Report on the works in the Xplore Project about the

Control of a flexible manufacturing cell by Petri nets in Java language.

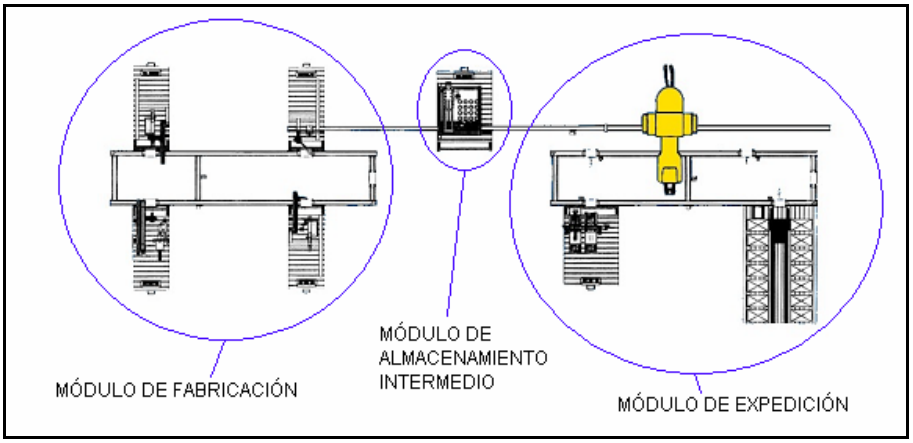
Ramón Piedrafita Moreno
Oscar García Flores
Alberto Gran tejero
Departamento de Informática e Ingeniería de Sistemas. Universidad de Zaragoza.

The proposal of this project is to get the control of the manufacturing cell of the Informatic and System Engineering department with Java language by an Interbus net. For this task a group of classes is going to be implemented that will model the cell's behaviour and let to control by means of Petri nets thought an Interbus net, also implemented with java. The communication with the cell is carried out by means of a controller card of Interbus net IBS PCI SC/I-T of Phoenix Contact.

1.- The flexible manufacturing cell.

The flexible manufacturing cell consists of several stations that allow the manufacture and the storage of pneumatic cylinders with different colours and taps. The cell is designed in order to work as help for the study of a real manufacturing process without have to resort to a full scale industrial process.

The cell is divided in two parts: the manufacturing zone that consists of the stations 1, 2, 3, 4 and the transport 1, and the dispatch zone that consists of the stations 6, 7, 8, 9 and the transport 2. The station 5 is the intermediate storage module that is the junction between the two parts.

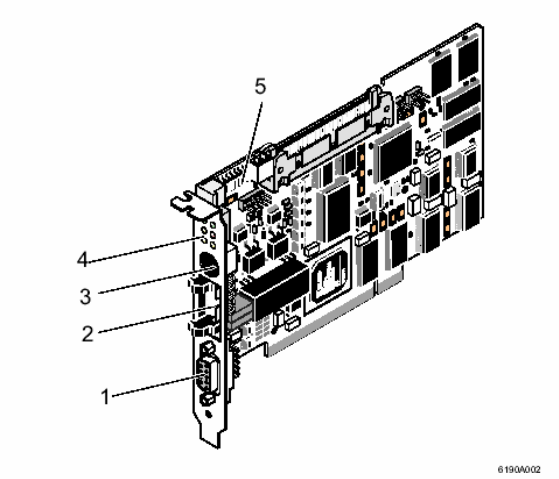


A lot of final degree projects have been carried out with this manufacturing cell, working over the stations communication. This communication is carried out through a Fipway net that subsequently, by means of the Premium and Micro of Modicon Telemecanique automatons, is controlled with the help of PL7pro software.

For this project we begin from a different base. An Interbus net has been built between the stations 1, 3, 4 and the transport 1 (manufacturing zone). So, Interbus Inline modules has been placed in the stations 1 and 4 while in the station 3 and the transport 1 we have used a connection with a module TSX Momentum of Schneider Electric with 16 digital inputs/outputs and 2 analogical inputs/outputs. We have to find the way to control the stations with this base. For more information about the stations function you can find a lot of projects and practise about the subject industrial informatic of Escuela Universitaria de Ingeniería Técnica Industrial de Zaragoza, EUITIZ [1], or go to the information in the web page [2], where you can find more information about the function of the stations.

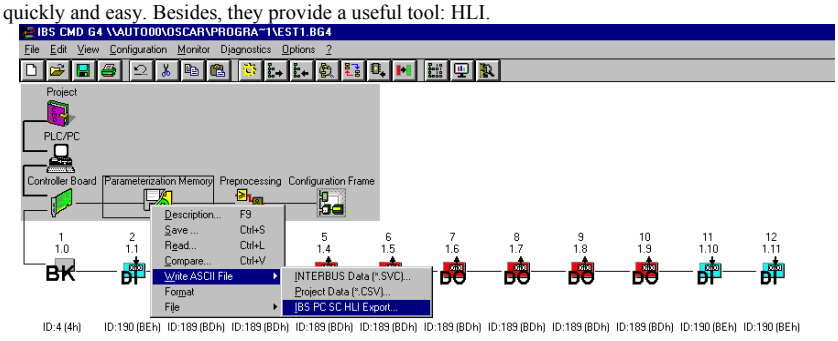
2.- The bus controller board IBS PCI SC/I-T. INTERBUS.

The board IBS PCI SC/I-T is a 4 generation controller board of Interbus with a remote interface for the PCI bus. By means of the CMD software that comes with the controller board, the parameters of the process to control can be configured completely. The IBS PCI SC/I-T board let us, by means of a programmable memory EPROM, the storage of the parameter data inside the board. [3].



According to the official page of Interbus [4], the INTERBUS net (IBS) provide us a serial link able to transmit data of I-O with real time speed. Here, real time means that the I-O data are refreshed a lot of times faster than the application can resolve the logic. The basic concept of an open bus system is to let us an exchange of information between devices manufactured by different manufacturers. The information includes commands and I-O data that have been defined as a standard profile by means of which the devices operate. The standard profiles are available for drivers, encoders, robotic controllers, pneumatic valves, etc. The INTERBUS protocol, DIN 19258, is the communication standard for this profiles. It is an open standard for the I-O nets for industrial applications.

The Phoenix Contact board provides us the CMD software in order to control the components connected to the board. This software let configure the bus quickly and easy. Besides, they provide a useful tool: HLI.



2.1- HLI (High-level Language Interface).

HLI is a group of library that can be used in order to develop a control program of the components of the bus by means of a high-level language. The configuration is carried out directly by means of the CMD software. The reference manual [5] is together with the board.

The bus configuration can be configured manually, adding the components that we know that are connected in the net or making that CMD auto configure the bus, reading the components that are available. Once configured the bus, we have the option to export the configuration to a folder with high level language code in which we have three functions: initiate the bus, finish the bus and cyclical process.

Libraries are only available in the next programming languages:

1 Microsoft C/C++.
2 Borland C/C++ (o compatible).
3 Microsoft VB 4.0 (o posterior).
4 Borland Delphi 2.0 (o posterior).

So, due to our proposal is to work with java language, we will use language C libraries because of his compatibility with java is easier of implement than the others as it is explained in the next paragraph.

3.- From C to Java. JNI.

3.1 - Java

Java programming language come up in 1991 because of the work of a small group of people leaded by James Gosling, they bring forward that in the future the small electrical appliances must be connected between them and a computer in order to create a domestic net. In order to program them, they look out for a language that let them to work independently of the dispositive or platform where it

runs. Although the small electrical applicance world was not prepared for this quality step, internet was prepared. By means of the introduction of the java technology in the sailor Netscape Communicator in 1995 began the faster develop of the computer history, due mainly to the two main characteristics of java: his portability (the same code can run in many different platforms) and his security (java was thought as a high level language directed to objects in order to profit from the including and hiding characteristics of information between inherent object to this types of language)[6].

3.2 – Conversion process.

For this degree final project we want to work with the java programming language, that it is not one of the languages which HLI support. However there is a way to work with C functions that has been generated by means of CMD : JNI of Java Native Interface. The process to use native code inside a java program can be sum up into six steps:

1 - Write java code.
2 - Use javac in order to compile the code.
3 - Use javah –jni in order to create a header file.
4 - Write the implementation of the native method.
5 - Compile the native code in a shared library and load it.
6 - Run the program using the java interpret.

4.- Petri nets.

A Petri net is a graphic directed to objects in which take part two types of graphical junctions, the places (represented by circumferences) and the transitions (represented by rectilinear segments), joined alternatively by arcs.

Up to now the unique implementation of Petri nets available in the department was carried out in order to control a robot with Ada language. This implementation is based in the creation of a coordinator that look around the net and every cycle fire the transitions depending on the conflicts model, this conflict model must be done by eye before, during the net creation. This implementation does not take into account the existence of the places basing the net structure in arrays of transitions.

Due to the flexibility of Java, we use a different model from the previous implementation. We have created a different model that includes the places and which structure is explained down.

4.1 The basic class : Places and transitions.

Java is an object directed language, due to this, the first step was to consider the possible objects that take part in a Petri net. From here comes up the class `Estado` y `Transicion`. The class arc `Arco` is not implemented, to replace the arcs we have based the structure in the input places and/or output places of the transitions.

We have used the `java.util.Vector` class in his version 1.5.0 of JDK (Java Development Kit). The structure of the Petri net is defined by a group of places and transitions, being the transitions the ones that support the structure of the net. The temporization of the places and the priority of the transitions are auxiliary tools in order to control simulate a real process with this implementation of Petri net.

4.2 Net class.

Net class is the one which have the information about the structure of the Petri net. The "`Transicion`" Transition class have not information about his fire condition in order to control the Petri net evolution, to carry out this control it is needed a coordinator. The Net class have the fields:

```
public class Red {
  public int[][] matrizIncidenciaPrevia;
  public int[][] matrizIncidenciaPosterior;
  public int[] marcado;
  public int[] marcadoInicial;
  public int[][] matrizPesos;
  public Vector < Conflicto > conflictos;
  public Vector < Estado > estados;
  public Vector < Transicion > transiciones;
```

The definition of the terms associated to the Petri net, of which have been deduced the fields of this class, can be found in [10]. Their mean can be seen intuitively (contrary than other implementations of structure more complicated); The Petri net is defined by his matrix of previous incidence and matrix of later incidence, by his mark and by his initial mark "token". The weight matrix become necessary in order to implement the functionality of the arcs. All these parameters come from the information of the places vector and transitions vector or the other way round can be builded the Petri net with his matrix of incidence and take from them the group of places and transitions that form it.

A conflict is produced when two or more simultaneous sensitized transitions descends from the same place and this one does not have enough number of tokens in order to fire them simultaneously [11]. According to the said before, a conflict will be basically a vector of transitions that fulfil the condition. The implementation of the functionality of the conflicts will be of utility at the time of implement the

coordinator of the Petri net. Besides the creation of this array is builded automatically at the time of making the Petri net, so it is not necessary to carry out a study process of the Petri net before carry out the programming implementation of the Petri net such as it is carried out in the implementation of the Petri net with Ada.

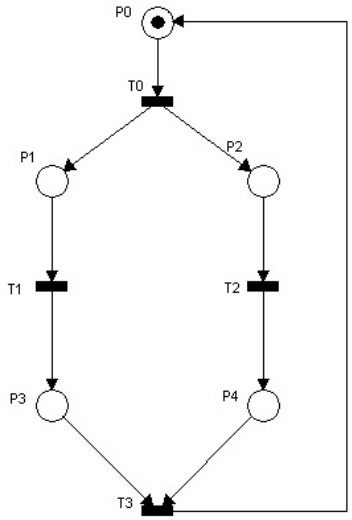
The methods that implements the creation of the Petri net are `construyeRedMatrizIncidencia,` that create the net from his incidence matrix and the initial mark and `construyeRedMatrizIncidencia` that create a Petri net from the places vector and the transitions vector.

4.2.1 Creation of a Petri net.

As it has been explained, there are two ways of create a Petri net: by means of his incidence matrix and his initial mark "token" or by means of his places and transitions vectors.

The first way is though in order to take advantage from the capacity editor of the Petri nets editor HPSim[12] that provide the parameters of a Petri net in text mode. With two of this parameters from the text file and then taking them to the method that builds the Petri net, we already have a Petri net rightly builded.

The second one way is though in order to create a Petri net by means of the creation of a subclass of Net that only contains the places and transitions implemented correctly, as it will be seen in the next example, with the next Petri net:



The net created with HPSim provide us a text file with the next form:

```
// Transition Name Vector:
(T0 ;T1 ;T2 ;T3 ;)
// Position Name Vector:
(P0;P1;P2;P3;P4;)
// Inzidenz Matrix:
{
( 1  0  0 -1 )
(-1  1  0  0 )
(-1  0  1  0 )
( 0 -1  0  1 )
( 0  0 -1  1 )
}
// Marking Vector:
(1 0 0 0 0 )
// Arc Type Matrix:
// Code:0 = None; 1 = Normal; 2 = Inhibitor; 3 = Test
{
(1 0 0 1 )
(1 1 0 0 )
(1 0 1 0 )
(0 1 0 1 )
(0 0 1 1 )
}
// Transition Time Model Vektor:
// Code:1 = Immidiate; 2= Delay;3 = Exponential; 4 = Equal Distibution;
```

(1 ;1 ;1 ;1 ;

The `TextoARed` class provides us a graphic interface in order to select the text file created by HPSim and convert the information in an object of the `Red` class. The current implementation of the class looks around the file in order to get the incidence matrix and the mark vector. Once we have this parameters, the `construyeRedMatrizIncidencia` method create the Petri net changing the incidence matrix into two matrix (previous incidence matrix and later incidence matrix) in order to create a group of places and transitions from the data that contain them and the mark [13]. The second method must be taken into account because it is needed in order to model a non-pure Petri net[14].

In order to create it manually, a Net subclass must be implemented, for example:

```
public class RedEjemplo extends Red {
  public RedEjemplo() {
    Vector <Estado> estados =
           new Vector <Estado> ();
    Estado lugar0 = new Estado(1);
    estados.add(lugar0);
    Estado lugar1 = new Estado(0);
    estados.add(lugar1);
    Estado lugar2 = new Estado(0);
    estados.add(lugar2);
    Estado lugar3 = new Estado(0);
    estados.add(lugar3);
    Vector < Transicion > transiciones =
           new Vector < Transicion > ();
//Transicion con más de un lugar de entrada o salida:
    Vector<Estado> vEnt = new Vector<Estado>();
    vEnt.add(lugar0);
    Vector<Estado> vSal = new Vector<Estado>();
    vSal.add(lugar1);
    vSal.add(lugar2);
    Transicion trans0 =
         new Transicion(vEnt, vSal);
    transiciones.add(trans0);
//Transicion simple: 1 lugar de entrada y salida:
    Transicion trans1 =
         new Transicion(lugar1, lugar3);
    transiciones.add(trans2);
    Transicion trans2 =
         new Transicion(lugar2, lugar4);
    transiciones.add(trans2);
    Transicion trans3 =
         new Transicion(lugar1, lugar13);
    transiciones.add(trans3);
    vEnt = new Vector<Estado>();
    vEnt.add(lugar3);
    vEnt.add(lugar4);
    vSal = new Vector<Estado>();
    vSal.add(lugar0);
    Transicion trans4 =
         new Transicion(vEnt, vSal);
    transiciones.add(trans0);

    construyeRedVectores(estados,transiciones);
}
```

Therefore nets can be builded quickly and clear. As the Petri net becomes more difficult, it becomes higher the necessity of use the first way of create a Petri net, it must not be forgotten that when you apply Petri net to a specify process, the firing condition of each transition must be implemented, moreover the possible actions associate to the firing of the transition or at the input, output or maintenance of a place, so that the second way of create a Petri net help to avoid confusions caused by the numeration of the transitions in the first mode.

4.3 The Coordinator.

A particular implementation of Petri net has been carried out with the behaviour of control the manufacturing cell. For these, we dispose of the classes "station": `Estacion1, Estacion3, Estacion4 and Transporte`, which implements the Petri net. In order to join together all, we need a "superclass" that carry out the firing control of the Petri net from the different input provided by the sensors of the cell, from this inputs the "superclass" activate the necessary outputs. This class is called "`Coordinador`" Coordinator.

The coordinator will need an object "`Red`" Net that represents the system to control. The *transicionesHabilitadas* vector is a group that form part of the *transitions* vector of the Petri net that belongs to all the net transitions that are sensitized to fire, that is, that his input

places have the enough marks "tokens" in order to carry out the fire if the fire condition is fulfil. The monitor variable take advantage another new functionality of JDK 1.5.0: the `java.util.concurrent` package that contains classes that help us in the concurrent programming, in this case the `ReentrantLock` class that makes possible that only one thread access to the net contains (in order to avoid, for example, simultaneous firing of transitions in conflict). The "r" variable will serve to us to select one transition or another in the case that there were more than one of them sensitized, his firing condition was fulfil, they were not in conflict and they have the same priority.

This Coordinator does not have stations to control; this is because we pretend with this class, as it was planned for the "Red" net class, is to create a father class in order to later implement the different specific coordinators that inherit from this father class. This is the basic structure in java language. The methods defined by this class are:

The begin and finish methods are specific of this project (because of all the coordinators builded here are to control the cell), while the other four methods implement the basic functionality of every Petri net coordinator: find out the possible firing transitions, implement the firing (unmark of input places, mark output places and code associated to the input or output of places) and effective firing of transitions[15]. Due that the present project treat to automatize a real manufacturing process (although in a small scale) it has been taken into account at the time of implement the programs of the different programmable automaton manufacturer based in Petri nets. Over all, due to the laboratory automatons belong to Modicon Telemecanique, it has been taken into account the structure of Grafcet of PL7Pro particularly at the time to design the structure of associated actions at the input, output or support of a place, just as the creation of timers in order to know the measure of time that one place is marked[16].

It has been considerate necessary that the specific coordinator besides of inherit for the `Coordinador`, it implements the `Runnable` interface, that is, he can run as a thread of independent execution. So this class must define a `run ()` method in which the timers associated to this class will initiate.

Due to the cyclical character of the control, the coordinator is executed periodically. So in java we need the `Timer` class. Once take into account this, the decision to take is if we use the `java.util.Timer` class or the `javax.swing.Timer` class. Due to almost all process use graphic interface, we use the second class.

Due to we need activate and deactivate the outputs of the cell stations from the inputs, it will need a way in order to administer these actions. It is going to be carried out by the methods:

```
public boolean condicionDisparo(Transicion t);
public void accionPrevia(Estado e);
public void accionPosterior(Estado e);
public void accionContinua(Estado e);
```

4.3.1 Timers

The use of timers and threads in graphic applications is well documented and a lot of information can be found in the tutorial page of Java [17][18].

To sum up we can say that the Timer object of the `javax.swing` package need as parameter an `ActionListener`, in order to later execute the code associated.

Every 50 milliseconds (according to the `Object.wait()` method) the code associated to the timer is executed, that in this case only read station inputs, updating his value, and change a fixed fields whose only function is to monitorize the process in order to see that the time specification of 50ms is really fulfil. The creation of the other timer,`timerGlobal`, is similar, with the difference that the associated code to this timer will be the one in charge to administer the possible firing of the net, that is, the real coordinator.

The run () method only has that launch the timers:
In order to administer the Petri net it has been selected a method dose not have into account the possible structural conflicts that can have the net. The process that follows is:

1. The continuous action of the marked places is carried out
2. An array with all sensitized transitions is created.
3. It is selected one of the sensitized transitions depending on his priority, or in the case of same priority, randomly.
4. It is checked that his firing condition is fulfil.
5. The transition is fired.
6. The code associated to the output of places and the mark of the output places is executed.

This code does not take into account concepts such us real time, impossible to control working with a Windows NT system.

5.- Products identifier.

The module of the manufacturing cell consist of four stations through which the palet pass with the manufactured piece until that his

departure is produced to the intermediate storage. But in order to control the global manufacturing process it is necessary to know what kind of piece is going to be manufactured and the exact contain of each palet. In order to fulfil this task the manufacturing cell is equiped of a product identifier model IVI-KHD2-4HRX [19]. This identifier let us to place until four read/write headers in each station. Depending on the process carried out in each station, the header will write/read a fixed information in the magnetic disk placed at the bottom of each palet. This information can be used by the following stations when the palet arrive to them in order to act over the information that this magnetic disk contains.

Como ya se ha comentado el módulo de fabricación de la célula de fabricación consta de cuatro estaciones por las que va pasando un palet con la pieza producida hasta que se produce su salida al almacén intermedio. Pero para el control global del proceso de producción es necesario saber que tipo de pieza se va a realizar y cual es exactamente el contenido de cada palet. Es por esto que la célula dispone de un identificador de productos modelo IVI-KHD2-4HRX [19]. Este identificador permite la colocación de hasta cuatro cabezales de lectura/escritura, que se colocan en cada estación. Según el proceso realizado en cada una de ellas se escribirá una determinada información en el disco magnético situado en la parte inferior del palet que podrá ser utilizada por las siguientes estaciones cuando el palet llegue a ellas para actuar de forma coherente con el contenido del mismo.

5.1 Communications protocol.

In order to communicate between the PC and the identifier, the protocol of the user manual must be used [21]. The IVI-KHD2-4HRX let us to connect directly by means of the protocol RS232 by means of a 9 pin connexion wire. Due to read and write of the magnetic disk of the palets is going to be a process that is carried out a few times compareTo to the read or write of inputs or outputs of the stations, we selected the basic read/write operating mode. We can send orders to the identifier of two types with this mode: read/write or system commands.

All the orders that are sent to the identifier must have a fixed structure: for example, in order to read a number of bytes of the header number 3 from the position number 8 the instruction to send must be:

w<HdNo><StAdrH><BytesH><CHCK><ETX>

And the answer in this case that there are not errors is:

w<Status><DB><CHCK><ETX>

In order to work with this in java we have created the class Trama. This class is basically an array of bytes formed depending on the identifier protocol. This is programming problematic because of the strict requisites at the time to create these plots. For example, the initial direction of read or write (StAdrH) must be two numbers in hexadecimal format and converted into 2 bytes to the plot. So the first step was to create a wide group of constants in order to later the building process of the plot is carried out adding bytes to the array of the Trama class. Once we have these constants, is enough with the creation of methods in order to add these plots.

The CHCK is calculated as the sum of all the bytes of the plot except the character end of plot ETX. If the final value of this add is higher than 3 hexadecimal digits the more significant is truncated[22].

Once we have reached join all in a satisfactory way, only remains to create a few common use plots in order to use them directly; it has been done by means of the support class `Comunicaciones` whose only function is to have this plots.

5.2 Passing plots into data.

In order to pass the plots received to a various fields that we can read with the program, we need a class that contains the information necessary: the class Palet.

The basic data to storage in the magnetic disk of the palet are: the last modification date (read or write) and the type of manufacturing piece, if there is a cylinder in the palet, etc... The type of piece is determined by the following constants:

```
public static final byte SINDEFINIR = 0x00;
public static final byte NEGRA = 0x10;
public static final byte NEGRACONTAPA = 0x20;
public static final byte ROJA = 0x30;
public static final byte ROJACONTAPA = 0x40;
public static final byte METALICA = 0x50;
public static final byte METALICACONTAPA= 0x60;
```

The class Palet also provides us methods in order to pass data to a chain of bytes, which can be added after to a Trama, and in order to read the data from a chain of bytes.

To the method Datos it is passed as parameter an object Trama, because of the chain of data that we want to decode will be contained always into a Trama that has been sent from the identifier after of a read request. Besides provide us the methods in order to modify the field in a usual way.

6.- Conclusions and future lines of research.

In the moment of writing this inform the project is in his final phase of implementation, being proved successfully the implementation of the Petri nets in the automatic mode over every station independently of the other stations. Only remains the final step of create the Petri net that controls the cell and the creation of his coordinator.

This project is the base for later projects. For example the extension of the control to the entire cell, the study of manufacturing times of the cell with the new implemented nets, etc...

References:

[1] Prácticas de Informática Industrial. Célula de fabricación flexible. EUITIZ, 2002.
[2] Página web de la célula de fabricación. http://automata.cps.unizar.es/celula.html
En particular, "Modelo, control de tiempos y obtención de prestaciones de una Célula Flexible de Fabricación". (Proyecto fin de Carrera). Pablo Zorraquino Guallar. Universidad de Zaragoza. 2004.
[3] IBS PCI SC/I-T Data Sheet 6039B. www.phoenixcontact.com.
[4] Interbus Club. http://www.Interbusclub.com/.
[5] INTERBUS User Manual. User Interface Version 2.x for High-Level Language Programming of INTERBUS Generation 4 Standard Controller Boards. Phoenix Contact, IBS PC SC HLI UM E. www.phoenixcontact.com.
[6] About java technology. http://www.java.com/en/about/java_technology.jsp/.
[7] Jbuilder X Foundation. www.borland.com.
[8] The Java Tutorial: Java Native Interface. http://java.sun.com/docs/books/tutorial/native1.1/index.html.
[9] Manuel Silva. "Las redes de Petri: en la Automática y la Informática." Editorial AC. ISBN 84-7288-045-1. 2002. p. 16.
[10] Manuel Silva. "Las redes de Petri: en la Automática y la Informática." Editorial AC. ISBN 84-7288-045-1. 2002. pp. 30-34.
[11] Manuel Silva. "Las redes de Petri: en la Automática y la Informática." Editorial AC. ISBN 84-7288-045-1. 2002. p. 22.
[12] Editor HPSim. Copyright (C) 1999 - 2001 Henryk Anschuetz. http://www.winpesim.de/petrinet/e/hpsim_e.htm.
[13] Manuel Silva. "Las redes de Petri: en la Automática y la Informática." Editorial AC. ISBN 84-7288-045-1. 2002. p. 30.
[14] Manuel Silva. "Las redes de Petri: en la Automática y la Informática." Editorial AC. ISBN 84-7288-045-1. 2002. p. 31,.
[15] Manuel Silva. "Las redes de Petri: en la Automática y la Informática." Editorial AC. ISBN 84-7288-045-1. 2002. p. 33,.
[16] Schneider Electric. Manual de PL/7 Pro.
[17] How to use Threads. http://java.sun.com/docs/books/tutorial/uiswing/misc/threads.html.
[18] How to use Swing Timers. http://java.sun.com/docs/books/tutorial/uiswing/misc/timer.html
[19] IVI-KHD2-4HRX DataSheet. http://www.pepperl-fuchs.com
[20] javax.comm page. http://java.sun.com/products/javacomm/
[21] IVI-KHD2-4HRX Manual. p. 12. http://www.pepperl-fuchs.com
[22] IVI-KHD2-4HRX Manual. p. 27. http://www.pepperl-fuchs.com