Implementación programada de Redes de Petri en Java. Control de una célula de fabricación flexible.

Ramón Piedrafita Moreno	José Luis Villarroel Salcedo
Dept. de Informática e Ingeniería de Sistemas	Dept. de Informática e Ingeniería de Sistemas
Universidad de Zaragoza	Universidad de Zaragoza
Maria de Luna,1	Maria de Luna,1
50018 Zaragoza	50018 Zaragoza
piedrafi@unizar.es	jlvilla@unizar.es

Introducción

El presente trabajo se enmarca en el desarrollo de sistemas de control de eventos discretos. Se pretende avanzar en las técnicas de control con Redes de Petri (RdP) de sistemas complejos y realizar también una evaluación del lenguaje Java como plataforma de implementación para estas técnicas.

Un objetivo práctico en este trabajo es la realización de una plataforma de desarrollo abierta, en contraposición a las actuales basadas en Autómatas Industriales. Dicha plataforma permitirá la implementación de diferentes técnicas de control y diferentes formas de gestionar la ejecución de las RdP.

Esta línea de trabajo es una continuación de los trabajos sobre implementación de RdP realizados en la Universidad de Zaragoza. En los más recientes se aborda la implementación de RdP para sistemas de Tiempo Real en lenguaje Ada [5]. En el presente trabajo se ha elegido el lenguaje Java por varias razones:

1. La posibilidad de ejecutar el mismo código en diferentes plataformas.
2. Efectuar una comparación con implementaciones previas realizadas en Ada.
3. Es un lenguaje que está en los primeros pasos en su aplicación al desarrollo de sistemas de control y empotrados.

Se están desarrollando dos líneas de trabajo, una de ellas en Java "clásico" y otra en Java para Tiempo Real. Nos planteamos evaluar Java como lenguaje para la implementación de RdP con tiempo.

En este primer artículo se presentan unos resultados preliminares:

- Se han adaptado a Java varias técnicas de implementación de RdP y se ha desarrollado una nueva, los coordinadores concurrentes.
- Se ha introducido en estas técnicas de implementación el tiempo mediante retrasos asociados a los lugares
- Se ha desarrollado un ejemplo de aplicación real: el control de una célula de fabricación flexible.

El desarrollo de esta plataforma supone en primer lugar la elaboración de un conjunto de clases Java para representar los lugares, transiciones y estructura de la RdP.

Se han elaborado también las tareas encargadas de la ejecución de forma concurrente de la RdP en sincronía con el sistema controlado. La ejecución de la RdP se realizará de forma centralizada por el "thread" (hilo) coordinador o por varios coordinadores en caso de control simultaneo de varias máquinas.

También se han desarrollado las clases que permiten la comunicación con el sistema controlado a través de funciones realizadas en "Java Native Interface" (JNI) [16].

Todo ello ha permitido el desarrollo de un ejemplo práctico: el control de la célula de fabricación flexible del Departamento de Informática e Ingeniería de Sistemas en la Universidad de Zaragoza a través del bus de campo Interbus.

1. Redes de Petri.

Una RdP es una herramienta matemática que puede servir para modelar comportamientos de sistemas de eventos discretos [17]. Las RdP son grafos bipartidos que constan de dos tipos de nodos: los lugares y las transiciones. Los lugares

son representados mediante circunferencias y las transiciones mediante segmento rectilíneos. Los lugares y las transiciones están unidos mediante arcos orientados. Un lugar se puede unir mediante un arco con una transición, y una transición se pude unir con un lugar también mediante un arco. Pero nunca se podrán unir mediante un arco dos lugares o dos transiciones. Un lugar puede contener un número positivo o nulo de marcas.

En el caso de la aplicación de las RdP a la descripción funcional de comportamientos en los sistemas de eventos discretos:

- Un lugar representa una parte del estado a la que puede llegar el sistema. Si un lugar está marcado, significa que es una parte del estado en la que está el sistema en ese momento.
- A las transiciones se les asocian los eventos. El disparo de una transición indica la ocurrencia de un evento que cambia el estado del sistema.

Para el uso de RdP en control de sistemas, se incorpora una interpretación que suele consistir en la asociación de [11]:

- Acciones impulsionales al disparo de transiciones (cambios en el valor de una señal de control, ejecución de un código, …).
- Generación de señales de control asociadas al marcado de lugares (si un lugar está marcado cierta señal estará en nivel alto).
- Predicados que condicionan el disparo de las transiciones sensibilizadas. Estos predicados son función de las entradas al sistema de control o de variables internas.

Una RdP con esta interpretación se le denomina diagrama de marcado o RdP sincronizada (con su entorno), o interpretada para control (su aplicación más difundida). Un ejemplo extendido en control con Autómatas Industriales es el GRAFCET [14].

En la implementación programada que se propone en Java los lugares serán objetos que dispondrán de métodos para efectuar acciones a la activación del lugar, durante el mantenimiento del marcado, y a la desactivación del lugar.

Las transiciones serán también objetos que dispondrán de un método para evaluar su condición de disparo.

Estos métodos podrán leer y escribir variables de entrada/salida y efectuar de esta forma el control del sistema.

2. Implementación programada de redes

Una Implementación programada de una RdP es un programa o algoritmo que ejecuta el disparo de las transiciones de la RdP, respetando las reglas de evolución del modelo.

La problemática de la implementación programada de RdP ha sido estudiada en la literatura en trabajos tales como [6], [9], [4] o [8].

Las implementaciones propuestas pueden ser secuenciales, donde un solo proceso ejecuta la totalidad de la red, como en [10]. También pueden ser concurrentes, dentro de los cuales puede haber centralizadas [15] [9] [8] donde la implementación esta compuesta por diversas tareas encargadas de ejecutar el código o parte operativa del sistema y una tarea denominada coordinador encargada de la parte de control, esto es de ejecutar las reglas de evolución de la RdP.

Las implementaciones concurrentes también pueden ser descentralizadas [9] [4] [8] en las cuales múltiples tareas ejecutan la parte de control de forma distribuida haciendo evolucionar la RdP. Comúnmente, cada una de las tareas en las que se descompone la parte de control, actúa sobre una red secuencial [8] [5].

En las implementaciones desarrolladas en el presente trabajo el programa carga la estructura de la red de un fichero texto proveniente de un editor de RdP. Por lo cual esta implementación es independiente de la red a ejecutar, se trata de un implementación interpretada. En la ejecución de la RdP un “thread” denominado coordinador se encarga de la ejecución del disparo de las transiciones y de actualizar el marcado de la red.

La implementación se puede realizar de forma centralizada con un solo coordinador ejecutando una única RdP encargada de controlar toda la célula de fabricación. La ejecución por el coordinador de la RdP se producirá de forma secuencial.

Como aportación se presenta la utilización de técnicas centralizadas en implementaciones descentralizadas, propuesta pero no desarrollada en [8]. La aplicación puede lanzar varios coordinadores simultáneamente ejecutando una subRdP cada uno de forma concurrente. La ejecución de la RdP en cada uno de los coordinadores se realiza de forma centralizada.

La subdivisión de la RdP global en varias subredes se realiza con criterios de control y de

entrada/salida. Los diferentes coordinadores se encargan de controlar una parte de la célula de fabricación, de la operación local en una estación, del control del movimiento físico del transporte y de la puesta en marcha y paro global de la célula de fabricación

3. La célula de fabricación flexible.

La célula de fabricación flexible está formada por un conjunto de estaciones que permiten la producción y almacenamiento de cilindros neumáticos.

La célula se divide en dos partes: la zona de producción, constituida por las estaciones 1, 2, 3 y 4 y el transporte 1, y la zona de expedición de pedidos con las estaciones 6, 7, los robots y el transporte 2. La estación 5 es el módulo de almacenamiento intermedio.

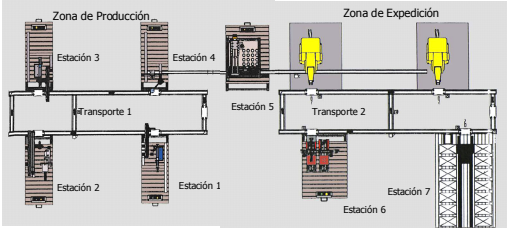

Figura 1. Célula de fabricación flexible

El primer sistema de control implantado en la célula consistió en dotar a cada estación de un autómata programable, interconectados una red industrial en tiempo real. Se decidió instalar un sistema de control alternativo mediante bus de campo Interbus. Se ha implantado el bus Interbus en las estaciones 1, 2, 3 y 4 y el transporte 1. En las estaciones 1 y 4 se han colocado módulos Interbus Inline, mientras que en las estaciones 2, 3 y el transporte 1 se ha optado por módulos TSX Momentum. El maestro del bus Interbus es un PC con una tarjeta IBS PCI SC/I-T, controladora de Interbus de 4ª generación [13].

Cada una de las estaciones de la célula dispone de un cabezal de lectura/escritura de la memoria de los palets donde se realiza la producción, estos cabezales están conectados a un módulo identificador de productos IVI-KHD2-4HRX de Pepperl&fuchs. [12]

La comunicación con el identificador de productos se realiza a través de un puerto serie insertado en el módulo Inline de la estación 1.

4. Implementación en Java

Existen trabajos previos de generación de código Java a partir de especificaciones realizadas con RdP, véase por ejemplo [2] o [3]. En ellos el objetivo es el prototipado y la simulación. Dado que nuestro objetivo es la generación de código Java para el control en Tiempo Real de sistemas, nos hemos decantado por la adaptación de técnicas clásicas (véase la sección 2) desarrolladas para la obtención de aplicaciones eficientes y predecibles, y su utilización en control.

Los primeros pasos abordados en la implementación programada en Java han sido realizar las clases básicas que permiten representar una RdP. También se han desarrollado las clases que permiten conectar con el medio físico y las clases encargadas de la ejecución de la RdP, así como las clases que permiten la comunicación entre los diferentes "threads", los monitores.

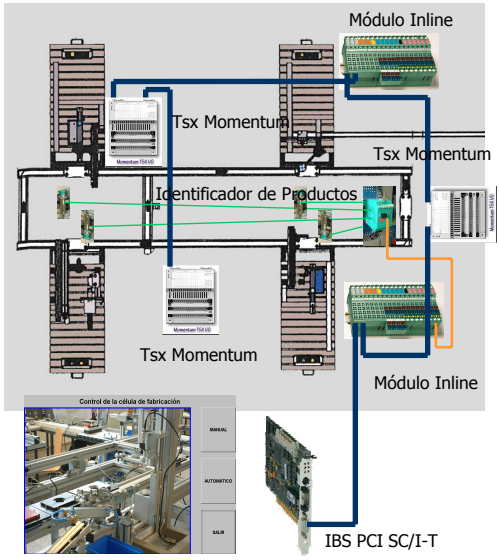

Figura 2. Bus de Campo

4.1. Las clases básicas: Estado y Transición.

La estructura de la RdP quedará definida por un conjunto de estados y transiciones y por las matrices de incidencia previa y posterior. El estado de la RdP quedará definido por los lugares

marcados y por el tiempo de marcado del lugar [11].

```
public class Estado
  int tokens = 0;
  Temporizador temporizador;

public class Transicion
  boolean habilitada = false;
  Vector <Estado> lugaresEntrada;
  Vector <Estado> lugaresSalida;
  int prioridad;
```

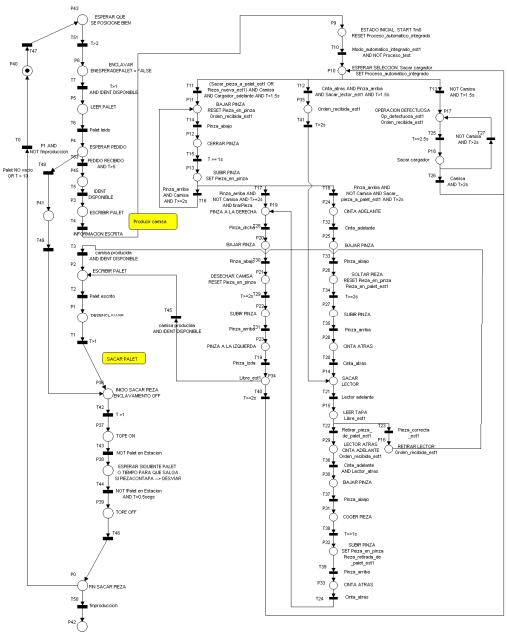

Figura 3. Red de Petri de la Estación 1.

4.2. La clase Red.

La clase Red es la que contiene la información acerca de la estructura de la RdP. La clase Red tiene los siguientes campos:

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

4.3. Creación de una RdP.

Para el prototipado rápido de controladores se dispone de la opción de leer los ficheros generados por el editor de RdP HPsim [7]. Esto permite que la puesta en marcha del controlador se realice de una forma más flexible y rápida que en el caso de definir explícitamente la RdP en un programa en Java. La red creada con HPSim proporciona un archivo de texto con el siguiente formato:

// 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)
....

El archivo de texto es cargado por la aplicación, y a partir de él crea de forma dinámica un objeto de la clase Red. A continuación se crea un "thread" de la clase coordinador, al que se pasa como argumento el objeto de clase Red.

4.4. El coordinador.

La clase coordinador será la encargada de hacer evolucionar el estado de la RdP en sincronía con el sistema controlado. El coordinador será un "thread" encargado de controlar el disparo de las transiciones de la RdP. Para evaluar las condiciones de las transiciones deberá acceder al bus de campo y a la información proveniente de la tarea de Interface Humano-Máquina (HMI). También se encargará de realizar las acciones sobre el sistema físico ejecutando el código asociado a los lugares de la RdP.

El acceso al bus de campo, al identificador de productos y el intercambio de información con la tarea HMI, se realizará a través de monitores que garantizaran la exclusión mutua en el acceso a las

variables. El coordinador será una tarea de ejecución periódica. El periodo de ejecución elegido es de 10 ms, suficiente dada la dinámica del sistema controlado. En cada periodo de ejecución deberá llamar a la función encargada de actualizar los datos del bus. El tiempo de lectura/escritura de los datos de proceso en el bus es inferior a los 2 ms. Por lo tanto los datos de proceso se actualizan de forma síncrona con la tarea de control. La función encargada de actualizar los datos del bus actualizará los valores de las variables en una memoria de imagen de las entradas y volcará la memoria imagen de las salidas sobre el bus de campo. La memoria de imagen de las entradas/salidas reside en los monitores.

La implementación de `Coordinador` posee los siguientes campos:

```
public class Coordinador {
  public Red red;
  public Vector < Transicion >
transicionesHabilitadas;
  private final ReentrantLock monitor
= new ReentrantLock();
```

El coordinador es una clase padre de la que descenderán las clase especifica encargada de del control de toda la célula, en el caso de una implementación centralizada. En el caso de una implementación descentralizada se tendrán varias clases descendientes, encargadas cada una del control de una parte de la célula. Los métodos que define son:

```
public Coordinador(Red r);
protected native void
inicializaComunicacion();
public void
setTransicionesHabilitadas();
public Transicion
getTransicionADisparar();
public void
disparaTransicion(Transicion t);
```

En las clases descendientes se deberán implementar los métodos encargados del "control real":

- Las acciones realizadas en el marcado, desmarcado o mantenimiento del marcado de un lugar de la RdP
- Evaluar la condición de disparo de las transiciones
- creación de temporizadores para conocer y controlar el tiempo de marcado de un lugar.
- Acceso al bus de campo y al identificador de productos

El coordinador al implementar la interfaz Runnable, puede ser lanzado como un "thread" de ejecución independiente. Para que se ejecute de forma periódica en Java se utilizará la clase `Timer`. En Java para Tiempo Real el coordinador se definirá como un Thread de tiempo Real periódico.

Al principio de su ejecución el coordinador carga la RdP, la analiza y crea un vector con las transiciones habilitadas según el marcado inicial. Para ejecutar la RdP el proceso cíclico que se sigue es:

1. Se realiza la acción continua de los lugares marcados.
2. Dentro del vector de transiciones habilitadas se escoge la de mayor prioridad, o en caso de igual prioridad, cualquiera, comprobando que su condición de disparo se cumpla. Esta política también se aplica en caso de conflicto.
3. Se dispara la transición.
4. Se ejecuta el código asociado al desmarcado de los lugares de entrada y al marcado de los lugares de salida.
5. Se actualiza el vector de transiciones habilitadas

Para la búsqueda eficiente de transiciones habilitadas y la actualización de la estructura de datos que las almacena se han propuesto en la literatura diversas técnicas: transiciones sensibilizadas, lugares representantes, lugares representantes dinámicos [11] [9] [8]. En el presente trabajo se han implementado las dos primeras.

5. Ejecución concurrente.

La ejecución de la aplicación de control se puede configurar de dos formas:

- Implementación centralizada. Un solo "thread" encargado de ejecutar la RdP que controla la totalidad de la célula de fabricación.
- Implementación descentralizada: Múltiples coordinadores pueden ser lanzados en concurrencia, realizando cada uno de ellos la ejecución de la subred de Petri encargada de controlar una de las estaciones de la célula. La técnica de implementación en cada coordinador será centralizada.

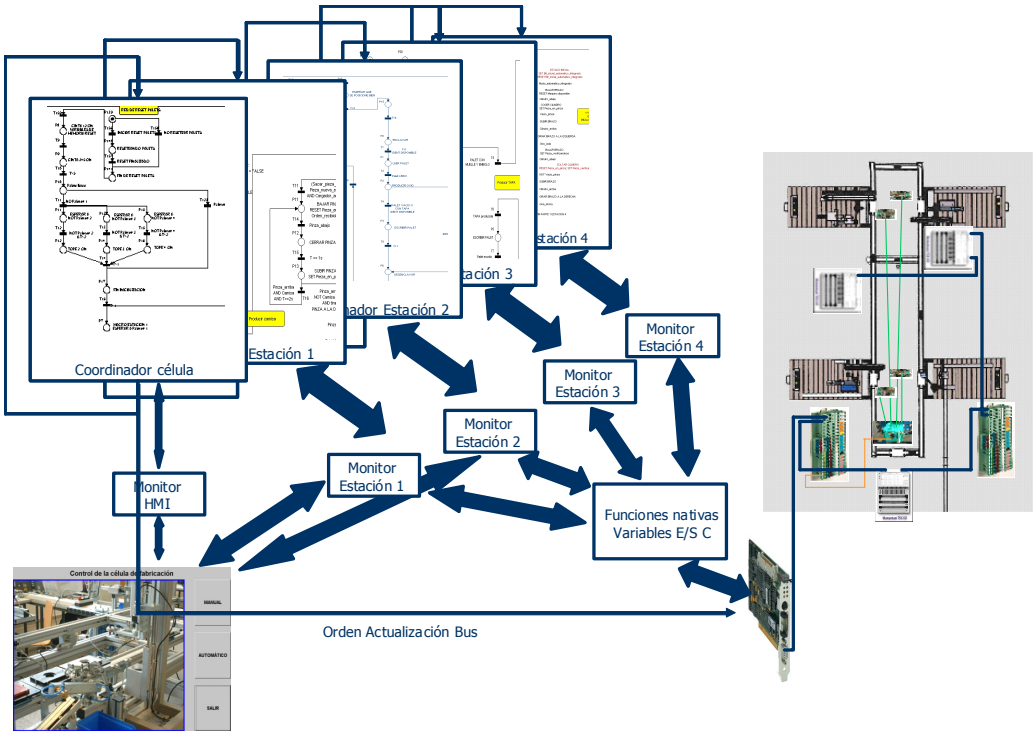

Figura 4. Ejecución concurrente

La comunicación entre los diferentes "threads" de la aplicación se realiza a través de monitores, en los cuales el acceso a las variables se realiza mediante métodos sincronizados. Se han implementado monitores para las estaciones donde "residirán" el valor de las variables de entrada y salida y variables referidas a su estado en el control (automático, manual, fallo…).

También se ha implementado una tarea gráfica con función HMI que se ejecuta en un "thread" de menor prioridad que los "threads" coordinadores. La tarea gráfica también escribe en un monitor las órdenes de mando que el operador envía al sistema de control.

5.1. Acceso al bus de campo

La topología de Interbus es en anillo, no se puede leer una entrada o escribir una salida individualmente, sino que la lectura de todas las entradas y escritura de todas las salidas se realiza en la misma operación por el maestro del bus.

En el caso de la implementación centralizada, el único coordinador que se ejecuta se encarga de llamar periódicamente cada 10 ms a la función desarrollada en JNI encargada de leer todas las variables de entrada del bus y escribir las variables de salida en el bus.

Cuando la ejecución se realice mediante múltiples coordinadores, solamente uno de los coordinadores, el que ejecuta la RdP de coordinación de la célula, se encarga de llamar a la función que actualiza los valores del bus.

Cuando en la ejecución de la RdP se tenga que acceder al valor de una variable de entrada se llama al método sincronizado del monitor de la estación, el cual consulta la memoria imagen de la variable sin acceder al bus. Cuando se deba escribir en una variable de salida esta se escribe en su memoria imagen en el monitor de la estación.

5.2. Acceso al identificador de productos.

Al principio de la operación en una estación de la célula se debe leer la memoria del palet que llega a

la estación para determinar la operación que se debe realizar. Al final de la operación de fabricación se debe actualizar la memoria del palet. Se pueda dar la situación de que varias estaciones quieran acceder a la memoria de sus palets al mismo tiempo.

El identificador de productos es un recurso compartido por todas las estaciones al cual se accede por un puerto serie. Se ha protegido el acceso a este recurso mediante un monitor que garantiza la exclusión mutua en su uso por las diversas RdP que controlan las estaciones.

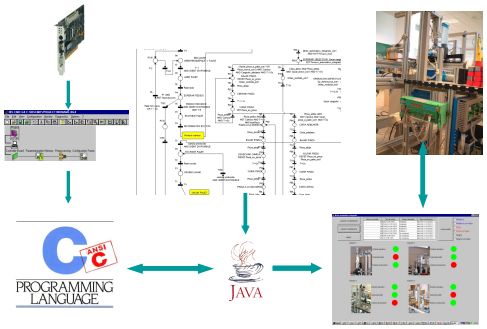

Figura 5. Desarrollo de aplicaciones de control

6. Desarrollo de aplicaciones de control.

Las funciones en lenguaje C se han desarrollado utilizando las librerías "High Level Interface" (HLI) [13] suministradas por el fabricante de la tarjeta maestra del bus. Se han desarrollado las funciones en JNI que trabajan con las funciones en C para acceder al bus.

El prototipado de la RdP se realizará mediante HPSim. El editor HPSim sólo permite definir la estructura de la RdP, la programación de las condiciones de las transiciones y del código a ejecutar en los lugares de la RdP se realizará directamente en Java y se asociarán a los correspondientes objetos lugar y transición.

7. Java como Plataforma de Ejecución

Java posee algunas de las características necesarias para implementar aplicaciones de control, entre ellas la ejecución concurrente de threads. Pero tiene otras características que dificultan su uso a la hora de programar una aplicación de control:

- Es un lenguaje Interpretado. Con la aparición de la técnica compilación "*Just in Time*" se acelera la interpretación de "bytecode".
- Los programas en Java enlazan las clases dinámicamente, existiendo así la posibilidad de que la máquina virtual tenga que descargar clases remotas retardando la ejecución de los programas.
- Se dedica un thread de prioridad máxima a realizar la tarea de recolección de basura.
- El planificador de Java no es "preemptivo" (expulsivo) ni posee prioridades estáticas, proporcionando un segmento de tiempo a todos los hilos que están en ejecución en el sistema ("row robbin").

Estas tres últimas características suponen un grado de impredecibilidad temporal en la ejecución de un programa multiflujo.

Estos problemas han sido abordados en la Especificación de Java para Tiempo Real [18]. En concreto, la plataforma utilizada jRate [1] que proporciona:

- Planificador expulsivo
- No ejecuta "bytecodes" sino que se compila a código máquina, aumentando la velocidad de ejecución
- Implementa los "*RealtimeThread*", "threads" de tiempo real que se puede definir como periódicos, aperiódicos y esporádicos
- Incorpora clases para el tratamiento de eventos asíncronos y para gestionar la transferencia asíncrona del control
- Permite trabajar con relojes de tiempo real de alta resolución
- Implementa *NoHeapRealtimeThread*, "threads" que se pueden ejecutar siempre antes que el recolector de basura.

Todas estas características proporcionan a Java Tiempo Real la predecibilidad temporal necesaria para la ejecución de aplicaciones de control.

8. Conclusiones y futuras líneas de investigación.

El objetivo fundamental de este trabajo ha sido la evaluación del lenguaje Java para la codificación de sistemas de control modelados mediante RdP.

Para ello se ha desarrollado una aplicación práctica que se encuentra actualmente en funcionamiento: el control de una célula de

fabricación del Departamento de Informática e Ingeniería de Sistemas en la Universidad de Zaragoza a través del bus de campo Interbus. De este trabajo cabe destacar:

- La adaptación de los métodos de implementación de RdP a Java no ha planteado ningún problema.
- La orientación a objeto del lenguaje y el que sea multiplataforma han facilitado el desarrollo de la aplicación de control.
- La planificación de tareas que realiza Java impiden que la aplicación construida sea predecible y por lo tanto se descarta el uso de Java "clásico" para la implementación de sistemas de control Tiempo Real.

Actualmente se está finalizando la implementación del sistema de control en Java Tiempo Real para concluir la evaluación del lenguaje. Este trabajo sienta las bases para próximas investigaciones en los campos de la implementación programada de RdP, de los lenguajes y plataformas de ejecución y tiempo real. Próximos pasos serán:

- la migración a sistemas operativos en tiempo real,
- la implementación de RdP con Tiempo en Java para Tiempo Real.
- Aplicación de algoritmos para descomposición de RdP, para implementaciones descentralizadas y distribuidas

9. Agradecimientos

Este trabajo ha sido parcialmente financiado por el proyecto MCyT DPI2003-07986.

Referencias

[1] A.Corsaro. JRate. http://jrate.sourceforge.net/
[2] C. Conway, Ch. Li, M. Pengelly. Pencil. http://www1.cs.columbia.edu/~conway/pencil/
[3] D. Buchs, S. Chachkov & D. Hurzder. Modelling a Secure, Mobile, and Transactional System with CO-OPN. Proc of the third Int. Conf. on Application of Concurrency to System Design. 2003.
[4] D. Taubner. On the implementation of Petri Nets. Advances in Petri Nets 1988, volume 340 of Lecture Notes in Computer Sciences, pages 418-419, Springer-Verlag, Berlin, Germany, 1988.
[5] F.J. Garcia. Modelado e Implementación de Sistemas de Tiempo Real mediante Redes de Petri con Tiempo. PhD thesis, Dpto. de Informática e Ingeniería de Sistemas, Universidad de Zaragoza, Septiembre 1999.
[6] G.W. Brams, editor. Reseaux de Petri. Theorie et practique. Masson, 1983.
[7] Henryk Anschuetz. Editor HPSim. http://www.winpesim.de/petrinet/e/hpsime.htm
[8] J.L. Villarroel. Integración Informática del Control de Sistemas Flexibles de Fabricación. PhD thesis, Dpto. de Ingeniería Eléctrica e Informática, Universidad de Zaragoza, Septiembre 1990.
[9] J.M. Colom, M. Silva, and J.L. Villarroel. On software implementation of petri nets and colored petri nets using high-level concurrent languages. In Proc of 7th European Workshop on Application and Theory of Petri Nets, pages 207-241, Oxford, July 1986.
[10] M. Silva and S. Velilla. Programable logic controllers and petri nets: A comparative study. In Proc. of the Third IFAC/IFIP Symposium on Software for Computer Control 1982, Madrid, Spain, October 1982.
[11] M. Silva. "Las redes de Petri: en la Automática y la Informática." Editorial AC. 1985.
[12] Pepperl&fuchs IVI-KHD2-4HRX DataSheet. http://www.pepperl-fuchs.com
[13] Phoenix Contact. INTERBUS User Manual. User Interface Phoenix Contact, IBSPCSC HLI UM E. www.phoenixcontact.com.
[14] R. David y H. Alla. Petri Nets & Grafcet. Prentice Hall. 1992.
[15] R. Valette, M. Courvoisier, J.M. Bigou, and J. Albukerque. A petri net based programmable logic controller. In Proc. of IFIP Conference on Computer Applications in Production and Engineering, CAP, pages 103-116, 1983.
[16] Sun microsystems. Java Native Interface. http://java.sun.com/docs/books/tutorial/native1.1/index.html.
[17] T. Murata. Petri nets: Properties, Analysis and Applications. Proc. of the IEEE, 77(4), 1989, 541-580.
[18] The Real Time Specification for Java. https://rtsj.dev.java.net/