Un método formal: Redes de Petri y Tiempo Real

Fco. J. García Izquierdo
U. de La Rioja
fgarcia@siur.unirioja.es

José Luis Villarroel Salcedo
U. de Zaragoza
JLVillarroel@posta.unizar.es

1. Introducción

Es de todos conocida la complejidad que presenta el diseño, análisis y puesta en marcha de los Sistemas de Tiempo Real, complejidad que se ve incrementada si consideramos aspectos de fiabilidad, tanto hardware como software. En este tipo de sistemas la fiabilidad es crítica debido a los posibles efectos catastróficos que puede provocar cualquier fallo. Tomando el software como principal objetivo, el uso de métodos formales durante todo el ciclo de vida del sistema puede contribuir a alcanzar el mencionado requisito de fiabilidad al utilizar los mecanismos de verificación formal, tanto temporales como funcionales, que el método proporcione. Si a todo ello añadimos el uso de una herramienta automática para la etapa de codificación, la ventaja es doble, pues a la evidente reducción de los errores de codificación añadimos una reducción de los costes de desarrollo.

El artículo está organizado de la siguiente manera: se comienza con una revisión histórica del formalismo elegido así como la justificación de su uso. Posteriormente se hará una breve descripción del formalismo pasando a mostrar como utilizarlo en el modelado de Sistemas de Tiempo Real. Hablaremos brevemente sobre la implementación automática del modelo antes de pasar a describir un ejemplo real de aplicación de la técnica.

2. Elección del formalismo: las redes de Petri con Tiempo.

Las redes de Petri (de aquí en adelante rdP) han sido ampliamente utilizadas para el modelado y análisis de sistemas de eventos discretos al permitir de manera sencilla el modelado de concurrencia, compartición de recursos, sincronizaciones, ... Además las rdP aportan una consolidada base matemática que facilita la validación y verificación de un amplio conjunto de propiedades de corrección y vivacidad. Una rdP (ver una revisión sobre los conceptos básicos en [Mur89]) es un grafo orientado con dos tipos de nodos; *lugares* (representados por medio de círculos) y *transiciones* (segmentos). Los lugares de la red pueden estar marcados (contienen *marcas*, puntos negros). En el caso más simple, una transición está *sensibilizada* (puede ser disparada) cuando todos sus lugares de entrada están marcados. El marcado de la red, o conjunto de lugares marcados en cada instante, representa el estado instantáneo del sistema modelado. Por tanto, el disparo de una transición provoca cambios en el estado del sistema, ya que retira las marcas de los lugares de entrada de la transición y coloca una marca en cada lugar de salida de la misma. Para que una rdP pueda representar a un sistema es necesario asociar una interpretación a los elementos de la red. De este modo nosotros supondremos que las transiciones representan actividades que son ejecutadas cuando la transición se dispara.

Sin embargo las rdP clásicas no son adecuadas para el modelado ni el análisis de Sistemas de Tiempo Real, ya que no es posible incluir características ni restricciones temporales en el modelo. Para superar esta limitación varios autores han propuesto extensiones que añaden información temporal a las rdP, asociando un valor de tiempo a las transiciones, es decir, a la duración de la actividad que desarrolla. Este valor puede ser fijo (rdP Temporizadas) o aleatorio (rdP Estocásticas). Ninguna de las dos extensiones anteriores es apropiada pues la primera no considera que la duración de las actividades desarrolladas por un Sistema de Tiempo Real pueden no ser fijas, y la segunda realiza un tratamiento aleatorio que no es capaz de garantizar propiedades temporales absolutas.

Es necesario recurrir a las redes de Petri con Tiempo para encontrar una solución a ambos problemas. En las redes de Petri con Tiempo (Time Petri Nets, [MF76], [BD91]) se asocia a cada transición un intervalo temporal que describe un límite superior e inferior del tiempo de disparo de cada transición. Podemos considerar que una rdP con Tiempo es una rdP con etiquetas temporales asociadas a sus transiciones: dos valores temporales [α_i, β_i] son asociados a cada transición, representando α_i el mínimo y β_i el máximo tiempos de disparo de la transición. Los tiempos son relativos al instante de sensibilización (t) de la transición, de modo que la transición se disparará en cualquier instante entre [$t+\alpha_i$, $t+\beta_i$]. Una vez que se decide disparar la transición, el

disparo no consume tiempo, es instantáneo. Para encontrar una descripción formal sobre rdP con Tiempo y sus técnicas de análisis recurrir a [BD91] o [Pop91].

Las rdPT son útiles para el desarrollo de sistemas de tiempo real debido a la posibilidad de modelar sincronizaciones, actividades periódicas, time-outs, concurrencia, ... Utilizando este formalismo durante todo el ciclo de vida del sistema podremos detectar propiedades de mal funcionamiento en las etapas iniciales del ciclo de vida. Es más, el uso de un método formal como las rdPT nos permitirá no tener que restringir la estructura del sistema a analizar. En este sentido se incrementa la flexibilidad en el diseño con respecto a las técnicas clásicas de análisis como *Rate Monotonic* o *Deadline Monotonic Analysis*. En estas aproximaciones, por ejemplo, para permitir el análisis, las comunicaciones entre las tareas periódicas deben hacerse a través de un servidor con entradas sin guardas. El uso de las rdPT evita este tipo de restricciones. Además el uso de un método formal nos permitirá automatizar la generación de código y, por tanto, eliminar los errores de codificación.

3. Cómo modelar.

Considérese el siguiente ejemplo de rdPT:

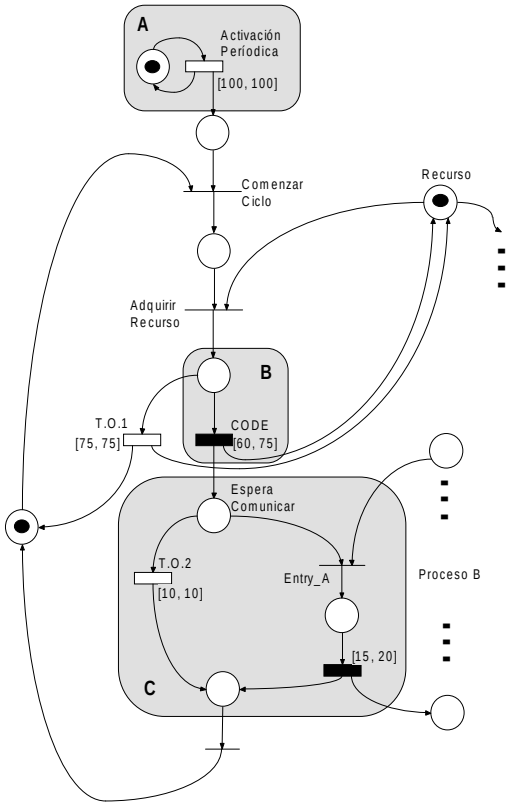


Fig. 1 Fragmento de sistema modelado con una rdPT.

Con ella se trata de modelar un fragmento de un Sistema de Tiempo Real compuesto por un proceso periódico (T= 100 unidades de tiempo). El proceso reserva un recurso necesario para la ejecución de un código (`CODE`) con un time-out ($T.O._1$) de 75 unidades de tiempo. Tras la ejecución del código el recurso es liberado procediéndose al establecimiento de una comunicación con un segundo proceso por medio de una cita (`entry_A`) que también tiene un time-out asociado de 10 u.t. ($T.O._2$). Se adjunta el código Ada para facilitar el entendimiento del modelo:

```
task body Proceso_A is
  next: Time := CLOCK + 100.0;
begin
  loop
    delay until next;
    Recurso.Adquirir;
    select
      delay 75.0;
    then abort
      CODE;
    end select;
    Recurso.Liberar;
    select
      Proceso_B.entry_A;
    or
      delay 10.0;
    end select;
    next := next + 100.0;
  end loop;
end;
```

Se han destacado tres elementos de la figura 1 que corresponden a estructuras típicas en Sistemas de Tiempo Real:

- La caja B (transición `CODE`) muestra un código que debe ser ejecutado por el proceso. La ejecución comienza en el momento en el que se marca el lugar de entrada de la transición y debe acabar en un instante comprendido entre 60 y 75 unidades de tiempo. Cuando la ejecución acaba, la transición es disparada. El código tiene asociado un time-out (transición $T.O._1$).

- La caja A modela la activación periódica del proceso. Cada 100 unidades de tiempo la transición `Activación_Periódica` se dispara provocando la ejecución del resto del proceso.

- La caja C muestra un time-out (transición $T.O._2$) asociado a una comunicación síncrona con el proceso_B. Supongamos que el lugar de entrada a la transición se marca en el instante *t*. Si la transición `entry_A` no se dispara (comienzo de la comunicación) antes de *t+10*, la transición $T.O._2$ se disparará, retirando la marca del lugar `Espera_Comunicar` y abortando el comienzo de la comunicación.

En este artículo supondremos que el lector está familiarizado con los convencionalismos sobre rdP que permiten el modelado de sistemas (ver un estudio

completo en [Sil85]). Para el modelado de Sistemas de Tiempo Real mediante rdP con Tiempo usaremos básicamente las mismas técnicas que para rdP clásicas, salvo algunas modificaciones referidas a las transiciones.

En los Sistemas de Tiempo Real hay diferentes situaciones susceptibles de ser modeladas con transiciones, lo que da lugar a tres tipos de transiciones en el modelo (dependiendo del papel que juega en el sistema):

- Transiciones CODE (CODE-T). Una de estas transiciones, asociadas a su lugar de entrada, representa una actividad desarrollada por el sistema. Esta actividad comienza a ejecutarse en el momento en que la transición es sensibilizada. Este tipo de transiciones están etiquetadas con dos valores temporales [α, β], asociados a la duración de la actividad que representa. En el mejor caso la ejecución del código durará α unidades de tiempo y en el peor β unidades. La finalización de la ejecución está representada por el disparo de la transición. Dibujaremos una CODE-T con un segmento grueso de color negro.

- Las transiciones TIME (TIME-T) son transiciones asociadas a algún evento temporal, como un time-out o la activación periódica de un proceso. Están etiquetadas con un intervalo temporal [α, α], donde α representa el instante, a partir del de sensibilización de la transición, en el que se producirá el evento. El disparo de este tipo de transiciones provoca que se desarrollen acciones de control en el sistema. Así, por ejemplo, si una CODE-T comparte lugar de entrada con una TIME-T que represente un time-out y ésta es disparada, la ejecución del código debe ser abortada (ya que la marca del lugar de entrada es extraída, desensibilizando la transición CODE). Dibujaremos una TIME-T como un segmento grueso de color blanco.

- Las transiciones SYCO (SYCO-T) son transiciones sin significado temporal asociado. Son usadas para modelar sincronizaciones (SYnchronization) y tareas de control (COntrol). El disparo de este tipo de transiciones conduce a simples cambios de estado o es utilizado para sincronizar actividades, modelar eventos no temporales, excepciones, ... Dibujaremos una SYCO-T como un segmento delgado.

Para poder modelar situaciones reales deberemos imponer una serie de restricciones a las rdPT utilizadas en nuestros modelos:

- Cada lugar puede tener simultáneamente como máximo, una CODE-T, una TIME-T y varias SYCO-T como transiciones de salida. De este modo se evita que puedan existir conflictos entre CODE-Ts y/o TIME-Ts (considerar que un proceso no puede simultáneamente ejecutar dos códigos distintos, ni puede, por ejemplo, tener dos time-outs asociados a un código ya que uno anularía al otro). Los conflictos pueden aparecer únicamente entre transiciones SYCO.

- Cada CODE-T tiene un único lugar de entrada. Consideramos que la unidad de ejecución en este tipo de redes es el par compuesto por cada CODE-T y su lugar de entrada. El lugar debe ser 1-limitado ya que sólo consideramos procesos secuenciales sin código reentrante.

- Los valores temporales asociados a las TIME-Ts deben ser iguales, es decir [α,α], ya que representan eventos temporales y éstos deben producirse en un instante determinado.

4. La implementación de la red.

Tras construir el modelo del sistema debe procederse al análisis de sus propiedades, tanto funcionales (vivacidad, limitación, ausencia de bloqueos, ...) como temporales (planificabilidad, cumplimiento de plazos, ...). En cuanto a las primeras debe decirse que los teoremas de verificación y validación de propiedades conocidos para rdP clásicas no suelen ser aplicables a las rdP con Tiempo, de modo que esta etapa del análisis es desarrollada gracias a la obtención del grafo de alcanzabilidad, o *Grafo de Clases de Estado* [BD91], en él cual se recogen todas las posibles evoluciones temporales (estados) del sistema. En él es posible verificar si determinados estados indeseados son o no alcanzables (existen).

Debido a que el sistema debe ser implantado en una plataforma concreta, es necesario imponer las restricciones que ésta incorpora (p.ej., número de procesadores, recursos disponibles, ...). Será por tanto necesario realizar una planificación de la ejecución de las acciones asociadas a las transiciones, de modo que se garantice que todas ellas acaban antes de sus correspondientes plazos de respuesta. El análisis temporal de esta planificación también puede desarrollarse por medio del Grafo de Clases de Estado. En este sentido se incrementa la flexibilidad en el diseño ya que no es necesario simplificar la estructura del sistema para proceder al estudio de su planificabilidad como ocurre en otras técnicas como RMA. Como resultado de esta etapa, se asocia a cada transición una prioridad estática que será tenida en cuenta posteriormente.

Una vez conseguido un modelo del sistema que cumple las especificaciones, e incluso considera las restricciones de implementación, es preciso generar dicha implementación. Una implementación software de una rdP es un programa que simula el disparo de las transiciones (es decir ejecuta las acciones asociadas a éstas) observando las reglas de evolución del marcado.

Por tanto es necesario elegir un lenguaje de programación para proceder con esta etapa. El lenguaje elegido es Ada 95, debido a sus especiales características relacionadas con el tiempo real [BW95] [Bar95]. En esta etapa, tal como se ha comentado previamente, el empleo de una técnica automática de generación de código es múltiplemente ventajoso, ya que facilita la codificación, reduce los errores de programación y, por todo ello, abarata los costes de implantación. Consideraremos para ello adaptaciones de las técnicas clásicas de implementación de rdP que consideren el tiempo aportado por las rdPT. Podemos clasificar las técnicas de implementación de redes en técnicas centralizadas y técnicas descentralizadas [CSV86][Bri96] [Vil90].

En las técnicas de implementación centralizada [CSV86][Vil90][GV96] se consideran dos tipos de procesos (de aquí en adelante hablaremos de tareas Ada cuando nos refiramos a procesos). En primer lugar las *tareas CODE*, que representan la parte operacional del sistema. Habrá una tarea CODE por cada transición CODE en la red encargada de ejecutar las acciones asociadas a esa transición. Cada tarea CODE tiene asociada una prioridad calculada para ella durante la planificación. En segundo lugar, un proceso llamado *coordinador* encargado de la parte de control del sistema y de la supervisión del tiempo. Cada tarea CODE se comunica con el coordinador que es el responsable de decidir cuando se dispara cada transición para lo que tiene en cuenta las distintas prioridades de las transiciones. El coordinador será la tarea más prioritaria de la implementación. Esta arquitectura puede compararse con un sistema operativo en el cual el coordinador hace de kernel y las tareas CODE son los procesos gestionados y ejecutados en él. El reparto del procesador es gestionado por el kernel de Ada teniendo en cuenta, para ello, las prioridades de las tareas de código y el coordinador.

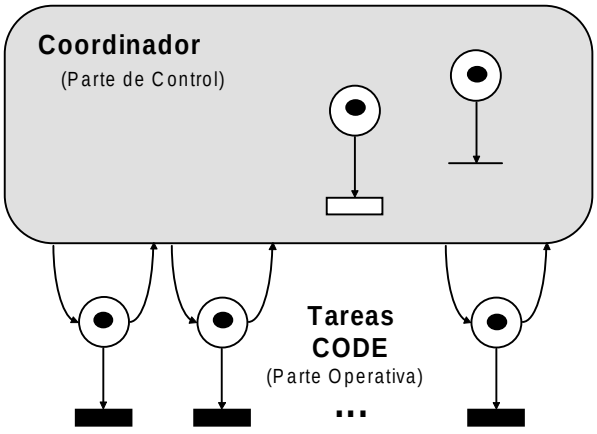


Fig. 2 Arquitectura de implementación centralizada

El coordinador es construido como un intérprete que interpreta una estructura de datos representativa de la red [GV96]. Esta técnica es por tanto especialmente adecuada para el prototipado y la simulación ya que realizar cambios en el sistema supone únicamente cambiar la estructura de datos interpretada por el coordinador. Sin embargo, a pesar de la sencillez en la implementación, esta técnica presenta algunos problemas: en primer lugar el número de tareas concurrentes en la implementación puede ser mucho mayor que la concurrencia real del sistema (ver, p. ej., la red de la figura 3, donde se representan únicamente dos procesos que realizan una cita, mientras que la implementación producirá 6 procesos: 5 tareas CODE y el coordinador). Además, la presencia del coordinador introduce una sobrecarga adicional en el sistema implementado, ya que debe actuar cada vez que una transición es disparada. Por último, la implementación generada es sensible a fallos: debido a que el coordinador es el único responsable de la ejecución de la red, un fallo de éste provoca una indisponibilidad total del sistema.

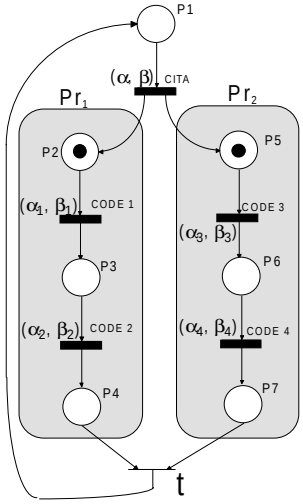


Fig. 3

Todos estos problemas pueden ser solucionados si se utilizan técnicas descentralizadas de implementación. La idea es muy sencilla: la red es dividida en varias subredes secuenciales, cada una de las cuales será implementada en un proceso concurrente con los demás. De este modo se integran en un mismo proceso las partes operacional y de control de cada subred, evitándose el uso del coordinador, reduciendo el número de procesos de la implementación, la sobrecarga y haciendo el sistema más tolerable a fallos, ya que una parte del mismo puede fallar sin que esto signifique la total indisponibilidad del sistema. Evidentemente, el primer paso para la implementación es la obtención de los procesos. Existen para ello técnicas de descomposición que generan una partición de las transiciones de la red en subredes implementables como procesos secuenciales. Los distintos procesos pueden compartir transiciones (lo que supone una comunicación síncrona entre ellos), lugares (comunicación asíncrona) o ambos (p. ej., comunicaciones síncronas en las que se ejecuta un código). Véase, p. ej., el caso de la figura 3, en

la que se pueden obtener dos subredes secuenciales que dan lugar a dos procesos, que se comunican entre si por medio de un lugar (P_1) y dos transiciones `t` y `CITA` (lo que modela una cita entre ellos). Una posible implementación puede verse a continuación:

```
task body Pr1 is
begin
  loop
    CODE1;
    CODE2;
    accept t do
      CITA;
    end;
  end loop;
end Pr1;
```

```
task body Pr2 is
begin
  loop
    CODE3;
    CODE4;
    Pr1.t;
  end loop;
end Pr2;
```

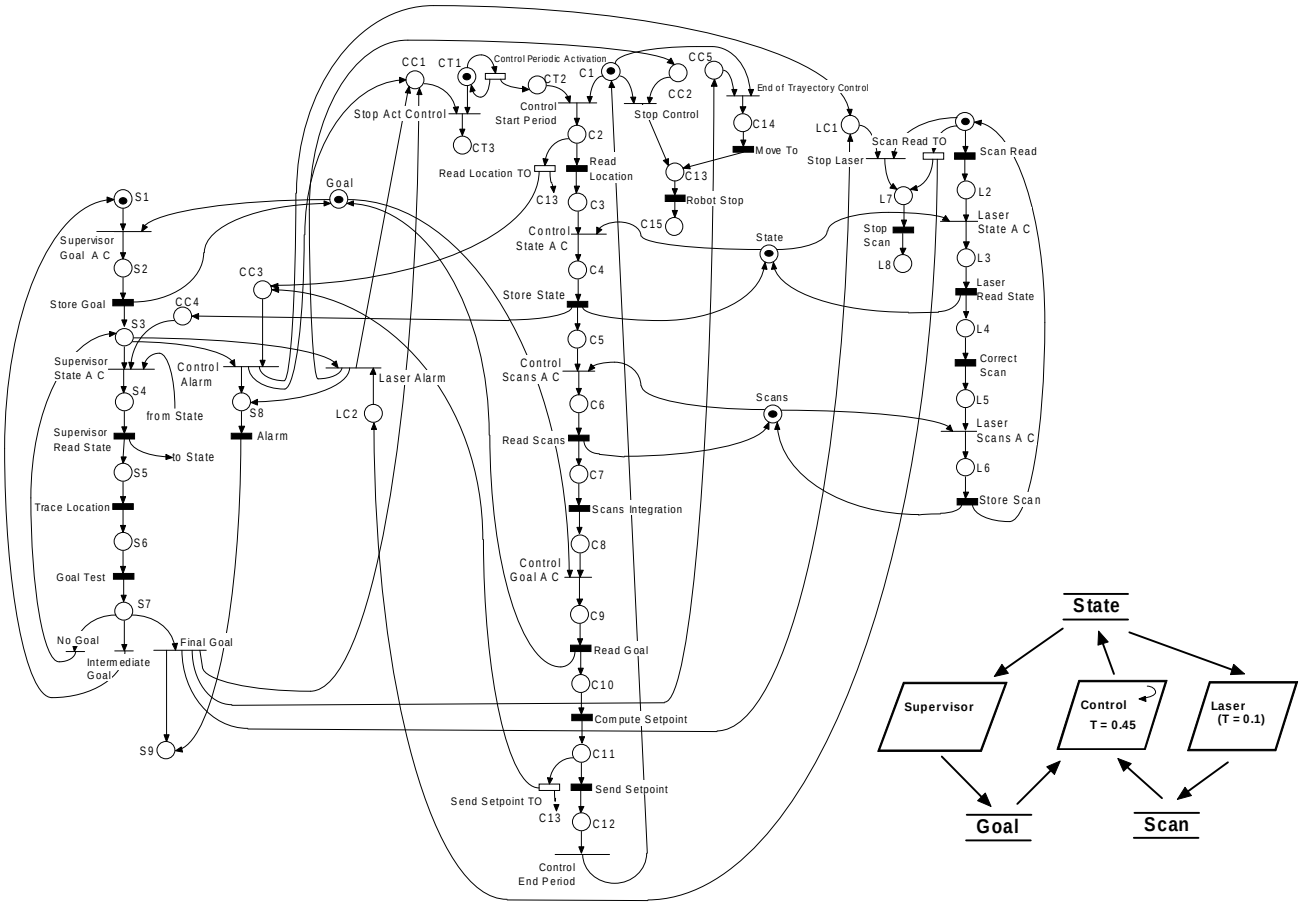
5. Un ejemplo real.

Las técnicas descritas en este artículo han sido aplicadas en el desarrollo del sistema de control de navegación de un robot móvil en tiempo real. Las trayectorias y movimientos programados off-line son modificados en tiempo real para evitar obstáculos, utilizando un comportamiento reactivo. La información acerca del entorno es proporcionada al control del sistema por un láser rotatorio 3D con dos grados de libertad.

El controlador del robot debe desarrollar tres actividades: control del movimiento, supervisión y procesado de los datos de los sensores. Para el control del movimiento se utiliza un proceso periódico (T=0.45 s.) que se comunica con el sistema de control interno del robot. Estas comunicaciones tienen asociados time-outs: 0.2 s. para la lectura de la posición actual y 0.4 s. para el envío de la siguiente posición. Durante la supervisión se debe monitorizar la trayectoria real, comprobar si se ha alcanzado el objetivo final, actualizar las posiciones intermedias si hay obstáculos y gestionar el sistema de alarmas. Por último se cuenta con un proceso periódico que gestiona el láser, que envía un nuevo dato al controlador cada 100 ms. Esta comunicación tiene un time-out de 0.2 s.

La figura 6 muestra la red de Petri que especifica el control del sistema (la información temporal ha sido excluida por claridad). Hemos utilizado la red durante las fases de análisis, validación y planificación del sistema.

Para el prototipado, hemos utilizado la técnica de implementación centralizada, y para la implementación definitiva, la descentralizada. La fase de descomposición en subredes secuenciales evidencia la existencia de cuatro procesos (*Control*, *Supervisor*, *Activador del Control* y *Láser*) acoplados por medio de lugares comunes (implementados como buffers de comunicación asíncrona). A continuación mostramos unos pequeños fragmentos de código de la implementación: el proceso Láser y el lugar *State*.



```
task body Laser is
  Scan_Read_TO: constant duration := 0.2;
begin
L1:
  loop
    select
      LC1.Demark;
      exit L1;
    then abort
      select
        delay Scan_Read_TO;
        exit L1;
      then abort
        Scan_Read;
      end select;
    end select;
    State.Demark;  Laser_Read_State;
    State.Mark;    Correct_Scan;
    Scans.Demark;  Store_Scan;
    Scans.Mark;
  end loop L1;
  Stop_Scan;
end;

protected body State is
  procedure Mark is
  begin
    M:= M + 1;
  end;
  entry Demark when M > 0 is
  begin
    M:= M - 1;
  end;
end;
```

6. Conclusiones

Se ha propuesto un formalismo para todo el ciclo de vida de los Sistemas de Tiempo Real: las redes de Petri con Tiempo. Su utilización acarrea varias ventajas: en primer lugar se trata de un formalismo fácil de entender debido a su naturaleza gráfica; permite la verificación y validación de la corrección del sistema en las etapas iniciales del ciclo de desarrollo; las técnicas de análisis de las rdPT (grafo de clases de estado) incrementan la flexibilidad del diseño, pues ya no es necesario imponer restricciones al sistema para poder analizar su comportamiento ni para verificar el cumplimiento de sus restricciones temporales; además el formalismo es ejecutable, lo que permite el prototipado y la simulación de sistemas.

De especial interés es el hecho de que es posible automatizar la etapa de codificación del sistema a partir de la red que lo modela. Esto elimina los errores de codificación, reduce el tiempo de implantación y por lo tanto reduce el coste del sistema.

Bibliografía

[Bar95] Barnes, A. *Programming in Ada 95*. Addison Wesley, 1995.

[BD91] Berthomieu, B. and M. Diaz. Modeling and Verification of time dependent systems using time petri nets. *IEEE transacctions on Software Engineering*, 17(3):259-273, March 1991.

[BW95] Burns, A. and Wellings, A. Concurrency in Ada. Cambridge University Press, 1995.

[Bri96] J.L. Briz. *Técnicas de implementación de Redes de Petri*. PhD Thesis, Universidad de Zaragoza. María de Luna 3, E-50015 Zaragoza, Spain.

[CSV86] J.M. Colom, M. Silva, J.L. Villarroel. On software implementation of Petri Nets and Colored Petri Nets using high-level concurrent languajes. In *Proceedings of 7th European Workshop on Application and Theory of Petri Nets*, pages 207-241, Oxford, July 1986.

[GV96] Garcia, F.J. and Villarroel, J.L. Modelling and Ada Implementation of Real-Time Systema using Time Petri Nets. In *Proc. of 21st Workshop on Real-Time Programming*, Gramado, Brasil. November 1996. Elsevier Science. In preparation

[MF76] Merlin, P. and D.J. Faber. Recoverability of communication protocols. *IEEE transactions on Communication*, 24(9), September 1976.

[Mur89] Murata, T.. Petri Nets: Properties, Analysis and Applications. In *Proc. of the IEEE*, vol. 77, nº. 4, pp. 541-580, Apr.1989.

[Pop91] Popova, L. (1991). On Time Petri Nets. *J. Inform. Proess. Cybern.*, vol EIK 27, N. 4, pp. 227-244.

[Sil85] Silva, M. *Las redes de Petri en la Informática y en la automática*. AC, Madrid, 1985.

[Vil90] J.L. Villarroel. *Integración Informática del Control en Sistemas Flexibles de Fabricación*. PhD thesis, Universidad de Zaragoza. María de Luna 3, E-50015 Zaragoza, Spain.