *defecto del modulo, puede contener algunos parametros.
76          * <p>
77          *Internamente tiene codigo <code>0x01</code>. El formato del mensaje es:<p>
78          *Destino:Tamanio:doo:Parametros<p>
79          *Con el mensaje final:<p>
80          *destino:<code>lParametros.length</code>:1:<code>lParametros</code>...
81          * (max. 13)
82          *
83          *@param lParametros Son los posibles parametros de la instruccion, maximo
84          *       13
85          *@return <code>true</code> si la respuesta fue Done, <code>false</code> si
86          *               la respuesta fue Fail.
87          *@exception FueraDeRangoException Se arroja en tiempo de ejecucion e indica
88          *               que algun valor ha sido excedido.
89          */
90         protected boolean doo(UByte[] lParametros) throws FueraDeRangoException{
91                 if(lParametros.length>13)
92                         throw new FueraDeRangoException("Cantidad de Parametros ("+
93                         lParametros.length+") fuera de rango (13).");
94                 Mensaje lMensaje= new Mensaje(null,new UByte(lParametros.length),
95                         new UByte(1),lParametros);
96                 return evaluaModalidad(lMensaje);
97         }
98         /**Instruccion de bajo nivel con la cual se ejecuta la
99          *<code>lInstruccion</code> extra definida internamente en hardware. Existen
100         *dos mecanismos para llevar a cabo esta actividad, si el modulo de hardware
101         *tiene menos de 244 instrucciones extra de bajo nivel, estas pueden ser
102         *implementadas mediante <code>ejecutaInstruccionExtra</code> que realiza el
103         *direccionamiento a la instruccion a nivel de mensaje. Con esta opcion, el
104         *direccionamiento debe estar implementado en (codigo del) hardware. Es
105         *recomendable realizar, de ser posible la primera opcion.<p>
106         *Internamente tiene codigo <code>0x02</code>. El formato del mensaje es:<p>
107         *Destino:Tamanio:doInst<p>
108         *Con el mensaje final:<p>
109         *destino:0:2:lInstruccion
110         *@param lInstruccion Es la instruccion de bajo nivel que debe ejecutarse.
111         *@return <code>true</code> si la respuesta fue Done, <code>false</code> si
112         *               la respuesta fue Fail.*/
113        protected boolean doInst(UByte lInstruccion){
114                Mensaje lMensaje= new Mensaje(null,new UByte(1),new UByte(2),
115                        lInstruccion);
116                return evaluaModalidad(lMensaje);
117        }
118        /**Instruccion de bajo nivel con la cual se ejecuta la
119         *<code>lInstruccion</code> extra definida internamente en hardware. Existen
120         *dos mecanismos para llevar a cabo esta actividad, si el modulo de hardware
121         *tiene menos de 244 instrucciones extra de bajo nivel, estas pueden ser
122         *implementadas mediante <code>ejecutaInstruccionExtra</code> que realiza el
123         *direccionamiento a la instruccion a nivel de mensaje. Con esta opcion, el
124         *direccionamiento debe estar implementado en (codigo del) hardware. Es
125         *recomendable realizar, de ser posible la primera opcion.<p>
126         *Internamente tiene codigo <code>0x02</code>. El formato del mensaje es:<p>
127         *Destino:Tamanio:doInst:Parametro<p>
128         *Con el mensaje final:<p>
129         *destino:1:2:lInstruccion:lParametro
130         *@param lInstruccion Es la instruccion de bajo nivel que debe ejecutarse.
131         *@param lParametro Es el parametro de la instruccion.
132         *@return <code>true</code> si la respuesta fue Done, <code>false</code> si
133         *               la respuesta fue Fail.*/
134        protected boolean doInst(UByte lInstruccion, UByte lParametro){
135                UByte lArray[]=new UByte[2];
136                lArray[0]= lInstruccion;
137                lArray[1]= lParametro;
138                Mensaje lMensaje= new Mensaje(null,new UByte(2),new UByte(2),lArray);
139                return evaluaModalidad(lMensaje);
```

```
140         }
141         /**Instruccion de bajo nivel con la cual se ejecuta la
142          *<code>lInstruccion</code> extra definida internamente en hardware.
143          *Existen dos mecanismos para llevar a cabo esta actividad, si el modulo de
144          *hardware tiene menos de 244 instrucciones extra de bajo nivel, estas
145          *pueden ser implementadas mediante <code>ejecutaInstruccionExtra</code> que
146          *realiza el direccionamiento a la instruccion a nivel de mensaje. Con esta
147          *opcion, el direccionamiento debe estar implementado en (codigo del)
148          *hardware. Es recomendable realizar, de ser posible la primera opcion.<p>
149          *
150          *Internamente tiene codigo <code>0x02</code>. El formato del mensaje es:<p>
151          *Destino:Tamanio:doInst:Parametros<p>
152          *Con el mensaje final:<p>
153          *destino:<code>lParametros.length+1</code>:2:lInstruccion:
154          *       lParametros...(max.12)
155          *@param lInstruccion Es la instruccion de bajo nivel que debe ejecutarse.
156          *@param lParametros Son los posibles parametros de la instruccion, maximo
157          *       12
158          *@return <code>true</code> si la respuesta fue Done, <code>false</code> si
159          *               la respuesta fue Fail.
160          *@exception FueraDeRangoException Se arroja en tiempo de ejecucion e indica
161          *               que algun valor ha sido excedido.*/
162         protected boolean doInst(UByte lInstruccion, UByte[] lParametros)
163                 throws FueraDeRangoException{
164                 if(lParametros.length>12)
165                         throw new FueraDeRangoException("Cantidad de Parametros ("+
166                 lParametros.length+") fuera de rango(12).");
167                 UByte lArray[]=new UByte[lParametros.length+1];
168                 lArray[0]= lInstruccion;
169                 for(int i=1;i<lParametros.length+1;i++)
170                         lArray[i]=lParametros[i-1];
171                 Mensaje lMensaje= new Mensaje(null,new UByte(lParametros.length+1),
172                         new UByte(2),lArray);
173                 return evaluaModalidad(lMensaje);
174         }
175         /**Instruccion de bajo nivel con la cual se avisa de la correcta ejecucion
176          *de la instruccion anterior.
177          *Internamente tiene codigo <code>0x03</code>. El formato del
178          *mensaje es: <p>
179          * Destino:Tamanio:done:instruccionRealizada<p>
180          *Con el mensaje final:<p>
181          * destino:1:3:<code>lInstruccion</code><p>
182          *@param lInstruccion Es la instruccion de bajo nivel ejecutada.
183         */
184         protected void done(UByte lInstruccion){
185                 Mensaje lMensaje= new Mensaje(null,new UByte(1),new UByte(3),
186                         lInstruccion);
187                 mControl.transmiteMensaje(lMensaje,this);
188         }
189         /**
190          * Instruccion de bajo nivel con la cual se avisa de la incorrecta ejecucion
191          * de la instruccion anterior y la clave del error.
192          * Internamente tiene codigo <code>0x04</code>. El formato del
193          * mensaje es: <p>
194          * Destino:Tamanio:fail:instruccionFallida:claveError<p>
195          * Con el mensaje final:<p>
196          * destino:1:4:<code>lInstruccion</code>:<code>lClaveError</code>
197          * @param lClaveError Es la clave del error ocurrido.
198          * @param lInstruccion Es la instruccion de bajo nivel fallida.*/
199         protected void fail(UByte lInstruccion, UByte lClaveError){
200                 UByte lArray[]=new UByte[2];
201                 lArray[0]= lInstruccion;
202                 lArray[1]= lClaveError;
203                 Mensaje lMensaje= new Mensaje(null,new UByte(2),new UByte(4),lArray);
204                 mControl.transmiteMensaje(lMensaje,this);
```

```
205         }
206         /**Instruccion de bajo nivel con la cual se solicita al modulo su familia.
207          *Espera la respuesta mediante un mensaje send y regresa dicho valor.
208          *Internamente tiene codigo <code>0x05</code>. El formato del
209          *mensaje es: <p>
210          *Destino:Tamanio:getID<p>
211          *Con el mensaje final:<p>
212          *destino:0:5
213          *@return La familia del modulo.*/
214         protected Familia getID(){
215                 Mensaje lMensaje= new Mensaje(null,new UByte(0),new UByte(5));
216                 if(!mControl.transmiteMensaje(lMensaje,this))
217                         return null;
218                 return new Familia(mValoresRegreso[0]);
219         }
220         /**
221          * Instruccion de bajo nivel con la cual se solicita al modulo alguno de sus
222          * parametros internos. Espera la respuesta mediante un mensaje send y
223          * regresa dicho valor.
224          * Internamente tiene codigo <code>0x06</code>. El formato del
225          * mensaje es: <p>
226          * Destino:Tamanio:getParam:<code>lParametro</code><p>
227          * Con el mensaje final:<p>
228          * destino:1:6:lParametro
229          * @return El parametro solicitado, puede ser de mas de 8bits
230          * @param lParametro Es el parametro en el modulo que se desa obtener.*/
231         protected UByte[] getParam(UByte lParametro){
232                 Mensaje lMensaje=new Mensaje(null,new UByte(1),new UByte(6),lParametro);
233                 if(!mControl.transmiteMensaje(lMensaje,this))
234                         return null;
235                 return mValoresRegreso;
236         }
237         /**
238          *Instruccion de bajo nivel con la cual se inicializa un modulo. Los
239          *fabricantes de hardware requieren de valores iniciales en algunas de sus
240          *variables, el estandar necesita enviar el M0 a cada modulo.
241          * Internamente tiene codigo <code>0x07</code>. El formato del
242          * mensaje es: <p>
243          * Destino:Tamanio:home:M0<p>
244          * Con el mensaje final:<p>
245          * destino:1:7:<code>lM0</code>
246          * @return <code>true</code> si la respuesta fue Done, <code>false</code> si
247          *                 la respuesta fue Fail.
248          * @param lPosicion Es la posicion en la cual se encuentra el modulo. Su
249          *       "direccion"
250          * @param lM0 Es la posicion relativa del maestro al modulo.*/
251         protected boolean home(UByte lM0, UByte lPosicion){
252                 UByte lArray[]=new UByte[2];
253                 lArray[0]= lM0;
254                 lArray[1]= lPosicion;
255                 Mensaje lMensaje= new Mensaje(null,new UByte(2),new UByte(7),lArray);
256                 return evaluaModalidad(lMensaje);
257         }
258         /**Instruccion de bajo nivel con la cual se habilitan y deshabilitan los
259          *mensajes de respuesta de los modulos al maestro.
260          *Internamente tiene codigo <code>0x08</code>. El formato del
261          *mensaje es: <p>
262          *Destino:Tamanio:respTogg<p>
263          *Con el mensaje final:<p>
264          *destino:0:8
265          *@return <code>true</code> si la respuesta fue Done, <code>false</code> si
266          *                 la respuesta fue Fail.
267          *@see mx.ipn.cic.modulos.Modulo#mModal*/
268         protected boolean respTogg(){
269                 Mensaje lMensaje= new Mensaje(null,new UByte(0),new UByte(8));
```

```
270             return evaluaModalidad(lMensaje);
271         }
272         /**Instruccion de bajo nivel con la cual se envian las respuestas al maestro
273          *Internamente tiene codigo <code>0x09</code>. El formato del
274          *mensaje es: <p>
275          *Destino:Tamanio:send:Valor<p>
276          *Con el mensaje final:<p>
277          *destino:1:9:<code>lValor</code>
278          *@param lValor Es el valor solicitado anteriormente.
279          *@return <code>true</code> si la respuesta fue Done, <code>false</code> si
280          *              la respuesta fue Fail.*/
281         protected boolean send(UByte lValor){
282             Mensaje lMensaje= new Mensaje(null,new UByte(1),new UByte(9),lValor);
283             return evaluaModalidad(lMensaje);
284         }
285         /**Instruccion de bajo nivel con la cual se envian las respuestas al maestro.
286          *Internamente tiene codigo <code>0x09</code>. El formato del
287          *mensaje es: <p>
288          *Destino:Tamanio:send:Valor<p>
289          *Con el mensaje final:<p>
290          *destino:<code>lValor.length</code>:9:<code>lValor</code>
291          *@param lValor Es el valor solicitado anteriormente.
292          *@return <code>true</code> si la respuesta fue Done, <code>false</code> si
293          *              la respuesta fue Fail.
294          *@exception FueraDeRangoException Se arroja en tiempo de ejecucion e indica
295          *              que algun valor ha sido excedido.*/
296         protected boolean send(UByte[] lValor) throws FueraDeRangoException{
297             if(lValor.length>13)
298                 throw new FueraDeRangoException("Tamanio de valor ("+lValor.length+
299                         ") fuera de rango (13).");
300             Mensaje lMensaje= new Mensaje(null,new UByte(lValor.length),
301                     new UByte(9),lValor);
302             return evaluaModalidad(lMensaje);
303         }
304         /**Instruccion de bajo nivel con la cual se actualiza algun parametro del
305          *modulo de hardware.
306          *Internamente tiene codigo <code>0x0A</code>. El formato del
307          *mensaje es: <p>
308          *Destino:Tamanio:setParam:Parametro:Valor<p>
309          *Con el mensaje final:<p>
310          *destino:2:10:<code>lParametro</code>:<code>lValor</code>
311          *@param lParametro Es el parametro por modificar.
312          *@param lValor Es el valor nuevo para el parametro <code>lParametro</code>.
313          *@return <code>true</code> si la respuesta fue Done, <code>false</code> si
314          *              la respuesta fue Fail.*/
315         protected boolean setParam(UByte lParametro, UByte lValor){
316             UByte lArray[]=new UByte[2];
317             lArray[0]= lParametro;
318             lArray[1]= lValor;
319             Mensaje lMensaje= new Mensaje(null,new UByte(2),new UByte(10),lArray);
320             return evaluaModalidad(lMensaje);
321         }
322         /**Instruccion de bajo nivel con la cual se actualiza algun parametro del
323          *modulo de hardware.
324          *Internamente tiene codigo <code>0x0A</code>. El formato del
325          *mensaje es: <p>
326          *Destino:Tamanio:setParam:Parametro:Valor<p>
327          *Con el mensaje final:<p>
328          *destino:<code>lValor.length</code>:10:<code>lParametro</code>:
329          *      <code>lValor</code>
330          *@param lParametro Es el parametro por modificar.
331          *@param lValor Es el valor nuevo para el parametro <code>lParametro</code>.
332          *@return <code>true</code> si la respuesta fue Done, <code>false</code> si
333          *              la respuesta fue Fail o si <code>lValor.length</code> se
334          *              encuentra fuera del rango establecido por la cantidad de
```

```
335          *                parametros.
336          *@exception FueraDeRangoException Se arroja en tiempo de ejecucion e indica
337          *                que algun valor ha sido excedido.*/
338         protected boolean setParam(UByte lParametro, UByte[] lValor)
339                         throws FueraDeRangoException{
340                 if(lValor.length>12)
341                         throw new FueraDeRangoException("Tamanio de valor ("+lValor.length+
342                                 ") fuera de rango (12).");
343                 UByte lArray[]=new UByte[lValor.length+1];
344                 lArray[0]= lParametro;
345                 for(int i=1;i<lValor.length+1;i++)
346                         lArray[i]=lValor[i-1];
347                 Mensaje lMensaje= new Mensaje(null,new UByte(lValor.length+1),
348                         new UByte(10),lArray);
349                 return evaluaModalidad(lMensaje);
350         }
351         /**Activa o desactiva el modulo, es decir su funcionalidad, ya que si el
352          *mecanismo de comunicacion requiere su presencia para cerrar un token ring,
353          *por ejemplo, debe ser capaz de cubrir la comunicacion.
354          *Internamente tiene codigo <code>0x0B</code>. El formato del
355          *mensaje es: <p>
356          *Destino:Tamanio:toggle<p>
357          *Con el mensaje final:<p>
358          *destino:0:11
359          *@return <code>true</code> si la respuesta fue Done, <code>false</code> si
360          *                la respuesta fue Fail.*/
361         protected boolean toggle(){
362                 Mensaje lMensaje= new Mensaje(null,new UByte(0),new UByte(11));
363                 return evaluaModalidad(lMensaje);
364         }
365         /**Sirve para ejecutar alguna instruccion de bajo nivel, definida en
366          *hardware.<p>
367          *Internamente no tiene codigo definido, ya que la instruccion es
368          *precisamente <code>lInstruccion</code>. El formato del mensaje es:
369          *Destino:Tamanio:InstruccionExtra:Parametro<p>
370          *Con el mensaje final:<p>
371          *destino:1:<code>lInstruccion</code>:<code>lParametro</code>
372          *@param lInstruccion Es la instruccion de bajo nivel que se desea ejecutar.
373          *@param lParametro Es el parametro de la instruccion
374          *@return <code>true</code> si la respuesta fue Done, <code>false</code> si
375          *                la respuesta fue Fail.
376          *@see mx.ipn.cic.modulos.Modulo#doInst(UByte,UByte[])*/
377         protected boolean ejecutaInstruccionExtra(UByte lInstruccion,
378                         UByte lParametro){
379                 Mensaje lMensaje=new Mensaje(null,new UByte(2),lInstruccion,lParametro);
380                 return evaluaModalidad(lMensaje);
381         }
382         /**Sirve para ejecutar alguna instruccion de bajo nivel, definida en
383          *hardware.<p>
384          *Internamente no tiene codigo definido, ya que la instruccion es
385          *precisamente <code>lInstruccion</code>. El formato del mensaje es:
386          *Destino:Tamanio:InstruccionExtra:Parametros<p>
387          *Con el mensaje final:<p>
388          *destino:<code>lParametros.length</code>:<code>lInstruccion</code>:
389          *       <code>lParametros</code>
390          *@param lInstruccion Es la instruccion de bajo nivel que se desea ejecutar.
391          *@param lParametros Son los posibles parametros de la instruccion
392          *@return <code>true</code> si la respuesta fue Done, <code>false</code> si
393          *                la respuesta fue Fail si <code>lParametros.length</code> se
394          *                encuentra fuera del rango establecido por la cantidad de
395          *                parametros.
396          *@see mx.ipn.cic.modulos.Modulo#doInst(UByte,UByte[])
397          *@exception FueraDeRangoException Se arroja en tiempo de ejecucion e indica
398          *                que algun valor ha sido excedido.*/
399         protected boolean ejecutaInstruccionExtra(UByte lInstruccion,
```

```
400             UByte[] lParametros)throws FueraDeRangoException{
401             if(lParametros.length>12)
402                     throw new FueraDeRangoException("Tamanio de valor ("+
403             lParametros.length+") fuera de rango (12).");
404             Mensaje lMensaje= new Mensaje(null,new UByte(lParametros.length),
405             lInstruccion,lParametros);
406             return evaluaModalidad(lMensaje);
407     }
408     /**Sirve para direccionar la transmision a modal o no modal.
409      *@param lMensaje Mensaje que se desea enviar.
410      *@return <CODE>true</CODE> en caso de exito, <CODE>false</CODE> de lo
411      *contrario.*/
412     private boolean evaluaModalidad(Mensaje lMensaje){
413             if(!mModal)
414                     return mControl.transmiteMensaje(lMensaje,this);
415             return mControl.transmiteMensajeModal(lMensaje,this);
416     }
417     /**Regresa el controlador del modulo.
418      *@return El controlador actual del modulo.*/
419     public Control getControl(){
420             return mControl;
421     }
422     /**Regresa la familia del modulo.
423      *@return La familia del modulo.*/
424     public Familia getFamilia(){
425             return mFamilia;
426     }
427     /**Regresa el M0 del modulo.
428      *@return La posicion relativa del modulo maestro a este.*/
429     public UByte getM0(){
430             return mM0;
431     }
432     /**Regresa la posicion del modulo.
433      *@return La posicion del modulo desde el maestro.*/
434     public int getPosicion(){
435             return miPosicion;
436     }
437     /**Regresa el estado de modalidad del modulo.
438      *@return <code>true</code> si el modulo es modal, <code>false</code> de lo
439      *          contrario.
440      *@see mx.ipn.cic.modulos.Modulo#mModal*/
441     public boolean isModal(){
442             return mModal;
443     }
444     /**Actualizaa el controlador del modulo.
445      *@param lControl Es el nuevo controlador del modulo.*/
446     public void setControl(Control lControl){
447             mControl=lControl;
448     }
449     /**Actualiza el M0 del modulo.
450      *@param lM0 La nueva posicion relativa del modulo maestro a este.*/
451     public void setM0(UByte lM0){
452             mM0=lM0;
453     }
454     /**Actualiza la posicion del modulo.
455      *@param liPosicion La nueva posicion del modulo al maestro.*/
456     public void setPosicion(int liPosicion){
457             miPosicion=liPosicion;
458     }
459     /**Actualiza el estado de modalidad del modulo.
460      *@param lModal <code>true</code> para modal, <code>false</code> de lo
461      *          contrario.
462      *@see mx.ipn.cic.modulos.Modulo#mModal*/
463     public void setModal(boolean lModal){
464             mModal=lModal;
```

```
465         }
466         /**Actualiza los valores de regreso del modulo.
467          *@param lValoresRegreso Arreglo de UByte que contiene los valores de
468          *       regreso.*/
469         public void setValoresRegreso(UByte []lValoresRegreso){
470                 mValoresRegreso=lValoresRegreso;
471         }
472         /**Actualiza los valores de regreso del modulo.
473          *@param lValoresRegreso Arreglo de UByte que contiene el valor de regreso.*/
474         public void setValoresRegreso(UByte lValoresRegreso){
475                 mValoresRegreso=new UByte[1];
476                 mValoresRegreso[0]=lValoresRegreso;
477         }
478         /**Representacion textual de la clase.
479          *@return Una cadena con el formato: Class.name:Control,Familia,M0,mModal*/
480         public String toString() {
481                 return this.getClass()+":"+
482                         "\tFamilia:"+mFamilia+
483                         "\tPosicion:"+miPosicion+
484                         "\tM0:"+mM0+
485                         "\tModal:"+mModal;
486         }
487         /**Convierte a cadena un arreglo de UBytes
488          *@param lArray Arreglo de <CODE>UByte</CODE> para ser convertido.
489          *@return Una cadena que contiene el arreglo.*/
490         private String toString(UByte[] lArray){
491                 String lstrTempo="[";
492                 for(int i=0;i<lArray.length;lstrTempo+=lArray[i++]+",");
493                 return lstrTempo+"]";
494         }
495 }
```

4.2. mx.cic.ipn.controladores

4.2.1. Clase mx.cic.ipn.controladores.Control

```
1  package mx.ipn.cic.controladores;
2  import java.util.AbstractList;
3  import mx.ipn.cic.comunicaciones.MecanismoComunicacion;
4  import mx.ipn.cic.comunicaciones.MensajeRecibidoListener;
5  import mx.ipn.cic.utilerias.*;
6  import mx.ipn.cic.modulos.Modulo;
7  /**Clase abstracta que especifica los elementos de un control para el estandar
8           *de hardware y software.
9           * @see  mx.ipn.cic.controladores.ConstructorControl*/
10 public abstract class Control implements MensajeRecibidoListener{
11         /**Arreglo de mecanismos utilizados.*/
12         protected static AbstractList<MecanismoComunicacion> mMecanismos;
13         /**Son las referencias completas de los modulos*/
14         protected AbstractList<ReferenciasModulos> mReferenciasModulos;
15         /**Es el total de modulos conectados al controlador y en funcionamiento.*/
16         protected int miTotalModulos;
17         /**Es el objeto encargado de registrar las actividades del control con
18          *respecto a los modulos y sus mensajes enviados y recibidos.*/
19         protected RegistroActividades mRegistroActividades;
20         /**Es el contenedor del Control.*/
21         protected Contenedor mContenedor;
22         /**Es el tiempo maximo que debe esperar una respuesta en milisegundos
23          *(transmision modal). Valor por defecto: 1000*/
24         protected int miTiempoMaxDeEspera=1000;
25         /** Crea una nueva instancia de la clase Control con la lista de modulos.
26          *@param lContenedor Es el contenedor del control y los modulos.
27          *@param lMecanismos Una lista de mecanismos para utilizar.
28          *@param lReferenciasModulos Una lista de caracteristicas de los modulos
```

```
29          *           contenidos.
30          */
31          public Control(Contenedor lContenedor,
32                  AbstractList<MecanismoComunicacion> lMecanismos,
33                  AbstractList<ReferenciasModulos> lReferenciasModulos) {
34                  mContenedor=lContenedor;
35                  mMecanismos=lMecanismos;
36                  mReferenciasModulos=lReferenciasModulos;
37                  miTotalModulos=0;
38                  mRegistroActividades=null;
39          }
40          /** Crea una nueva instancia de la clase Control con la lista de modulos.
41           *@param lContenedor Es el contenedor del control y los modulos.
42           *@param lMecanismos Una lista de mecanismos para utilizar.
43           *@param lReferenciasModulos Una lista de caracteristicas de los modulos
44           *          contenidos.
45           *@param lRegistroActividades Es el Administrador del registro de
46           *      actividades.*/
47          public Control(Contenedor lContenedor,
48                  AbstractList<MecanismoComunicacion> lMecanismos,
49                  AbstractList<ReferenciasModulos> lReferenciasModulos,
50                  RegistroActividades lRegistroActividades) {
51                  mContenedor=lContenedor;
52                  mMecanismos=lMecanismos;
53                  mReferenciasModulos=lReferenciasModulos;
54                  miTotalModulos=0;
55                  mRegistroActividades=lRegistroActividades;
56          }
57          /**Camino por el cual un modulo envia un mensaje. Completa el mensaje
58           *y lo envia al canal de comunicacion asignado al modulo.
59           *@param lMensaje Es el mensaje que debe completarse con la posicion del
60           *          destinatario y enviarse.
61           *@param lModuloSender Es el modulo que envia el mensaje.
62           *@return <code>true</code> si el mensaje se ha transmitido exitosamente,
63           *          <code>false</code> de lo contrario. Considerar la opcion de
64           *          modalidad.
65           *@see mx.ipn.cic.modulos.Modulo#mModal*/
66          public abstract boolean transmiteMensaje(Mensaje lMensaje,
67                          Modulo lModuloSender);
68          /**Camino por el cual un modulo envia un mensaje. Completa el mensaje
69           *y lo envia al canal de comunicacion asignado al modulo. Modal significa
70           *que se esperara la respuesta del modulo con la finalidad de asegurar su
71           *correcta o incorrecta operacion.
72           *@param lMensaje Es el mensaje que debe completarse con la posicion del
73           *          destinatario y enviarse.
74           *@param lModuloSender Es el modulo que envia el mensaje.
75           *@return <code>true</code> si el mensaje se ha transmitido exitosamente,
76           *          <code>false</code> de lo contrario. Considerar la opcion de
77           *          modalidad.
78           *@see mx.ipn.cic.modulos.Modulo#mModal*/
79          public abstract boolean transmiteMensajeModal(Mensaje lMensaje,
80                          Modulo lModuloSender);
81          /**Debe analizar los elementos conectados externamente, actualizar la
82           *posicion en la tabla de modulos y referencias y ejecutar el home de todos
83           *los modulos conectados.
84           *@return <CODE>true</CODE> en caso de exito, <CODE>false</CODE> de lo
85           *      contrario.*/
86          public abstract boolean inicializaControl();
87          /**Actualiza el tiempo maximo de espera.
88           *@param liTiempoMaxDeEspera Es el tiempo en milisegundos que se debe
89           * esperar una respuesta de un modulo cuando la transmision es modal.
90           * Si el numero es negativo, la espera sera infinita (peligroso).*/
91          public void setTiempoMaxEspera(int liTiempoMaxDeEspera){
92                  miTiempoMaxDeEspera=liTiempoMaxDeEspera;
93          }
```

```
94          /**Regresa el tiempo maximo de espera.
95           *@return El tiempo en milisegundos que se debe
96           * esperar una respuesta de un modulo cuando la transmision es modal.*/
97          public int getTiempoMaxEspera(){
98                  return miTiempoMaxDeEspera;
99          }
100         /**Representacion textual de la clase.
101          *@return Una cadena con el formato: TotalModulos\nModulo 1\n Modulo 2.*/
102         public String toString() {
103                 return getClass()+"\n"+mContenedor.toString();
104         }
105         /**Finaliza cada Mecanismo de comunicacion.*/
106         public void finalize(){
107                 for(MecanismoComunicacion m:mMecanismos){
108                         m.finalize();
109                 }
110         }
111 }
```

4.2.2. Clase mx.cic.ipn.controladores.ReferenciasModulos

```
1  package mx.ipn.cic.controladores;
2  /**Clase que es almacenada en la lista de modulos del contenedor.*/
3  public abstract class ReferenciasModulos
4                  implements Comparable<ReferenciasModulos>{
5          /**Indice del modulo en la lista abstracta de modulos.*/
6          private int miModulo;
7          /**Posicion en el modelo de cada modulo.*/
8          private int miPosicion;
9          /**Indice del mecanismo de comunicacion en la lista abstracta.*/
10         private int miMecanismoComunicacion;
11         /**Indica si el modulo se encuentra realizando una operacion
12          *(<code>true</code>) o si esta disponible (<code>false</code>).*/
13         private boolean mbOcupado;
14         /** Crea una nueva instancia de la clase ReferenciasModulos
15          *@param liModulo Indice del modulo en la lista abstracta de modulos.
16          *@param liPosicion Posicion en el modelo de cada modulo.
17          *@param liMecanismoComunicacion Indice del mecanismo de comunicacion en la
18          * lista abstracta.*/
19         public ReferenciasModulos(int liModulo, int liPosicion,
20                         int liMecanismoComunicacion){
21                 miModulo = liModulo;
22                 setPosicion(liPosicion);
23                 miMecanismoComunicacion = liMecanismoComunicacion;
24                 mbOcupado=false;
25         }
26         /**Regresa el indice del modulo en la lista.
27          *@return El indice del modulo en la lista.*/
28         public int getModulo(){
29                 return miModulo;
30         }
31         /**Regresa la "posicion" del modulo en la cadena de hardware.
32          *@return La "posicion" del modulo en la cadena de hardware*/
33         public int getPosicion(){
34                 return miPosicion;
35         }
36         /**Regresa el indice del mecanismo de comunicacion en la lista.
37          *@return El indice del mecanismo de comunicacion en la lista.*/
38         public int getMecanismoComunicacion(){
39                 return miMecanismoComunicacion;
40         }
41         /**Actualiza la "posicion" en la cual se encuentra el modulo.
42          *@param liPosicion Es la nueva "posicion"*/
43         public void setPosicion(int liPosicion) {
44                 miPosicion = liPosicion;
```

```
45          }
46          /**Actualiza el estado del modulo.
47           *@param lbOcupado <code>true</code> si se encuentra ocupado,
48           * <code>false</code> de lo contrario.*/
49          public void setOcupado(boolean lbOcupado){
50                  mbOcupado=lbOcupado;
51          }
52          /**Responde el estado del modulo.
53           *@return <code>true</code> si se encuentra ocupado, <code>false</code> de
54           * lo contrario.*/
55          public boolean isOcupado(){
56                  return mbOcupado;
57          }
58          /**Representacion textual de la clase.
59           *@return Una cadena con el formato: (Modulo,Posicion,Mecanismo)*/
60          public String toString() {
61                  return "("+miModulo+","+miPosicion+","+miMecanismoComunicacion+")";
62          }
63          /**Compara este objeto con <code>lObjeto</code>, en caso de ser otra
64           * <code>ReferenciasModulos</code> utiliza como valor de comparacion
65           * el codigo hash.
66           * @param lObjeto El objeto con el cual compararse.
67           * @return <code>true</code> si los objetos son iguales;
68           *                      <code>false</code> de lo contrario.*/
69          public boolean equals(Object lObjeto) {
70                  if (lObjeto instanceof ReferenciasModulos)
71                          return hashCode()==lObjeto.hashCode();
72                  return false;
73          }
74          /** Compara una ReferenciasModulos con otra.
75           *@param lOtraReferenciasModulos Es otra ReferenciasModulos con la
76           *          cual compararse.
77           *@return El valor <code>0</code> si ambas ReferenciasModulos son la
78           *          misma, un valor menor a <code>0</code> si esta
79           *          ReferenciasModulos es menor a otra en por lo menos uno de los
80           *          valores; y un valor mayor a <code>0</code> si esta
81           *          ReferenciasModulos es mayor a la otra en por lo menos uno de los
82           *          valores.*/
83          public int compareTo(ReferenciasModulos lOtraReferenciasModulos){
84                  int resp=0;
85                  if((resp=miModulo-lOtraReferenciasModulos.miModulo)!=0)
86                          return resp;
87                  else if((resp=miPosicion-lOtraReferenciasModulos.miPosicion)!=0)
88                          return resp;
89                  else if((resp=miMecanismoComunicacion-
90                                  lOtraReferenciasModulos.miMecanismoComunicacion)!=0)
91                          return resp;
92                  else
93                          return 0;
94          }
95          /**Codigo hash del objeto formado por la concatenacion del indice del modulo
96           * y la posicion del mismo.
97           *@return El codigo hash del objeto.*/
98          public int hashCode(){
99                  int tmp=miModulo<<8;
100                 return tmp|(miPosicion&255);
101         }
102 }
```

4.2.3. Clase mx.cic.ipn.controladores.ConstructorControl

```
1 package mx.ipn.cic.controladores;
2 import mx.ipn.cic.utilerias.Contenedor;
3 import mx.ipn.cic.excepciones.ImposibleInicializarMedioException;
4 /**Clase abstracta que define a los constructores de controladores para
```

```
5  * distintos tipos de Control.*/
6  public abstract class ConstructorControl {
7          /**Crea y regresa un Control con su respectivo contenedor en el cual se
8           *agrega. Un usuario de estandar debe utilizar siempre este metodo para
9           *crear el controlador.
10          *@param lContenedor Es el contenedor en el cual esta agregado el Control.
11          *@return El Controlador deseado.
12          *@throws mx.ipn.cic.excepciones.ImposibleInicializarMedioException
13          *       Ejecutada cuando no es posible iniciar el medio de comunicacion*/
14         public static Control construyeControl(Contenedor lContenedor)
15                         throws ImposibleInicializarMedioException {return null;}
16         /**
17          * Crea y regresa un Control con su respectivo contenedor en el cual se
18          * agrega y que escribe un registro de las actividades de los modulos.
19          * Un usuario de estandar debe utilizar siempre este metodo para crear
20          * el controlador.
21          * @return El Controlador deseado.
22          * @param lContenedor Es el contenedor en el cual esta agregado el Control.
23          * @param lPathArchivoRegistro Path del archivo de registro de las
24          *       actividades.
25          * @throws mx.ipn.cic.excepciones.ImposibleInicializarMedioException
26          *       Ejecutada cuando no es posible iniciar el medio de comunicacion*/
27         public static Control construyeControl(Contenedor lContenedor,
28                         String lPathArchivoRegistro)
29                         throws ImposibleInicializarMedioException {return null;}
30         /**Crea y regresa un contenedor con las caracteristicas definidas por el
31          *tipo de control implementado.
32          *@return El Contenedor deseado.*/
33         public static Contenedor construyeContenedor(){return null;}
34 }
```

4.3. mx.cic.ipn.utilerias

4.3.1. Clase mx.cic.ipn.utilerias.UByte

```
1  package mx.ipn.cic.utilerias;
2  /**Esta clase es un wrapper de un byte sin signo, encapsula un short, pero
3   *proporciona la funcionalidad de un byte sin signo, lamentablemente no existe
4   *el tipo de dato primitivo unsigned byte.*/
5  public final class UByte extends Number implements Comparable<UByte> {
6          /** Una constante con el valor minimo de UByte (0)*/
7          public static final short   MIN_VALUE = 0;
8          /** Una constante con el valor maximo de UByte (255)*/
9          public static final short   MAX_VALUE = 255;
10         /**La cantidad de bits utilizados para representar un byte sin signo*/
11         public static final int SIZE = 8;
12         /**Almacenado del valor*/
13         private final short mUByte;
14         /*Clase que almacena todos los 255 posibles objetos UByte*/
15         /**Clase para contener los UByte instanciados*/
16         private static class UByteCache {
17                 /**Contruye un objeto de la clase*/
18                 private UByteCache(){}
19                 /**Arreglo de valores UByte*/
20                 static final UByte cache[] = new UByte[255];
21                 static {
22                 for(int i = 0; i < cache.length; i++)
23                                 cache[i] = new UByte((short)(i));
24                 }
25         }
26         /**Obtiene una instancia de <tt>UByte</tt> representando el valor de
27          *<tt>short</tt>.
28          *@return Una instancia de <tt>UByte</tt> que representa a <tt>lsUByte</tt>.
29          *@param lsUByte el valor del byte sin signo.
```

```
30          *@throws java.lang.NumberFormatException Arrojada por el metodo.*/
31         public static UByte valueOf(short lsUByte) throws NumberFormatException{
32                 if (lsUByte < MIN_VALUE || lsUByte > MAX_VALUE)
33                         throw new NumberFormatException(
34                                 "Valor fuera de rango. Valor:\"" + lsUByte + "\"");
35                 return UByteCache.cache[(int)lsUByte];
36         }
37         /**Crea y retorna el objeto <code>UByte</code> a partir de una cadena y una
38          *base.
39          * @param lstrUByte La cadena a convertir.
40          * @param liBase La base en la cual se representa en <code>lstrUByte</code>
41          * @return Un objeto <code>UByte</code> con el valor representado en
42          *                  la cadena bajo la base dada.
43          * @exception NumberFormatException si el objeto <code>String</code>
44          *                  no contiene un byte sin signo valido.*/
45         public static UByte valueOf(String lstrUByte, int liBase)
46                         throws NumberFormatException {
47                 int i = Integer.parseInt(lstrUByte, liBase);
48                 if (i < MIN_VALUE || i > MAX_VALUE)
49                         throw new NumberFormatException(
50                                         "Valor fuera de rango. Valor:\"" + lstrUByte +
51                                         "\" Raiz:" + liBase);
52                 return new UByte(Short.parseShort(lstrUByte, liBase));
53         }
54         /**Crea y retorna el objeto <code>UByte</code> a partir de una cadena.
55          * @param lstrUByte La cadena a convertir.
56          * @return Un objeto <code>UByte</code> con el valor representado en
57          *                  la cadena bajo la base dada.
58          * @exception NumberFormatException si el objeto <code>String</code>
59          *                  no contiene un byte sin signo valido.*/
60         public static UByte valueOf(String lstrUByte) throws NumberFormatException {
61                 return valueOf(lstrUByte, 10);
62         }
63         /**Decodifica una cadena en un <code>UByte</code>. Acepta bases 8, 10 y 16.
64          *@param lsUByte La <code>String</code> a decodificar.
65          *@return Un objeto <code>UByte</code> con el valor representado en
66          *                  la cadena.
67          *@exception NumberFormatException si el objeto <code>String</code>
68          *                  no contiene un byte sin signo valido.*/
69         public static UByte decode(String lsUByte) throws NumberFormatException {
70                 int radix = 10;
71                 int index = 0;
72                 boolean negative = false;
73                 UByte result;
74                 if (lsUByte.startsWith("-")) {
75                         throw new NumberFormatException("Valor fuera de rango. " +
76                                         "Valor negativo");
77                 }
78                 if (lsUByte.startsWith("0x", index) || lsUByte.startsWith("0X", index)){
79                         index += 2;
80                         radix = 16;
81                 } else if (lsUByte.startsWith("#", index)) {
82                         index++;
83                         radix = 16;
84                 } else if (lsUByte.startsWith("0", index) && lsUByte.length()>1+index){
85                         index++;
86                         radix = 8;
87                 }
88                 if (lsUByte.startsWith("-", index))
89                         throw new NumberFormatException("Negative sign in wrong position");
90                 try {
91                         result = UByte.valueOf(lsUByte.substring(index), radix);
92                 } catch (NumberFormatException e) {
93                         String constant = lsUByte.substring(index);
94                         result = UByte.valueOf(constant, radix);
```

```
95              }
96              return result;
97      }
98      /**Crea una nueva instancia de la clase UByte a partir del valor short
99       *(<code>lsUByte</code>).
100      *@param lsUByte Valor en <code>short</code> que sera encapsulado.
101      *@exception NumberFormatException si el valor de <code>lsUByte</code> se
102      *encuentra fuera del rango.*/
103     public UByte(short lsUByte) throws NumberFormatException{
104             if (lsUByte < MIN_VALUE || lsUByte > MAX_VALUE)
105                     throw new NumberFormatException(
106                             "Valor fuera de rango. Valor:\"" + lsUByte + "\"");
107             mUByte=lsUByte;
108     }
109     /** Crea una nueva instancia de la clase UByte a partir del valor int
110      *(<code>liUByte</code>).
111      *@param liUByte Valor en <code>int</code> que sera encapsulado.
112      *@exception NumberFormatException si el valor de <code>liUByte</code> se
113      *encuentra fuera del rango.*/
114     public UByte(int liUByte) throws NumberFormatException{
115             if (liUByte < MIN_VALUE || liUByte > MAX_VALUE)
116                     throw new NumberFormatException(
117                             "Valor fuera de rango. Valor:\"" + liUByte + "\"");
118             mUByte=(short)liUByte;
119     }
120     /** Crea una nueva instancia de la clase UByte a partir de la cadena
121      *(<code>lstrUByte</code>).
122      *@param lstrUByte Cadena que contiene el valor que sera encapsulado.
123      *@exception NumberFormatException si el valor de <code>lstrUByte</code> se
124      *encuentra fuera del rango o no es valido.*/
125     public UByte(String lstrUByte) throws NumberFormatException {
126             short tmp=Short.parseShort(lstrUByte, 10);
127             if (tmp < MIN_VALUE || tmp > MAX_VALUE)
128                     throw new NumberFormatException(
129                             "Valor fuera de rango. Valor:\"" + tmp + "\"");
130             mUByte = tmp;
131     }
132     /**Sirve para hacer una copia profunda del objeto. Sobreescribe el metodo
133      * <code>clone</code> de <code>Object</code>.
134      *@return El objeto nuevo con el mismo valor que este.*/
135     protected Object clone(){
136             return new UByte(shortValue());
137     }
138     /**Devuelve el valor en tipo <code>short</code> del <code>UByte</code>.
139      *@return El valor del <code>UByte</code> como <code>short</code>.*/
140     public short shortValue(){
141             return mUByte;
142     }
143     /**Devuelve el valor en tipo <code>int</code> del <code>UByte</code>.
144      *@return El valor del <code>UByte</code> como <code>int</code>.*/
145     public int intValue() {
146             return (int)mUByte;
147     }
148     /**Devuelve el valor en tipo <code>long</code> del <code>UByte</code>.
149      *@return El valor del <code>UByte</code> como <code>long</code>.*/
150     public long longValue() {
151             return (long)mUByte;
152     }
153     /**Devuelve el valor en tipo <code>float</code> del <code>UByte</code>.
154      *@return El valor del <code>UByte</code> como <code>float</code>.*/
155     public float floatValue() {
156             return (float)mUByte;
157     }
158     /**Devuelve el valor en tipo <code>double</code> del <code>UByte</code>.
159      *@return El valor del <code>UByte</code> como <code>double</code>.*/
```

```
160         public double doubleValue() {
161                 return (double)mUByte;
162         }
163         /**Crea un arreglo de UByte a partir de valores enteros
164          *@param liTamanio Tamanio del arreglo.
165          *@param liArreglo Valores que debe contener el arreglo final.
166          *@return El nuevo arreglo de UByte*/
167         public static UByte[] creaArray(int liTamanio,int[] liArreglo){
168                 UByte lUBytes[]=new UByte[liTamanio];
169                 for(int i=0;i<liTamanio;lUBytes[i]=new UByte(liArreglo[i++]));
170                 return lUBytes;
171         }
172         /**Obtiene un objeto nuevo <code>String</code> que representa al
173          *<code>UByte</code>.
174          * @return La cadena que representa al <code>UByte</code>*/
175         public String toString() {
176                 return String.valueOf((int)mUByte);
177         }
178         /**Regresa el valor Hash para este objeto UByte
179          *@return El codigo hash del objeto.*/
180         public int hashCode() {
181                 return (int)mUByte;
182         }
183         /**Compara este objeto con <code>lObjeto</code>, en caso de ser otro
184          *<code>UByte</code>
185          * utiliza el valor short para la comparacion.
186          * @param lObjeto El objeto con el cual compararse.
187          * @return <code>true</code> si los objetos son iguales;
188          *                      <code>false</code> de lo contrario.*/
189         public boolean equals(Object lObjeto) {
190                 if (lObjeto instanceof UByte) {
191                         return mUByte == ((UByte)lObjeto).shortValue();
192                 }
193                 return false;
194         }
195         /**Compara este objeto con <code>lOtroUByte</code> utilizando el valor short
196          * para la comparacion.
197          * @param lOtroUByte El otro <code>UByte</code> con el cual compararse.
198          * @return <code>0</code> si los dos <code>UByte</code>'s son iguales; un
199          *       valor mayor a <code>0</code> si este es mayor que el otro y un
200          *       valor menor a <code>0</code> en otro caso.*/
201         public int compareTo(UByte lOtroUByte) {
202                 return mUByte-lOtroUByte.mUByte;
203         }
204 }
```

4.3.2. Clase mx.cic.ipn.utilerias.Familia

```
1  package mx.ipn.cic.utilerias;
2  import mx.ipn.cic.excepciones.FueraDeRangoException;
3  /**Esta clase abstrae el concepto de una Familia de modulos. Cada elemento de
4   * hardware contiene un identificador (ID) que debe ser considerado para varios
5   * propositos en el disenio de un prototipo. La Familia o ID esta compuesto por
6   * el fabricante del modulo (3 bits) concatenado con un numero identificador de
7   * tipo (5 bits) del mismo.
8   * De esta manera, un modulo con ID 30(decimal) indica que es de tipo 30 y de
9   * fabricante 0, mientras que un ID 94(decimal) es del mismo tipo 30 pero del
10  * fabricante 3.*/
11 public class Familia implements Comparable<Familia>{
12         /**Fabricante del modulo.*/
13         private UByte mFabricante;
14         /**Tipo o familia de modulo.*/
15         private UByte mTipo;
16         /**Crea una nueva instancia de la clase Familia mediante un numero de
17          *familia.
```

```
18          *@param lFamilia Es el valor concatenado de fabricante y tipo.*/
19         public Familia(UByte lFamilia){
20                 setFamilia(lFamilia);
21         }
22         /**Crea una nueva instancia de la clase Familia mediante el fabricante y el
23          *tipo.
24          *@param lFabricante Es el fabricante del modulo.
25          *@param lTipo Es el tipo de modulo.
26          *@exception FueraDeRangoException si <code>lFabricante</code> o si el
27          *              <code>lTipo</code> se encuentra fuera del rango establecido.*/
28         public Familia(UByte lFabricante, UByte lTipo) throws FueraDeRangoException{
29                 setFabricante(lFabricante);
30                 setTipo(lTipo);
31         }
32         /**Regresa el fabricante de la familia.
33          *@return El fabricante de la familia.*/
34         public UByte getFabricance(){
35                 return mFabricante;
36         }
37         /**Regresa el tipo de familia.
38          *@return El tipo de familia.*/
39         public UByte getTipo(){
40                 return mTipo;
41         }
42         /**Regresa la familia del modulo en forma de UByte
43          *@return La familia del modulo en un valor de tipo UByte.*/
44         public UByte getFamilia(){
45                 return (UByte)new UByte(((mFabricante.intValue()<<5)|mTipo.intValue()));
46         }
47         /**Actualiza el valor del fabricante del modulo.
48          *@param lFabricante Nuevo fabricante del modulo.
49          *@exception FueraDeRangoException si <code>lFabricante</code> se encuentra
50          *              fuera del rango establecido.*/
51         private void setFabricante(UByte lFabricante) throws FueraDeRangoException{
52                 if(lFabricante.shortValue()<0 || lFabricante.shortValue()>7)
53                         throw new FueraDeRangoException("Fabricante␣("+lFabricante+
54                                 ")␣fuera␣de␣rango.");
55                 mFabricante=lFabricante;
56         }
57         /**Actualiza el valor del tipo de modulo.
58          *@param lTipo Nuevo tipo de modulo.*/
59         private void setTipo(UByte lTipo) {
60                 if(lTipo.shortValue()<0 || lTipo.shortValue()>7)
61                         throw new FueraDeRangoException("Tipo␣("+lTipo+")␣fuera␣de␣rango.");
62                 mTipo=lTipo;
63         }
64         /** Actualiza la familia mediante un numero de familia.
65          *@param lFamilia Es el valor concatenado de fabricante y tipo.*/
66         private void setFamilia(UByte lFamilia){
67                 int ltmp=0;
68                 ltmp=lFamilia.intValue() & 31;
69                 mTipo=new UByte(ltmp);
70                 mFabricante=new UByte((lFamilia.intValue()&224)>>5);
71         }
72         /**Representacion textual de la clase.
73          *@return Una cadena con el formato: FabricanteTipo*/
74         public String toString() {
75                 return ""+mFabricante+mTipo;
76         }
77         /**Compara este objeto con <code>lObjeto</code>, en caso de ser otra
78          *<code>Familia</code> utiliza como valor de comparacion la familia
79          *(concatenacion de fabricante y tipo).
80          * @param lObjeto El objeto con el cual compararse.
81          * @return <code>true</code> si los objetos son iguales;
82          *                      <code>false</code> de lo contrario.*/
```

```
83          public boolean equals(Object lObjeto) {
84                  if (lObjeto instanceof Familia)
85                          return hashCode() == lObjeto.hashCode();
86                  return false;
87          }
88          /**Sirve para hacer una copia profunda del objeto. Sobreescribe el metodo
89           * <code>clone</code> de <code>Object</code>.
90           *@return El objeto nuevo con el mismo valor que este.*/
91          protected Object clone(){
92                  return new Familia(getFamilia());
93          }
94          /** Compara una Familia con otra.
95           *@param lOtraFamilia Es otra familia con la cual compararse.
96           *@return El valor <code>0</code> si ambas familias son la misma (mismo
97           *       fabricante y mismo tipo), un valor menor a <code>0</code> si esta
98           *       familia es menor a otra en por lo menos uno de los valores; y un
99           *       valor mayor a <code>0</code> si esta familia es mayor a la otra en
100          *       por lo menos uno de los valores.*/
101         public int compareTo(Familia lOtraFamilia){
102                 return this.getFamilia().compareTo(lOtraFamilia.getFamilia());
103         }
104         /**Codigo hash del objeto es la familia.
105          *@return El codigo hash del objeto.*/
106         public int hashCode(){
107                 return getFamilia().intValue();
108         }
109 }
```

4.3.3. Clase mx.cic.ipn.utilerias.Mensaje

```
1  package mx.ipn.cic.utilerias;
2  import mx.ipn.cic.excepciones.FueraDeRangoException;
3  /**Esta clase abstrae el concepto de mensaje para el estandar de comunicacion.
4   * Proporciona un medio de almacenamiento de los mensajes como concepto
5   * principal de comunicacion entre el maestro y los modulos instanciados con
6   * la finalidad de encapsular todos los datos del mismo.
7   * Un mensaje esta compuesto por los siguientes elementos obligatorios:
8   * Un byte de Destinatario(<code>mDestino</code>)
9   * Un byte de Tamanio(<code>mTamanio</code>)
10  * Un byte de Instruccion(<code>mInstruccion</code>)
11  * Aunado a dichos elementos se cuenta con un arreglo de maximo 13 bytes
12  * destinados para los parametros. En total se cuenta con mensajes desde
13  * 3 bytes y hasta 16 bytes.
14  *El Destinatario(<code>mDestino</code>) tiene un rango de entre 0 y hasta 255
15  *que es el total de modulos conectados mediante el estandar. El Tamanio
16  *(<code>mTamanio</code>) es el atributo que indica la cantidad de parametros
17  *que contiene el mensaje, pueden ser ninguno y hasta 13. El byte de Instruccion
18  *(<code>mInstruccion</code>) almacena un valor entre 0 y 255, que es la
19  *instruccion que debe ejecutar el modulo, los primeros doce valores (0-11) son
20  *instrucciones definidas para todos los modulos y especificadas en el estandar.
21  * Indistintamente se utiliza el termino byte para designar un valor de 8 bits
22  * de longitud sin signo, para almacenarlo se utiliza el tipo wrapper UByte.
23  * Las caracteristicas de los bytes descritos son validadas en esta clase y se
24  * arrojan excepciones considerando errores de rangos.
25  * @see      mx.ipn.cic.excepciones.FueraDeRangoException
26  * @see      mx.ipn.cic.utilerias.UByte*/
27 public class Mensaje implements Comparable<Mensaje>{
28         /**Valor entre 0 y 255 que es el modulo de destino del mensaje. Cuando el
29         * mensaje es recibido, este valor debera ser cero.*/
30         protected UByte mDestino;
31         /**Valor entre 0 y 13, es la cantidad de parametros contenidos en el
32         * mensaje.*/
33         protected UByte mTamanio;
34         /**Valor entre 0 y 255, es la instruccion del mensaje.*/
35         protected UByte mInstruccion;
```

```
36          /**Son los parametros del mensaje, entre 0 y 13 parametros, valores entre
37          * 0 y 255 por cada uno.*/
38          protected UByte[] mParametros;
39          /** Crea una nueva instancia de la clase Mensaje, un mensaje vacio.*/
40          public Mensaje() {
41                  mDestino=null;
42                  mTamanio=null;
43                  mInstruccion=null;
44                  mParametros=null;
45          }
46          /** Crea una nueva instancia de la clase Mensaje, con destinatario
47           * <code>lDestino</code>, tamanio cero, instruccion <code>lInstruccion</code>
48           * y ningun parametro.
49           *@param lDestino Destinatario del mensaje.
50           *@param lInstruccion Instruccion del mensaje.*/
51          public Mensaje(UByte lDestino, UByte lInstruccion){
52                  mDestino=lDestino;
53                  mTamanio=null;
54                  mInstruccion=lInstruccion;
55                  mParametros=null;
56          }
57          /** Crea una nueva instancia de la clase Mensaje, con destinatario
58           * <code>lDestino</code>, tamanio <code>lTamanio</code>,
59           * instruccion <code>lInstruccion</code> y los <code>lTamanio</code>
60           * parametros en cero.
61           *@param lDestino Destinatario del mensaje.
62           *@param lTamanio Cantidad de parametros del mensaje.
63           *@param lInstruccion Instruccion del mensaje.*/
64          public Mensaje(UByte lDestino, UByte lTamanio, UByte lInstruccion){
65                  mDestino=lDestino;
66                  mTamanio=lTamanio;
67                  mInstruccion=lInstruccion;
68                  mParametros=new UByte[mTamanio.intValue()];
69                  for(int i=0;i<mTamanio.intValue();mParametros[i++]=new UByte(0));
70          }
71          /**Crea una nueva instancia de la clase Mensaje, con destinatario
72           *<code>lDestino</code>, tamanio <code>lTamanio</code>,
73           *instruccion <code>lInstruccion</code> y un parametro <code>lParametro
74           *</code>.
75           *@param lDestino Destinatario del mensaje.
76           *@param lTamanio Cantidad de parametros del mensaje.
77           *@param lInstruccion Instruccion del mensaje.
78           *@param lParametro Parametro del mensaje.*/
79          public Mensaje(UByte lDestino, UByte lTamanio, UByte lInstruccion,
80                  UByte lParametro){
81                  mDestino=lDestino;
82                  mTamanio=lTamanio;
83                  mInstruccion=lInstruccion;
84                  mParametros=new UByte[1];
85                  mParametros[0]=lParametro;
86          }
87          /** Crea una nueva instancia de la clase Mensaje, con destinatario
88           * <code>lDestino</code>, tamanio <code>lTamanio</code>,
89           * instruccion <code>lInstruccion</code> y los parametros <code>lParametros
90           * </code>.
91           *@param lDestino Destinatario del mensaje.
92           *@param lTamanio Cantidad de parametros del mensaje.
93           *@param lInstruccion Instruccion del mensaje.
94           *@param lParametros Parametros del mensaje.
95           *@exception FueraDeRangoException si <code>lTamanio</code> es mayor al
96           *                tamanio del arreglo <code>lParametros</code>.*/
97          public Mensaje(UByte lDestino, UByte lTamanio, UByte lInstruccion,
98                          UByte[] lParametros)
99                          throws FueraDeRangoException {
100                 if(lTamanio.intValue()>lParametros.length)
```

```
101                     throw new FueraDeRangoException(
102                             "Faltan parametros o el tamanio es "+"incorrecto.");
103             mDestino=lDestino;
104             mTamanio=lTamanio;
105             mInstruccion=lInstruccion;
106             mParametros=new UByte[mTamanio.intValue()];
107             for(int i=0;i<mTamanio.intValue();mParametros[i]=lParametros[i++]);
108         }
109         /**Actualiza el destino del mensaje
110          *@param lDestino Nuevo destino del mensaje.*/
111         public void setDestino(UByte lDestino) {
112             mDestino=lDestino;
113         }
114         /**Actualiza el tamanio del mensaje
115          *@param lTamanio Nuevo tamanio del mensaje.*/
116         public void setTamanio(UByte lTamanio) {
117             mTamanio=lTamanio;
118         }
119         /**Actualiza la instruccion del mensaje
120          *@param lInstruccion Nueva instruccion del mensaje.*/
121         public void setInstruccion(UByte lInstruccion) {
122             mInstruccion=lInstruccion;
123         }
124         /**Actualiza los parametros del mensaje
125          *@param lParametros Nuevos parametros del mensaje.
126          *@exception FueraDeRangoException si <code>lTamanio</code> es mayor al
127          *              tamanio del arreglo <code>lParametros</code>.*/
128         public void setParametros(UByte[] lParametros) throws FueraDeRangoException{
129             if(mTamanio.intValue()>lParametros.length){
130                     throw new FueraDeRangoException(
131                             "Faltan parametros o el tamanio es "+"incorrecto.");
132             }
133             for(int i=0;i<mTamanio.intValue();mParametros[i]=lParametros[i++]);
134         }
135         /**Actualiza el <code>lIndice</code> parametro del mensaje
136          *@param liIndice Indice por actualizar.
137          *@param lParametro Nuevo parametro del mensaje.
138          *@exception FueraDeRangoException si <code>lIndice</code> se encuentra
139          *              fuera del rango establecido por la cantidad de parametros en
140          *              el arreglo <code>lParametros</code>.*/
141         public void setParametro(int liIndice, UByte lParametro)
142                     throws FueraDeRangoException {
143             if(liIndice>mParametros.length){
144                     throw new FueraDeRangoException("Indice ("+liIndice+
145                             ") fuera de rango");
146             }
147             mParametros[liIndice]=lParametro;
148         }
149         /**Regresa el destino del mensaje
150          *@return Destino del mensaje.*/
151         public UByte getDestino(){
152             return mDestino;
153         }
154         /**Regresa el tamanio del mensaje
155          *@return Tamanio del mensaje.*/
156         public UByte getTamanio(){
157             return mTamanio;
158         }
159         /**Regresa la instruccion del mensaje
160          *@return Instruccion del mensaje.*/
161         public UByte getInstruccion(){
162             return mInstruccion;
163         }
164         /**Regresa los parametros del mensaje
165          *@return Clon de los parametros del mensaje.*/
```

```
166         public UByte[] getParametros(){
167                 return mParametros.clone();
168         }
169         /**Regresa el <code>lIndice</code> parametro del mensaje
170          *@param liIndice Indice del arreglo de parametros.
171          *@return El parametro <code>lIndice</code> del arreglo <code>lParametros
172          *       </code>.
173          *@exception FueraDeRangoException si <code>lIndice</code> se encuentra
174          *       fuera del rango establecido por la cantidad de parametros en el
175          *       arreglo <code>lParametros</code>.*/
176         public UByte getParametro(int liIndice) throws FueraDeRangoException{
177                 if(liIndice>mParametros.length){
178                         throw new FueraDeRangoException("Indice ("+liIndice+
179                                 ") fuera de rango");
180                 }
181                 return mParametros[liIndice];
182         }
183         /**Compara este objeto con <code>lObjeto</code>, en caso de ser otro <code>
184          *Mensaje</code> utiliza <code>compareTo</code> para comparar los objetos.
185          * @param lObjeto El objeto con el cual compararse.
186          * @return <code>true</code> si los objetos son iguales;
187          *                      <code>false</code> de lo contrario.
188          *@see mx.ipn.cic.utilerias.Mensaje#compareTo(Mensaje)*/
189         public boolean equals(Object lObjeto) {
190                 if (lObjeto instanceof Mensaje) {
191                         return compareTo((Mensaje)lObjeto)==0?true:false;
192                 }
193                 return false;
194         }
195         /** Compara un Mensaje con otro.
196          *@param lOtroMensaje Es otro mensaje con el cual compararse.
197          *@return el valore <code>0</code> si ambos mensajes son iguales, byte a
198          *       byte; un valor menor a <code>0</code> si este mensaje es menor al
199          *       otro en al menos un byte; y un valor mayor a <code>0</code> si este
200          *       mensaje es mayor al otro en al menos un byte.*/
201         public int compareTo(Mensaje lOtroMensaje){
202                 int li=0;
203                 if((li=mDestino.compareTo(lOtroMensaje.mDestino))!=0) return li;
204                 if((li=mTamanio.compareTo(lOtroMensaje.mTamanio))!=0) return li;
205                 if((li=mInstruccion.compareTo(lOtroMensaje.mInstruccion))!=0) return li;
206                 if(mTamanio.intValue()!=0){
207                         for(int i=0;i<mTamanio.intValue();i++){
208                                 if(
209                                 (li=mParametros[i].compareTo(lOtroMensaje.mParametros[i]))
210                                 !=0)
211                                 return li;
212                         }
213                 }
214                 return 0;
215         }
216         /**Sirve para hacer una copia profunda del objeto. Sobreescribe el metodo
217          * <code>clone</code> de <code>Object</code>.
218          *@return El objeto nuevo con los mismos valores que este.*/
219         protected Object clone(){
220                 if(mParametros!=null){
221                         UByte lArray[]= new UByte[mParametros.length];
222                         for(int i=0;i<mParametros.length;lArray[i]=mParametros[i++]);
223                         return new Mensaje(mDestino,mTamanio,mInstruccion,lArray);
224                 }
225                 return new Mensaje(mDestino,mTamanio,mInstruccion);
226         }
227         /**Representacion textual de la clase.
228          *@return Una cadena con el formato: Destinatario:Tamanio:Instruccion:
229          *           Parametro1:Parametro2.*/
230         public String toString() {
```

```
231                 String lstrTempo=mDestino+":"+mTamanio+":"+mInstruccion;
232                 if(mTamanio.intValue()!=0){
233                         for(int i=0;i<mTamanio.intValue();lstrTempo+=":"+mParametros[i++]);
234                 }
235                 return lstrTempo;
236         }
237         /**Convierte el mensaje a un arreglo de enteros.
238          *@return El arreglo de <code>int</code> conteniendo el mensaje.*/
239         public int[] toIntArray(){
240                 int tmp[]=new int[mTamanio.intValue()+3];
241                 tmp[0]=mDestino.intValue();
242                 tmp[1]=mTamanio.intValue();
243                 tmp[2]=mInstruccion.intValue();
244                 for(int i=3;i<tmp.length;i++){
245                         tmp[i]=mParametros[i-3].intValue();
246                 }
247                 return tmp;
248         }
249 }
```

4.3.4. Clase mx.cic.ipn.utilerias.RegistroActividades

```
1  package mx.ipn.cic.utilerias;
2  import java.io.*;
3  import java.text.DateFormat;
4  import java.util.Date;
5  import java.util.Vector;
6  import mx.ipn.cic.modulos.Modulo;
7  /**Clase encargada de registrar las actividades de los modulos.*/
8  public class RegistroActividades {
9          /**Flujo de datos de salida al archivo.*/
10         private DataOutputStream mSalida;
11         /**Vector que almacena una cantidad de entradas (100) para ser escritas en
12         *disco al llenarse o al concluir el programa.*/
13         private Vector<String> mvecBuffer;
14         /**Es el tamanio del buffer de almacenamiento en memoria.*/
15         private int miBufferSize;
16         /**Crea una nueva instancia de la clase RegistroActividades con un archivo
17          * especifico.
18          * @param lPathArchivoRegistro Cadena que contiene la ruta completa del
19          *          archivo de registro.*/
20         public RegistroActividades(String lPathArchivoRegistro){
21                 FileOutputStream lfos=null;
22                 try{
23                         lfos = new FileOutputStream(lPathArchivoRegistro, true);
24                 } catch(FileNotFoundException fnfe){
25                         System.err.println("Archivo no encontrado");
26                         fnfe.printStackTrace(System.err);
27                 } catch(SecurityException se){
28                         System.err.println("No es posible abrir el archivo por cuestiones "+
29                                 "de seguridad");
30                         se.printStackTrace(System.err);
31                 }
32                 mSalida=new DataOutputStream(lfos);
33                 miBufferSize=100;
34                 mvecBuffer=new Vector<String>(miBufferSize+1);
35         }
36         /**Registra una actividad. El formato es:<p>
37          *dd/MM/AAAA hh:mm:ss AM/PM,TX/RX, ClaseModulo, Familia, Posicion, Mensaje
38          *@param lModulo Es el modulo que realiza la accion.
39          *@param lbTransmision <code>true</code> si el mensaje fue transmitido,
40          *<code>false</code> de lo contrario.
41          *@param lMensaje El mensaje transmitido o recibido.*/
42         public void registra(Modulo lModulo, boolean lbTransmision,
43                         Mensaje lMensaje){
```

```
44                  String lstrTempo=null;
45                  if(lModulo==null){
46                          lstrTempo= DateFormat.getDateTimeInstance().format(
47                                  new Date(System.currentTimeMillis()))+",“+
48                                  (lbTransmision==true?"TX,":"RX,")+"Control"+
49                                  ",DESC,DESC,"+lMensaje+"\n";
50                  }
51                  else {
52                          lstrTempo= DateFormat.getDateTimeInstance().format(
53                                  new Date(System.currentTimeMillis()))+",“+
54                                  (lbTransmision==true?"TX,":"RX,")+lModulo.getClass()+
55                                  ","+lModulo.getFamilia()+","+lModulo.getPosicion()+","+
56                                  lMensaje+"\n";
57                  }
58                  mvecBuffer.add(lstrTempo);
59                  if(mvecBuffer.size()>=miBufferSize){
60                          flush();
61                  }
62          }
63          /**Registra una actividad. El formato es:<p>
64           *dd/MM/AAAA hh:mm:ss AM/PM, Cadena
65           *@param lstrCadena Texto a agregar en el registro.*/
66          public void registra(String lstrCadena){
67                  mvecBuffer.add(DateFormat.getDateTimeInstance().format(
68                          new Date(System.currentTimeMillis()))+",“+lstrCadena+"\n");
69                  if(mvecBuffer.size()>=miBufferSize){
70                          flush();
71                  }
72          }
73          /**Escribe en el archivo.*/
74          private void flush(){
75                  for(String lstrTempo:mvecBuffer){
76                          try {
77                                  mSalida.writeBytes(lstrTempo);
78                          } catch (IOException ex) {
79                                  System.err.println("No es posible escribir en el archivo");
80                                  ex.printStackTrace(System.err);
81                          }
82                  }
83                  mvecBuffer.removeAllElements();
84          }
85          /**Actualiza el tamanio del buffer de almacenamiento.
86           *@param liBufferSize Es el nuevo tamanio del buffer.*/
87          public void setBufferSize(int liBufferSize){
88                  miBufferSize=liBufferSize;
89                  Vector<String> lvecTempo = new Vector<String>(miBufferSize+1);
90                  lvecTempo.addAll(mvecBuffer);
91                  mvecBuffer=lvecTempo;
92          }
93          /**Regresa el tamanio del buffer de almacenamiento.
94           *@return El tamanio del buffer.*/
95          public int getBufferSize(){
96                  return miBufferSize;
97          }
98          /**Antes de eliminarse el objeto, se debe escribir lo contenido en el
99           *buffer.
100          *@throws java.lang.Throwable Arrojada por el metodo.*/
101         protected void finalize() throws Throwable {
102                 flush();
103         }
104 }
```

4.3.5. Clase mx.cic.ipn.utilerias.Contenedor

```
1 package mx.ipn.cic.utilerias;
```

```
2  import java.util.AbstractList;
3  import mx.ipn.cic.controladores.Control;
4  import mx.ipn.cic.modulos.Modulo;
5  /**Clase que almacena los modulos y es la clase abstracta padre de cualquier
6   * controlador.
7   * @see  mx.ipn.cic.controladores.ConstructorControl*/
8  public class Contenedor {
9          /**Es el conjunto de modulos.*/
10         private AbstractList<Modulo> mModulos;
11         /**Es el conjunto de modulos.*/
12         private AbstractList<Control> mControladores;
13         /**Crea una nueva instancia de la clase Contenedor con la coleccion dada.
14          * Realiza una copia superficial (asignacion) de la lista.
15          * @param lControladores Es la lista de controladores.
16          * @param lModulos Es la lista o conjunto de modulos.*/
17         public Contenedor(AbstractList<Control> lControladores,
18                           AbstractList<Modulo> lModulos) {
19                 mModulos=lModulos;
20                 mControladores=lControladores;
21         }
22         /**Inicializa el control.
23          *@param lControl Control a contener.
24          *@return <code>true</code> si el contenedor se modifico despues de la
25          *          llamada, <code>false</code> de lo contrario.*/
26         public boolean addControl(Control lControl){
27                 return mControladores.add(lControl);
28         }
29         /**Regresa el control contenido.
30          * @param liIndice Indice del Control a regresar.
31          * @return El control solicitado.*/
32         public Control getControl(int liIndice){
33                 return mControladores.get(liIndice);
34         }
35         /**Agrega un modulo al contenedor.
36          *@param lModulo Modulo por agregar.
37          *@return <code>true</code> si el contenedor se modifico despues de la
38          *          llamada, <code>false</code> de lo contrario.*/
39         public boolean addModulo(Modulo lModulo){
40                 return mModulos.add(lModulo);
41         }
42         /**Regresa el modulo ubicado en la "posicion" <code>liIndice</code>, sin
43          *eliminarlo de la lista.
44          *@param liIndice Es la "posicion" mantenida por el modulo. Depende del tipo
45          *       de lista implementada.
46          *@return La referencia del Modulo solicitado.*/
47         public Modulo getModulo(int liIndice){
48                 return mModulos.get(liIndice);
49         }
50         /**Regresa la cantidad de modulos contenidos.
51          *@return La cantidad de modulos contenidos.*/
52         public int getCantidadModulos(){
53                 return mModulos.size();
54         }
55         /**Regresa la cantidad de modulos contenidos.
56          *@return La cantidad de modulos contenidos.*/
57         public int getCantidadControladores(){
58                 return mControladores.size();
59         }
60         /**Retira el modulo ubicado en la "posicion" <code>liIndice</code> del
61          *contenedor.
62          *@param liIndice "Posicion" del modulo por retirar.
63          *@return El modulo retirado.*/
64         public Modulo removeModulo(int liIndice){
65                 return mModulos.remove(liIndice);
66         }
```

```
67          /**Retira el modulo especificado
68           *@param lModulo Es el modulo que se desea eliminar del contenedor.
69           *@return <code>true</code> si el contenedor se modifico despues de la
70           *       llamada, <code>false</code> de lo contrario.*/
71          public boolean removeModulo(Modulo lModulo){
72                  return mModulos.remove(lModulo);
73          }
74          /**Retira el control ubicado en la "posicion" <code>liIndice</code> del
75           *contenedor.
76           *@param liIndice "Posicion" del control por retirar.
77           *@return El control retirado.*/
78          public Control removeControl(int liIndice){
79                  return  mControladores.remove(liIndice);
80          }
81          /**Retira el Control especificado
82           *@param lControl Es el Control que se desea eliminar del contenedor.
83           *@return <code>true</code> si el contenedor se modifico despues de la
84           *       llamada, <code>false</code> de lo contrario.*/
85          public boolean removeModulo(Control lControl){
86                  return mModulos.remove(lControl);
87          }
88          /**Representacion textual de la clase.
89           *@return Una cadena con el formato: TotalModulos\nModulo 1\n Modulo 2.*/
90          public String toString() {
91                  String lstrRet="Total de Modulos Contenidos: "+mModulos.size()+"\n";
92                  for(Modulo lModulo:mModulos){
93                          lstrRet+=""+lModulo+"\n";
94                  }
95                  return lstrRet;
96          }
97  }
```

4.4. mx.cic.ipn.comunicaciones

4.4.1. Clase mx.cic.ipn.comunicaciones.MecanismoComunicacion

```
1  package mx.ipn.cic.comunicaciones;
2  import mx.ipn.cic.utilerias.Mensaje;
3  import java.util.AbstractQueue;
4  import javax.swing.event.EventListenerList;
5  /** En esta clase se definen las operaciones que debe cubrir un sistema de
6   *comunicacion dentro del Estandar. Con ello sera posible la extension del mismo
7   *a distintos mecanismos o vias de comunicacion.
8   *Internamente, se almacenan los mensajes recibidos en una cola abstracta.
9   *@see java.util.AbstractQueue
10  *@see javax.swing.event.EventListenerList*/
11 public abstract class MecanismoComunicacion implements Runnable{
12          /**Lista ligada (FIFO) de mensajes*/
13          protected AbstractQueue<Mensaje> mMensajes;
14          /**Lista de EventListeners.*/
15          protected EventListenerList listenerList = new EventListenerList();
16          /** Crea una nueva instancia de la clase MecanismoComunicacion. Coloca
17           *<code>mMensajes</code> a <code>null</code>*/
18          public MecanismoComunicacion(){
19                  mMensajes=null;
20          }
21          /** Crea una nueva instancia de la clase MecanismoComunicacion. Asigna
22           *<code>lMensajes</code> a <code>mMensajes</code>, sin clonar.
23           *@param lMensajes La cola que debe contener la clase.*/
24          public MecanismoComunicacion(AbstractQueue<Mensaje> lMensajes){
25                  mMensajes=lMensajes;
26          }
27          /**Transmite el <code>lMensaje</code> por el canal. Las clases que heredan
28           *de esta, deben administrar la forma en la cual envian los datos por el
```

```
29          *medio. No deben regresar el control hasta lograr la transmision
30          *solicitada.
31          *@param lMensaje Es el mensaje a transmitir.
32          *@return <code>true</code> si fue posible la transmision,
33          *       <code>false</code> de no ser posible por cualquier razon.*/
34          public abstract boolean transmiteMensaje(Mensaje lMensaje);
35          /**Obtener el siguiente mensaje de una cola de mensajes recibidos.
36          *@return El siguiente mensaje almacenado en la cola, <code>null</code> si
37          *              la cola se encuentra vacia*/
38          public abstract Mensaje recibeMensaje();
39          /**Finaliza el objeto. Se debe implementar para cerrar o salirse de los
40          *mecanismos de comunicacion abiertos.*/
41          public abstract void finalize();
42          /**Agrega <code>lListener</code> a los listeners de la clase.
43          *@param lListener Es el nuevo listener de la clase
44          *          <code>MensajeRecibidoListener</code> por agregar.*/
45          public void addMensajeRecibidoListener(MensajeRecibidoListener lListener) {
46                  listenerList.add(MensajeRecibidoListener.class, lListener);
47          }
48          /**Elimina <code>lListener</code> de los listeners de la clase.
49          *@param lListener Es el listener de la clase <code>MensajeRecibidoListener
50          *          </code> por eliminar.*/
51          public void removeMensajeRecibidoListener(MensajeRecibidoListener lListener){
52                  listenerList.remove(MensajeRecibidoListener.class, lListener);
53          }
54          /**Dispara el evento <code>mensajeRecibidoEvent</code> de listener.
55          *@param lEvento Evento que debe acompaniar la llamada.
56          *@see mx.ipn.cic.comunicaciones.MensajeRecibidoListener#mensajeRecibido
57          *       (MensajeRecibidoEvent)*/
58          public void fireMensajeRecibidoEvent(MensajeRecibidoEvent lEvento) {
59                  Object[] listeners = listenerList.getListenerList();
60                  for (int i=0; i<listeners.length; i+=2) {
61                          if (listeners[i]==MensajeRecibidoListener.class) {
62                                  if (lEvento == null)
63                                          lEvento = new MensajeRecibidoEvent(this,
64                                  MensajeRecibidoEvent.MENSAJE_DISPONIBLE);
65                                  ((MensajeRecibidoListener)listeners[i+1]).mensajeRecibido(
66                                          lEvento);
67                          }
68                  }
69          }
70  }
```

4.4.2. Clase mx.cic.ipn.comunicaciones.MensajeRecibidoListener

```
1  package mx.ipn.cic.comunicaciones;
2  import java.util.EventListener;
3  /**Interface con la cual definir las clases que reciban eventos de tipo
4   * mensajeRecibido*/
5  public interface MensajeRecibidoListener extends EventListener {
6          /*Metodo con el cual se recupera un evento de la clase
7           * @param lEvento El evento ocurrido.*/
8          public void mensajeRecibido(MensajeRecibidoEvent lEvento);
9  }
```

4.4.3. Clase mx.cic.ipn.comunicaciones.MensajeRecibidoEvent

```
1  package mx.ipn.cic.comunicaciones;
2  import java.util.EventObject;
3  /**Evento arrojado cuando un mensaje completo es recibido.*/
4  public class MensajeRecibidoEvent extends EventObject {
5          /**Fuente del evento.*/
6          private Object mSource;
7          /**Estado actual del mensaje.*/
```

```
8           private int mEstado;
9           /**Existe un mensaje en espera.*/
10          public static final int MENSAJE_DISPONIBLE=1;
11          /** Crea una nueva instancia de la clase MensajeRecibidoEvent
12           *@param source El objeto fuente del evento.
13           *@param lEstado Es el estado actual del mensaje.*/
14          public MensajeRecibidoEvent(Object source, int lEstado) {
15                  super(source);
16                  mSource=source;
17                  mEstado=lEstado;
18          }
19          /**Regresa la fuente del evento.
20           *@return La fuente del evento.*/
21          public Object getSource(){
22                  return mSource;
23          }
24          /**Regresa el estado del mensaje.
25           *@return El estado del mensaje.*/
26          public int getEstado(){
27                  return mEstado;
28          }
29  }
```

4.5. mx.cic.ipn.excepciones

4.5.1. Clase mx.cic.ipn.excepciones.ImposibleInicializarMedioException

```
1  package mx.ipn.cic.excepciones;
2  /**Excepcion que es arrojada al momento de intentar inicializar un medio de
3   * comunicacion.*/
4  public class ImposibleInicializarMedioException extends Exception {
5           /**Crea una nueva instancia de la clase ImposibleInicializarMedioException*/
6           public ImposibleInicializarMedioException() {
7                   super();
8           }
9           /** Crea una nueva instancia de la clase ImposibleInicializarMedioException
10           *con un mensaje conteniendo el detalle del error.
11           *@param lstrMensajeError Mensaje con detalle del error.*/
12          public ImposibleInicializarMedioException(String lstrMensajeError) {
13                  super(lstrMensajeError);
14          }
15          /** Crea una nueva instancia de la clase ImposibleInicializarMedioException
16           *con un mensaje conteniendo el detalle y la causa del error.
17           *@param lstrMensajeError Mensaje con detalle del error.
18           *@param lCausa Causa del error.*/
19          public ImposibleInicializarMedioException(String lstrMensajeError,
20                          Throwable lCausa) {
21                  super(lstrMensajeError,lCausa);
22          }
23          /** Crea una nueva instancia de la clase ImposibleInicializarMedioException
24           * con la causa del error.
25           *@param lCausa Causa del error.*/
26          public ImposibleInicializarMedioException(Throwable lCausa) {
27                  super(lCausa);
28          }
29  }
```

4.5.2. Clase mx.cic.ipn.excepciones.FueraDeRangoException

```
1  package mx.ipn.cic.excepciones;
2  /**Es arrojada cuando un valor solicitado del UByte es menor que cero (negativo)
3   *o es mayor a un limite especificado.*/
4  public class FueraDeRangoException extends RuntimeException {
```

```
5           /** Crea una nueva instancia de la clase FueraDeRangoException */
6           public FueraDeRangoException() {}
7           /** Crea una nueva instancia de la clase FueraDeRangoException con un
8            *mensaje conteniendo el detalle del error.
9            *@param lstrMensajeError Mensaje con detalle del error.*/
10          public FueraDeRangoException(String lstrMensajeError) {
11                  super(lstrMensajeError);
12          }
13          /** Crea una nueva instancia de la clase FueraDeRangoException con un
14           *mensaje conteniendo el detalle y la causa del error.
15           *@param lstrMensajeError Mensaje con detalle del error.
16           *@param lCausa Causa del error.*/
17          public FueraDeRangoException(String lstrMensajeError, Throwable lCausa) {
18                  super(lstrMensajeError,lCausa);
19          }
20          /** Crea una nueva instancia de la clase FueraDeRangoException con la
21           * causa del error.
22           *@param lCausa Causa del error.*/
23          public FueraDeRangoException(Throwable lCausa) {
24                  super(lCausa);
25          }
26  }
```

5. Extensión del Framework

Para utilizar el framework directamente sobre módulos reales, es necesaria la implementación de algunas clases concretas.

La primera fase es dotar al framework de las clases concretas con las cuales sea posible controlar los módulos, los diagramas de dichas clases se muestran en la figura 10 y son las tres siguientes:

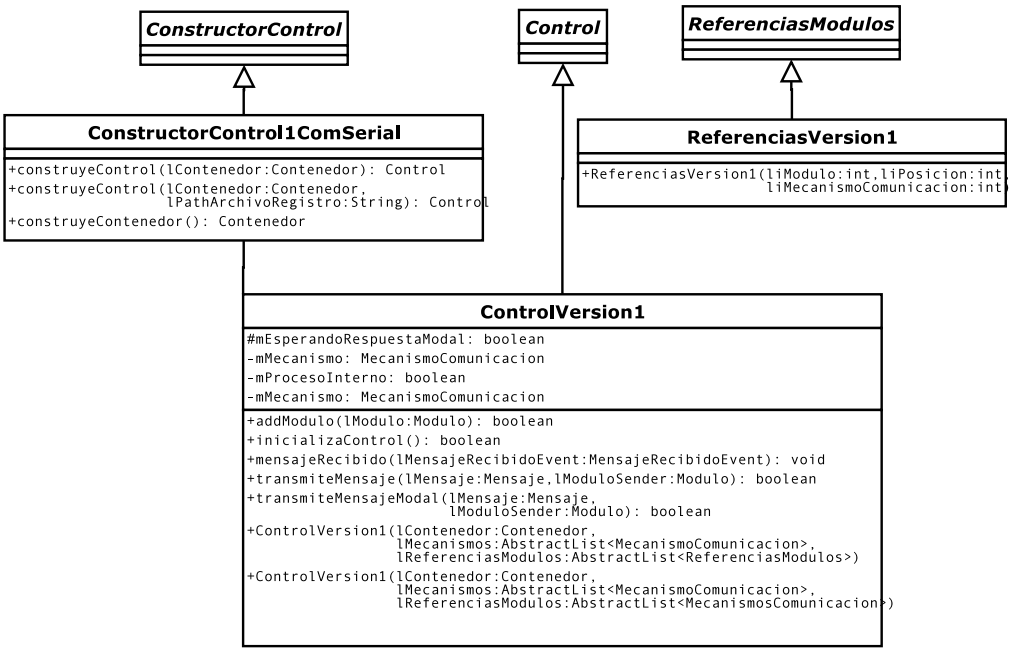


Figura 10: Clases concretas del paquete mx.ipn.cic.controladores.

ReferenciasVersion1 Su única labor es crear una clase concreta que utilice los métodos implementados por el padre (ReferenciasModulos).

ControlVersion1 Cumple con el mecanismo de control establecido dentro del estándar. Por definición, sólo utiliza un mecanismo de comunicación.

ConstructorVersion1ComSerial Se encarga de proporcionar el mecanismo de creación de contenedores que utilizan como lista, objetos de la clase java.util.Vector; también proporciona el mecanismo para crear los objetos de la clase ControlVersion1, con un mecanismo de comunicación serial almacenado en un objeto de la clase java.util.Vector y crea otro objeto de la misma clase en la cual listar los objetos de la clase ReferenciasModulos.

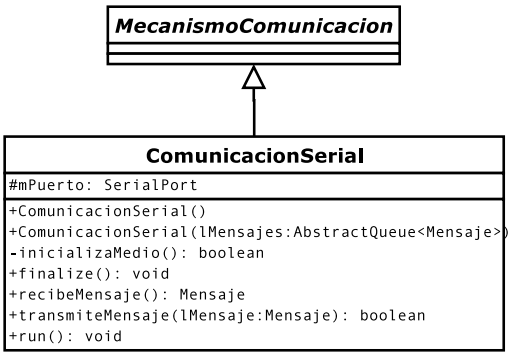


Figura 11: Clase concreta del paquete mx.ipn.cic.comunicaciones.

Dentro del paquete mx.ipn.cic.comunicaciones, es necesaria la creación de una clase para administrar la comunicación serial de la computadora (M0). Su labor principal es transmitir los mensajes indicados por el control, almacenar los mensajes recibidos en una cola (lista FIFO) e implementar un mecanismo para avisar acerca de la presencia de mensajes en espera al sistema de control. El diagrama de la clase se muestra en la figura 11.

Por último, los módulos construidos en hardware requieren de una clase por cada familia en el paquete mx.ipn.cic.modulos con la cual proporcionar la abstracción de alto nivel a la Capa de Aplicación. Al ser clases de pruebas, las instrucciones de bajo nivel han sido sobreescritas con visibilidad pública y de ésta forma hacer posible su utilización desde la Capa de Aplicación.

La implementación de dichas clases concretas se enlista a continuación.

5.1. mx.ipn.cic.controladores.ReferenciasVersion1

```
1 package mx.ipn.cic.controladores;
2 /**Clase concreta utilizada en el control version 1.*/
3 public class ReferenciasVersion1 extends ReferenciasModulos {
4         /** Crea una nueva instancia de la clase ReferenciasModulos
5          *@param liModulo Indice del modulo en la lista abstracta de modulos.
```

```
6           *@param liPosicion Posicion en el modelo de cada modulo.
7           *@param liMecanismoComunicacion Indice del mecanismo de comunicacion en la
8           * lista abstracta.*/
9           public ReferenciasVersion1(int liModulo, int liPosicion,
10                                     int liMecanismoComunicacion){
11                  super(liModulo,liPosicion,liMecanismoComunicacion);
12          }
13  }
```

5.2. mx.ipn.cic.controladores.ControlVersion1

```
1  ackage mx.ipn.cic.controladores;
2  import java.util.AbstractList;
3  import java.util.Calendar;
4  import mx.ipn.cic.modulos.Modulo;
5  import mx.ipn.cic.comunicaciones.*;
6  import mx.ipn.cic.utilerias.*;
7  /**Control con un solo canal de comunicacion, al cual asigna todos los modulos,
8   * al agregar un modulo le asigna dicho canal.
9   * @see  mx.ipn.cic.controladores.ConstructorControl1ComSerial*/
10 public class ControlVersion1 extends Control {
11          /**Indica si se espera una respuesta modal de alguna instruccion.*/
12          protected static boolean mEsperandoRespuestaModal=true;
13          /**Unico mecanismo de comunicacion empleado.*/
14          private static MecanismoComunicacion mMecanismo;
15          /**Identificar si el tratamiento de lensaje entrante es normal o especial.*/
16          private boolean mProcesoInterno;
17          /**Respuesta a una instruccion. <code>true</code> si se recibe Done o Send,
18           *  <code>false</code> en caso de Fail*/
19          private boolean mRespuesta;
20          /** Crea una nueva instancia de la clase ControlVersion1.
21           *@param lContenedor Es el contenedor del control y los modulos.
22           *@param lMecanismos Es la lista de mecanismos de comunicacion.
23           *@param lReferenciasModulos Es la lista de referencias de los modulos.*/
24          public ControlVersion1(Contenedor lContenedor,
25                                 AbstractList<MecanismoComunicacion> lMecanismos,
26                                 AbstractList<ReferenciasModulos> lReferenciasModulos){
27                  super(lContenedor,lMecanismos,lReferenciasModulos);
28                  mMecanismo=lMecanismos.get(0);
29                  mMecanismo.addMensajeRecibidoListener(this);
30                  mRegistroActividades.setBufferSize(1);
31          }
32          /** Crea una nueva instancia de la clase ControlVersion1.
33           *@param lContenedor Es el contenedor del control y los modulos.
34           *@param lMecanismos Es la lista de mecanismos de comunicacion.
35           *@param lReferenciasModulos Es la lista de referencias de los modulos.
36           *@param lRegistroActividades Es el Administrador del registro de
37           *       actividades.*/
38          public ControlVersion1(Contenedor lContenedor,
39                                 AbstractList<MecanismoComunicacion> lMecanismos,
40                                 AbstractList<ReferenciasModulos> lReferenciasModulos,
41                                 RegistroActividades lRegistroActividades){
42                  super(lContenedor,lMecanismos,lReferenciasModulos,lRegistroActividades);
43                  mMecanismo=lMecanismos.get(0);
44                  mMecanismo.addMensajeRecibidoListener(this);
45                  mRegistroActividades.setBufferSize(1);
46          }
47          /**Camino por el cual un modulo envia un mensaje. Completa el mensaje
48           *y lo envia al canal de comunicacion asignado al modulo
49           *@param lMensaje Es el mensaje que debe completarse con la posicion del
50           *          destinatario y enviarse.
51           *@param lModuloSender Es el modulo que envia el mensaje.
52           *@return <code>true</code> si el mensaje se ha transmitido exitosamente,
53           *          <code>false</code> de lo contrario. Considerar la opcion de
```

```
54          *           modalidad.
55          *@see mx.ipn.cic.modulos.Modulo#mModal*/
56         public boolean transmiteMensaje(Mensaje lMensaje, Modulo lModuloSender){
57                 int liIndiceReferencias=lModuloSender.getPosicion()-1;
58                 ReferenciasModulos lTabla= mReferenciasModulos.get(liIndiceReferencias);
59                 if(lTabla.isOcupado())
60                         return false;
61                 int posicion=lModuloSender.getPosicion();
62                 if(posicion==-1)
63                         return false;
64                 lMensaje.setDestino(new UByte(posicion));
65                 lTabla.setOcupado(true);
66                 if(mMecanismo.transmiteMensaje(lMensaje)){
67                         mRegistroActividades.registra(lModuloSender,true,lMensaje);
68                         return true;
69                 }
70                 return false;
71         }
72         /**Camino por el cual un modulo envia un mensaje. Completa el mensaje
73          *y lo envia al canal de comunicacion asignado al modulo. Modal significa
74          *que no se regresa el control hasta evaluar si la respuesta del modulo.
75          *@param lMensaje Es el mensaje que debe completarse con la posicion del
76          *           destinatario y enviarse.
77          *@param lModuloSender Es el modulo que envia el mensaje.
78          *@return <code>true</code> si el mensaje se ha transmitido y ejecutado
79          *           exitosamente, <code>false</code> de lo contrario.
80          *           Considerar la opcion de modalidad.
81          *@see mx.ipn.cic.modulos.Modulo#mModal*/
82         public synchronized boolean transmiteMensajeModal(Mensaje lMensaje,
83                         Modulo lModuloSender){
84                 int liIndiceReferencias=lModuloSender.getPosicion()-1;
85                 ReferenciasModulos lTabla;
86                 try {
87                         lTabla = mReferenciasModulos.get(liIndiceReferencias);
88                 }catch(IndexOutOfBoundsException iobe){
89                         if(mRegistroActividades!=null)
90                                 mRegistroActividades.registra("El modulo:"+lModuloSender+
91                                         " no se encuentra conectado");
92                         System.err.println("El modulo:"+lModuloSender+
93                                         " no se encuentra conectado");
94                         return false;
95                 }
96                 if(lTabla.isOcupado())
97                         return false;
98                 int posicion=lModuloSender.getPosicion();
99                 if(posicion==-1)
100                        return false;
101                lMensaje.setDestino(new UByte(posicion));
102                try{wait(2000);}catch(InterruptedException ie){
103                        ie.printStackTrace();
104                }
105                lTabla.setOcupado(true);
106                if(!mMecanismo.transmiteMensaje(lMensaje))
107                        return false;
108                mRegistroActividades.registra(lModuloSender,true,lMensaje);
109                mEsperandoRespuestaModal=true;
110                Calendar lAhora=Calendar.getInstance();
111                lAhora.add(Calendar.MILLISECOND,miTiempoMaxDeEspera);
112                while(mEsperandoRespuestaModal){
113                        if(lAhora.getTimeInMillis()<=
114                                        Calendar.getInstance().getTimeInMillis()){
115                                if(mRegistroActividades!=null)
116                                        mRegistroActividades.registra(
117                                                "Tiempo de espera agotado,"+lModuloSender+
118                                                ","+lMensaje);
```

```
119                                         System.err.println(
120                                                 "Tiempo de espera agotado,"+lModuloSender+
121                                                 ","+lMensaje);
122                                         break;
123                                 }
124                         }
125                         lTabla.setOcupado(false);
126                         return mRespuesta;
127                 }
128                 /**Analiza los elementos conectados externamente, actualiza la posicion en
129                  *la tabla de modulos y referencias ejecuta el home de todos los modulos
130                  *conectados.
131                  *@return <code>true</code> si el prototipo se ha inicializado con al menos
132                  *          un modulo conectado y funcionando, <code>false</code> de lo
133                  *          contrario.*/
134                 public boolean inicializaControl(){
135                         mProcesoInterno=true;
136                         int liArray[]=new int[2];
137                         UByte lFamilia;
138                         UByte lArray[];
139                         Mensaje lMensaje;
140                         Modulo lModulo=null;
141                         liArray[0]=255;
142                         liArray[1]=1;
143                         lArray=UByte.creaArray(2,liArray);
144                         lMensaje=new Mensaje(new UByte(1),new UByte(2),new UByte(7),lArray);
145                         if(!mMecanismo.transmiteMensaje(lMensaje))
146                                 return false;
147                         mRegistroActividades.registra(lModulo,true,lMensaje);
148                         mEsperandoRespuestaModal=true;
149                         while(mEsperandoRespuestaModal);
150                         lMensaje=mMecanismo.recibeMensaje();
151                         mRegistroActividades.registra(lModulo,false,lMensaje);
152                         UByte lDestino=lMensaje.getDestino();
153                         //Contabiliza modulos conectados en el mecanismo.
154                         miTotalModulos=255-lDestino.intValue()+1;
155                         if(miTotalModulos==0)
156                                 return false;
157                         for(int i=0;i<miTotalModulos;i++){
158                                 liArray[0]=miTotalModulos-i;
159                                 liArray[1]=i+1;
160                                 lArray=UByte.creaArray(2,liArray);
161                                 //Envia Home con el M0 y la posicion.
162                                 lMensaje=new Mensaje(new UByte(i+1),new UByte(2),new UByte(7),
163                                         lArray);
164                                 if(!mMecanismo.transmiteMensaje(lMensaje))
165                                         return false;
166                                 mRegistroActividades.registra(lModulo,true,lMensaje);
167                                 mEsperandoRespuestaModal=true;
168                                 while(mEsperandoRespuestaModal);
169                                 lMensaje=mMecanismo.recibeMensaje();
170                                 mRegistroActividades.registra(lModulo,false,lMensaje);
171                                 //Ubica familia para insertar las referencias.
172                                 lMensaje=new Mensaje(new UByte(i+1),new UByte(0),new UByte(5));
173                                 if(!mMecanismo.transmiteMensaje(lMensaje))
174                                         return false;
175                                 mRegistroActividades.registra(lModulo,true,lMensaje);
176                                 mEsperandoRespuestaModal=true;
177                                 while(mEsperandoRespuestaModal);
178                                         lMensaje=mMecanismo.recibeMensaje();
179                                 mRegistroActividades.registra(lModulo,false,lMensaje);
180                                 lFamilia=lMensaje.getParametro(1);
181                                 //Buscar la primera coincidencia en la cual se cumpla la familia.
182                                 int j=0;
183                                 for(j=0;j<mContenedor.getCantidadModulos();j++){
```

```
184                             lModulo=mContenedor.getModulo(j);
185                             if(lModulo.getFamilia().compareTo(
186                                             new Familia(lFamilia))==0 &&
187                                             lModulo.getPosicion()==-1){
188                                     lModulo.setM0(new UByte(miTotalModulos-i));
189                                     lModulo.setPosicion(i+1);
190                                     ReferenciasVersion1 ltab=
191                                             new ReferenciasVersion1(j,i+1,0);
192                                     mReferenciasModulos.add(ltab);
193                                     break;
194                             }
195                     }
196                     if(lModulo.getPosicion()==-1){
197                             ReferenciasVersion1 ltab= new ReferenciasVersion1(j,-1,0);
198                             mReferenciasModulos.add(ltab);
199                             if(mRegistroActividades!=null)
200                                     mRegistroActividades.registra("Familia :"+lFamilia+
201                                     " no encontrada en mecanismo de comunicacion:"+0);
202                             System.err.println("Familia :"+lFamilia+
203                                     " no encontrada en mecanismo de comunicacion:"+0);
204                             continue;
205                     }
206             }
207             mProcesoInterno=false;
208             return true;
209     }
210     /**Mecanismo de recepcion de mensajes. Se ejecuta cuando el mecanismo de
211      *comunicacion ha concretado la recepcion de un mensaje completo.
212      *Actualiza la informacion de los parametros de los modulos.
213      *@param lEvento Evento de completar un mensaje.*/
214     public void mensajeRecibido(MensajeRecibidoEvent lEvento) {
215             if(lEvento.getEstado()==MensajeRecibidoEvent.MENSAJE_DISPONIBLE &&
216                             mEsperandoRespuestaModal){
217                     mEsperandoRespuestaModal=false;
218             }
219             if(mProcesoInterno)
220                     return;
221             Mensaje lMensaje = mMecanismo.recibeMensaje();
222             //Descrementar el desstino en uno, si no es cero, reenviar el valor
223             if(lMensaje.getDestino().intValue()-1!=0){
224                     lMensaje.setDestino(new UByte(lMensaje.getDestino().intValue()-1));
225                     mMecanismo.transmiteMensaje(lMensaje);
226                     return;
227             }
228             //Si es cero, el mensaje es para mi.
229             int liInstruccion=lMensaje.getInstruccion().intValue();
230             UByte lParams[]=lMensaje.getParametros();
231             ReferenciasModulos lReferencias=
232             mReferenciasModulos.get(lParams[0].intValue()-1);
233             Modulo lModulo = mContenedor.getModulo(lReferencias.getModulo());
234             lReferencias.setOcupado(false);
235             mRegistroActividades.registra(lModulo,false,lMensaje);
236             if(liInstruccion==3){ //Done
237                     mRespuesta=true;
238             }
239             else if(liInstruccion==4){ //Fail
240                     mRespuesta=false;
241             }
242             else if(liInstruccion==9){ //Send
243                     if(lParams.length==2){
244                             lModulo.setValoresRegreso(lParams[1]);
245                     }
246                     else{
247                             UByte lValores[]=new UByte[lParams.length-1];
248                             for(int i=0; i< lValores.length;lValores[i]=lParams[i++1]);
```

```
249                                     lModulo.setValoresRegreso(lValores);
250                             }
251                             mRespuesta=true;
252                     }
253                     else{
254                             if(mRegistroActividades!=null){
255                                     mRegistroActividades.registra("Otra Respuesta desconocida "+
256                                             "recibida: "+lMensaje);
257                             }
258                     System.err.println("Otra Respuesta desconocida recibida: "+
259                     lMensaje);
260                     mRespuesta=false;
261                     }
262             }
263 }
```

5.3. mx.ipn.cic.controladores.ConstructorControl1ComSerial

```
1  package mx.ipn.cic.controladores;
2  import java.util.concurrent.ConcurrentLinkedQueue;
3  import mx.ipn.cic.comunicaciones.ComunicacionSerial;
4  import mx.ipn.cic.comunicaciones.MecanismoComunicacion;
5  import mx.ipn.cic.comunicaciones.MensajeRecibidoEvent;
6  import mx.ipn.cic.comunicaciones.MensajeRecibidoListener;
7  import mx.ipn.cic.modulos.Modulo;
8  import mx.ipn.cic.utilerias.Contenedor;
9  import mx.ipn.cic.excepciones.ImposibleInicializarMedioException;
10 import java.util.Vector;
11 import mx.ipn.cic.utilerias.RegistroActividades;
12 /**Constructor del controlador para la version 1 del estandar. Se crea una via
13  *de Comunicacion Serial y tanto la lista de los mecanismos de comunicacion,
14  *como la tabla de referencias son objetos de la clase
15  * <code>java.util.Vector</code>.*/
16 public class ConstructorControl1ComSerial extends ConstructorControl {
17         /**Primer control creado.*/
18         private static ControlVersion1 mControl;
19         /**Construye un Control con un mecanismo de comunicacion serial y un vector
20          * para almecenar los modulos.
21          * @return El control al que deben ser agregados los modulos.
22          * @param lContenedor Es el contenedor en el cual esta agregado el Control.
23          * @throws mx.ipn.cic.excepciones.ImposibleInicializarMedioException
24          *       Ejecutada cuando no es posible iniciar el medio de comunicacion.*/
25         public static Control construyeControl(Contenedor lContenedor)
26                         throws ImposibleInicializarMedioException{
27                 if(mControl!=null) return mControl;
28                 ComunicacionSerial lComunicacionSerial = new ComunicacionSerial();
29                 Vector<MecanismoComunicacion> lvecCom =
30                         new Vector<MecanismoComunicacion>();
31                 lvecCom.addElement(lComunicacionSerial);
32                 Vector<ReferenciasModulos> lvecTabla=new Vector<ReferenciasModulos>();
33                 mControl=new ControlVersion1(lContenedor, lvecCom,
34                         lvecTabla);
35                 return mControl;
36         }
37         /**Crea y regresa un Control que agrega el registro de las actividades de
38          * los modulos. Un usuario de estandar debe utilizar siempre este metodo
39          * para crear el controlador.
40          * @param lContenedor Es el contenedor en el cual esta agregado el Control.
41          * @param lPathArchivoRegistro Path del archivo de registro de las
42          *       actividades.
43          * @return El Controlador deseado.
44          * @throws mx.ipn.cic.excepciones.ImposibleInicializarMedioException
45          *       Ejecutada cuando no es posible iniciar el medio de comunicacion*/
46         public static Control construyeControl(Contenedor lContenedor,
```

```
47                      String lPathArchivoRegistro)
48                      throws ImposibleInicializarMedioException {
49              if(mControl!=null) return mControl;
50              ComunicacionSerial lComunicacionSerial = new ComunicacionSerial();
51              Vector<MecanismoComunicacion> lvecCom =
52                      new Vector<MecanismoComunicacion>();
53              lvecCom.addElement(lComunicacionSerial);
54              Vector<ReferenciasModulos> lvecTabla=new Vector<ReferenciasModulos>();
55              mControl=new ControlVersion1(lContenedor, lvecCom,
56                      lvecTabla, new RegistroActividades(lPathArchivoRegistro));
57              return mControl;
58      }
59      /**Crea y regresa un contenedor con Vectores tanto para los controles como
60       * para los modulos.
61       *@return El Contenedor deseado.*/
62      public static Contenedor construyeContenedor(){
63              return new Contenedor(new Vector<Control>(),new Vector<Modulo>());
64      }
65 }
```

5.4. mx.ipn.cic.comunicaciones.ComunicacionSerial

```
1  package mx.ipn.cic.comunicaciones;
2  import java.io.IOException;
3  import java.util.AbstractQueue;
4  import java.util.EventObject;
5  import java.util.TooManyListenersException;
6  import java.util.concurrent.ConcurrentLinkedQueue;
7  import gnu.io.*;
8  import mx.ipn.cic.excepciones.ImposibleInicializarMedioException;
9  import mx.ipn.cic.utilerias.*;
10 /**Esta clase implementa lo estipulado en el estandar para la comunicacion
11  * serial.
12  * Encapsula el envio y la recepcion de mensajes para abstraer el proceso en
13  * el controlador y desacoplar dicha funcionalidad.
14  *@see mx.ipn.cic.excepciones.ImposibleInicializarMedioException
15  *@see <a href="http://www.rxtx.org/">RXTX</a>*/
16 public class ComunicacionSerial extends MecanismoComunicacion{
17      /**Puerto Serial*/
18      private SerialPort mPuerto;
19      /**Crea una nueva instancia de la clase ComunicacionSerial con una cola
20       * @throws mx.ipn.cic.excepciones.ImposibleInicializarMedioException
21       *      Ejecutada cuando no es posible iniciar el medio de comunicacion*/
22      public ComunicacionSerial() throws ImposibleInicializarMedioException{
23              mMensajes=new ConcurrentLinkedQueue<Mensaje>();
24              if(!inicializaMedio())
25                      throw new ImposibleInicializarMedioException();
26      }
27      /**Crea una nueva instancia de la clase ComunicacionSerial con
28       * <code>lMensajes</code> como cola.
29       * @param lMensajes Cola de mensajes.
30       * @throws mx.ipn.cic.excepciones.ImposibleInicializarMedioException
31       *      Ejecutada cuando no es posible iniciar el medio de comunicacion*/
32      public ComunicacionSerial(AbstractQueue<Mensaje> lMensajes)
33                      throws ImposibleInicializarMedioException{
34              super(lMensajes);
35              if(!inicializaMedio())
36                      throw new ImposibleInicializarMedioException();
37      }
38      /**Transmite el mensaje por el puerto Serial.
39       * @param lMensaje es el mensaje a recibir, ya debe estar validado.
40       * @return <CODE>true</CODE> si la transmision se realizo con exito,
41       *          <CODE>false</CODE> de lo contrario.*/
42      public boolean transmiteMensaje(Mensaje lMensaje) {
```

```
43                  System.out.println("Transmitiendo:\t"+lMensaje);
44                  int mandando[]=lMensaje.toIntArray();
45                  for(int i=0;i< mandando.length;i++){
46                          try {
47                                  mPuerto.getOutputStream().write(mandando[i]);
48                                  mPuerto.getOutputStream().flush();
49                          } catch (IOException ex) {
50                                  System.err.println("Error con el puerto Serial:"+ex);
51                                  ex.printStackTrace(System.err);
52                          }
53                  }
54                  return true;
55          }
56          /**Obtiene la cabecera de la cola.
57           *@return El mensaje que haya llegado primero, <code>null</code> si la cola
58           *       se encuentra vacia.*/
59          public Mensaje recibeMensaje() {
60                  Mensaje lMensaje=mMensajes.poll();
61                  System.out.println("Recibido:\t"+lMensaje);
62                  return lMensaje;
63          }
64          /**Clase que administra la generacion de los mensajes a partir de los bytes
65           * recibidos por el puerto serial.*/
66          private class SerialListener implements SerialPortEventListener{
67                  /**Arreglo de UBytes del mensaje.*/
68                  private UByte    mArray[];
69                  /**Indice del arreglo.*/
70                  private int      miIndice;
71                  /**Cantidad de parametros del mensaje.*/
72                  private int      miTamanio;
73                  /**Padre de la clase.*/
74                  private ComunicacionSerial mPadre;
75                  /**Construye el listener con el lPadre.
76                   * @param lPadre la clase que utiliza la funcionalidad de esta.*/
77                  public SerialListener(ComunicacionSerial lPadre){
78                          mPadre=lPadre;
79                          miTamanio=miIndice=0;
80                  }
81                  /**Maneja el evento que sucede cuando un dato se encuentra disponible
82                   *en el buffer de recepcion serial.
83                   *@param spe Es el evento acontecido.*/
84                  public void serialEvent(SerialPortEvent spe){
85                          if(spe.getEventType()!=SerialPortEvent.DATA_AVAILABLE)
86                                  return;
87                          UByte lUByte=null;
88                          try{
89                                  lUByte=new UByte(mPuerto.getInputStream().read());
90                          } catch (NumberFormatException ex) {
91                                  System.err.println("Error con el valor del puerto: "+ex);
92                                  ex.printStackTrace(System.err);
93                          } catch (IOException ex) {
94                                  System.err.println("Error de entrada/salida: "+ex);
95                                  ex.printStackTrace(System.err);
96                          }
97                          if(mArray==null){ //Nuevo mensaje, se recibe Destinatario.
98                                  miIndice=miTamanio=0;
99                                  mArray=new UByte[16];
100                                 mArray[0]=lUByte;
101                                 miIndice++;
102                         }
103                         else{
104                                 if(miIndice==1){ //Recibir mTamanio
105                                         mArray[miIndice++]=lUByte;
106                                         miTamanio=lUByte.intValue();
107                                 }
```

```
108                                     else if(miIndice==2) //Recibir Instruccion
109                                             mArray[miIndice++]=lUByte;
110                                     else if(miIndice<miTamanio+3)//Reciben Parametros
111                                             mArray[miIndice++]=lUByte;
112                                     if(miIndice==miTamanio+3){ //mensaje terminado
113                                             UByte lParams[]=new UByte[miTamanio];
114                                             for(int i=0;i<miTamanio;lParams[i]=mArray[i+++3]);
115                                             mMensajes.add(new Mensaje(mArray[0],mArray[1],
116                                                     mArray[2],lParams));
117                                             mArray=null;
118                                             fireMensajeRecibidoEvent(new MensajeRecibidoEvent(
119                                                     (ComunicacionSerial)mPadre,
120                                                     MensajeRecibidoEvent.MENSAJE_DISPONIBLE));
121                                     }
122                             }
123                     }
124             }
125             /**Inicializa el mecanismo de comunicacion
126              * @return <CODE>true</CODE> en caso de exito, <CODE>false</CODE> de lo
127              *       contrario.
128              * @throws mx.ipn.cic.excepciones.ImposibleInicializarMedioException
129              *       Ejecutada cuando no es posible iniciar el medio de comunicacion*/
130             private boolean inicializaMedio() throws ImposibleInicializarMedioException{
131                     try {
132                             mPuerto = new RXTXPort("COM1");
133                             mPuerto.addEventListener(new SerialListener(this));
134                             mPuerto.notifyOnDataAvailable(true);
135                     } catch (PortInUseException ex) {
136                             System.err.println("Puerto Serial ocupado: "+ex);
137                             ex.printStackTrace(System.err);
138                             return false;
139                     } catch(TooManyListenersException tmle) {
140                             System.err.println("Demasiados listeners utilizados: "+tmle);
141                             tmle.printStackTrace(System.err);
142                             return false;
143                     }
144                     try {
145                             mPuerto.setFlowControlMode(SerialPort.FLOWCONTROL_NONE);
146                             mPuerto.setSerialPortParams(9600,SerialPort.DATABITS_8,
147                             SerialPort.STOPBITS_1,SerialPort.PARITY_NONE);
148                     } catch (UnsupportedCommOperationException ucoe) {
149                             System.err.println("Operacion no valida del puerto serial: "+
150                                     ucoe);
151                             ucoe.printStackTrace(System.err);
152                             return false;
153                     }
154                     return true;
155             }
156             /**Finaliza el objeto. Cierra el puerto abierto de la comunicacion serial.*/
157             public void finalize(){
158                     mPuerto.close();
159             }
160             /**Implementacion de Runnable.*/
161             public void run() {
162                     System.out.println("Run ComunicacionSerial, Run!");
163             }
164     }
```

5.5. mx.ipn.cic.modulos.ModuloPrueba

```
1 package mx.ipn.cic.modulos;
2 import mx.ipn.cic.controladores.Control;
3 import mx.ipn.cic.utilerias.Familia;
4 import mx.ipn.cic.utilerias.UByte;
```

```
5  /**Modulo concreto para llevar a cabo algunas pruebas con los modulos.*/
6  public class ModuloPrueba extends Modulo{
7          /**Crea una nueva instancia de la clase ModuloPrueba con un controlador
8           *dado. Inicializa la familia.
9           *@param lControl Es el controlador que utiliza el modulo.*/
10         public ModuloPrueba(Control lControl) {
11                 super(lControl);
12                 mFamilia=new Familia(new UByte(0),new UByte(1));
13         }
14         /**Prueba 2. doo -> toggle -> doo
15          *1 5 1 34 32 34 32 34 <p>
16          *1 0 B <p>
17          *1 5 1 64 37 78 83 93 <p>
18          * @return respuesta*/
19         public boolean prueba2(){
20                 int []tmp=new int[5];
21                 tmp[0]=0x34;
22                 tmp[1]=0x32;
23                 tmp[2]=0x34;
24                 tmp[3]=0x32;
25                 tmp[4]=0x34;
26                 if(!super.doo(UByte.creaArray(5,tmp))) return false;
27                 if(!super.toggle()) return false;
28                 tmp[0]=0x64;
29                 tmp[1]=0x37;
30                 tmp[2]=0x78;
31                 tmp[3]=0x83;
32                 tmp[4]=0x93;
33                 if(!super.doo(UByte.creaArray(5,tmp))) return false;
34                 return true;
35         }
36         /**Prueba 3.<p>
37          *RESPTOGG, TOGGLE, DOO, TOGGLE Y DOO<p>
38          *Mensajes:<p>
39          * 1 1 7 3<p>
40          * 1 0 8<p>
41          * 1 0 B<p>
42          * 1 5 1 10 20 30 40 50<p>
43          * 1 0 B<p>
44          * 1 5 1 60 70 80 90 A0<p>
45          * @return respuesta*/
46         public boolean prueba3(){
47                 respTogg();
48                 toggle();
49                 int []tmp=new int[5];
50                 tmp[0]=0x10;
51                 tmp[1]=0x20;
52                 tmp[2]=0x30;
53                 tmp[3]=0x40;
54                 tmp[4]=0x50;
55                 doo(UByte.creaArray(5,tmp));
56                 toggle();
57                 tmp[0]=0x60;
58                 tmp[1]=0x70;
59                 tmp[2]=0x80;
60                 tmp[3]=0x90;
61                 tmp[4]=0xA0;
62                 doo(UByte.creaArray(5,tmp));
63                 return true;
64         }
65         /**Prueba 4
66          *DOINSTR, ABORT, DOO, DOO<p>
67          *Mensajes:<p>
68          * 1 1 7 3<p>
69          * 1 1 2 FF<p>
```

```
70          * 1 0 0<p>
71          * 1 1 2 EE<p>
72          * 1 1 2 DD<p>*/
73         public boolean prueba4(){
74                 UByte lParametro = new UByte(3);
75                 doInst(lParametro);
76                 abort();
77                 doo();
78                 doo();
79                 return true;
80         }
81         /**Mapea la instruccion del estandar.*/
82         public boolean abort(){
83                 return super.abort();
84         }
85         /**Mapea la instruccion del estandar.*/
86         public boolean doo(UByte []tmp){
87                 return super.doo(tmp);
88         }
89         /**Mapea la instruccion del estandar.*/
90         public boolean home(UByte lpar,UByte lpos){
91                 return super.home(lpar,lpos);
92         }
93         /**Mapea la instruccion del estandar.*/
94         public boolean setParam(UByte lPar,UByte lVal){
95                 return super.setParam(lPar,lVal);
96         }
97         /**Mapea la instruccion del estandar.*/
98         public boolean toggle(){
99                 return super.toggle();
100        }
101 }
```

6. Resultados

En la figura 12 se muestran tres módulos de prueba que fueron implementados para la comprobación de [2].

7. Trabajo Futuro

Algunos puntos por realizar son los siguientes:

- Implementar el módulo base para resolver problemas modularmente separables, por ejemplo, automatizar líneas de producción y crear diferentes tipos de robots. Áreas como la domótica, juguetería y entretenimiento, industria automotríz, etc.
- Difundir la especificación en la búsqueda de desarrolladores de módulos de hardware que generen un conjunto amplio de opciones en el área para que la generación de prototipos sea tan rápida como conectar y probar.
- Abordar otros medios, dispositivos, mecanismos y protocolos de comunicación. Como son: USB, Firewire, Bluetooth, etc.
- Crear nuevos controles, es decir, mecanismos mediante los cuales administrar los módulos.

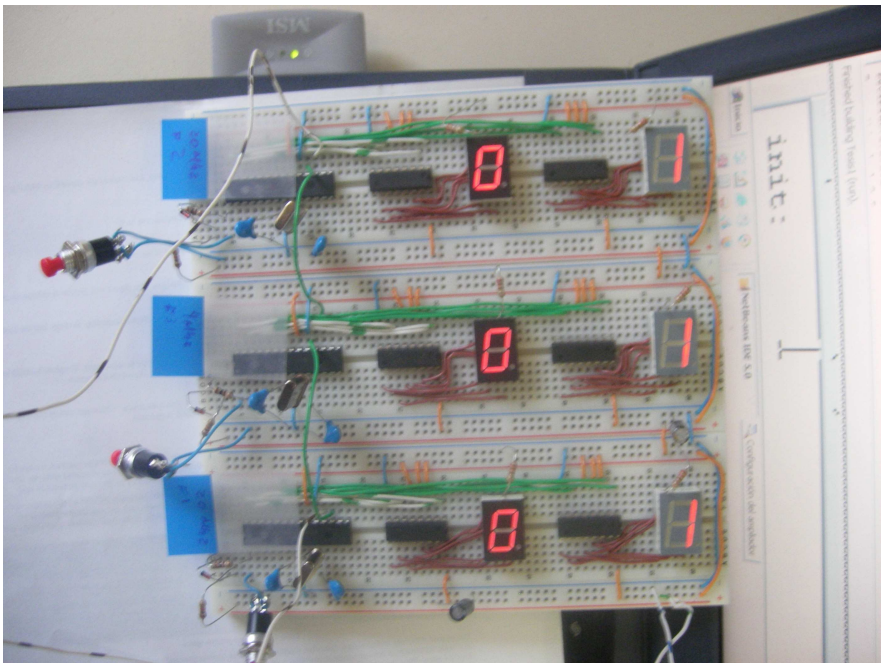


Figura 12: Modulos de Prueba en uso.

- Proponer un organismo que regule las nuevas familias de módulos por crear.
- Crear metodologías de diseño de prototipos, es decir, representar la funcionalidad de los módulos de hardware y sus respectivas clases de software mediante diagramas o algún tipo de lenguaje para la descripción.

Referencias

[1] Daniel Aaron Torrescano Arias and Juan Luis Díaz de León Santiago. *"Paquete de LaTeX 2ε para escribir Informes Técnicos del CIC-IPN"*.

[2] Daniel Aaron Torrescano Arias and Juan Luis Díaz de León Santiago. "Estándar de Hardware y Software, para el Control Modular de Prototipos Industriales". Master's thesis, Centro de Investigación en Computación, CIC-IPN, 2007.

[3] H.M. Deitel and P.J. Deitel. *"Cómo Programar en Java"*. Pearson Educación, 1a. edición edition, 1995.

[4] Eric Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *"Design Patterns. Elements of Reusable Object Oriented Software"*. Addison Wesley Longman, Inc., 14th edition, 1998.

[5] James Rumbaugh, Ivar Jacobson, and Grady Booch. *"El Lenguaje Unificado de Modelado. Manual de Referencia.* 1999.

[6] Carsten Nitsch, Karlheinz Weiss, Thorsten Steckstor, and Wolfgang Rosentiel. "Embedded System Architecture Design Based on Real-Time Emulation". Paris, Francia, 2000. RSP'2000.