Programación Concurrente y Distribuida

Ingeniería Informática
Facultad de Ciencias
Universidad de Cantabria.

Documento:

Cruces con semáforos con RMI

Autores: Laura Barros
José M. Drake

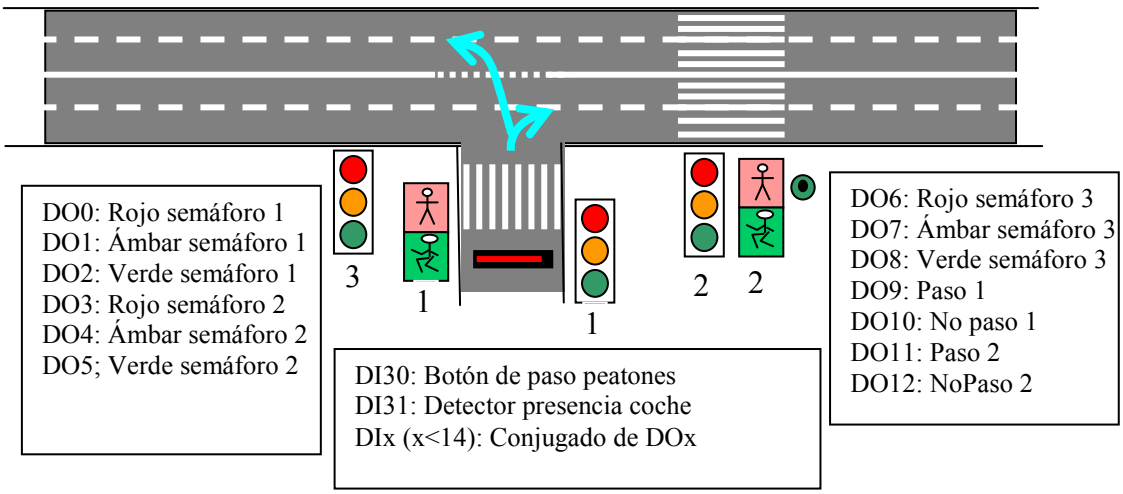
Santander, 14-15 Diciembre 2011

Objetivo de la práctica

El objeto de la práctica es transformar la aplicación de control de cruces de calles, que ya ha sido desarrollada como un programa concurrente, y que se entrega operativa, para que funcione como una aplicación distribuida.

Descripción de la aplicación “CruceConSemáforos”:

La aplicación controla el sistema de semáforos de la intercepción de una vía transversal de bajo tráfico con una vía de alto tráfico.



Como se muestra en la figura, la infraestructura viaria desplegada en el cruce se compone de los siguientes elementos:

Semáforo 1: Bloquea la salida de la vía lenta.

Semáforo 2: Bloquea el paso de la vía rápida para permitir que los peatones crucen la vía rápida por el paso de peatones.

Semáforo 3: Bloquea el paso de la vía rápida para permitir la salida de los vehículos de la vía lenta.

Paso Peatones 1: Indica a los peatones si pueden cruzar o no la vía lenta.

Paso Peatones 2: Indica a los peatones si pueden cruzar o no la vía rápida.

Detector de coche: Esta situado junto al semáforo 1 y Detecta la presencia de un coche en la vía lenta cuando se encuentra parado por el semáforo 1 para salir a la vía rápida.

Pulsador paso de peatones: Esta situado en el semáforo 2, y debe ser pulsado por los peatones que quieren cruzar la vía rápida, a fin de interrumpir el tráfico por ella.

Operación del sistema.

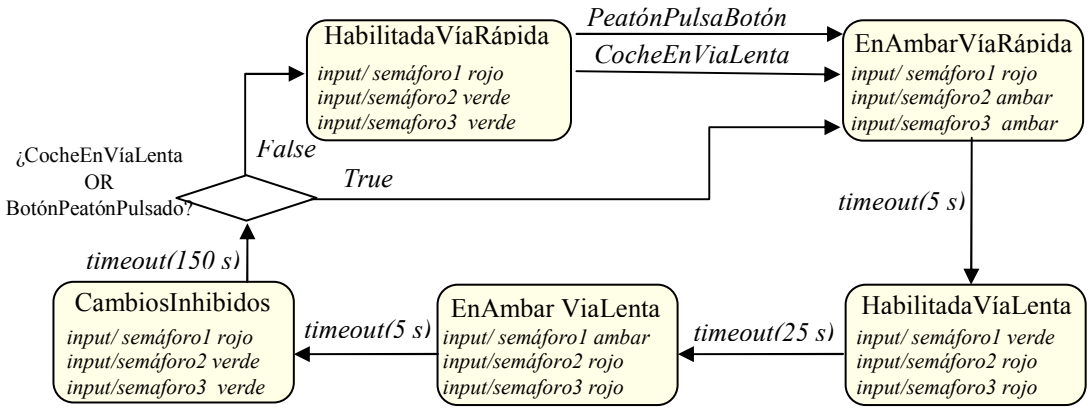
En situación normal, el cruce debe estar establecido para dar paso a los coches de la vía rápida (semáforos 2 y 3 en verde, semáforo 1 en rojo, paso de peatones 1 verde y paso de peatones dos rojo) de forma continuada.

Cuando el detector de presencia de un coche en el semáforo de la vía lenta se activa, o un peatón pulsa el botón de paso de peatones por la vía rápida, se inicia un proceso de cierre del paso de cebra de la vía lenta, y se corta el tráfico de la vía rápida, se da paso durante 30 segundos a la vía lenta, y a los peatones por el paso de cebra de la vía rápida. Pasados estos 30 segundos se recupera el tráfico en la vía de nuevo rápida.

En cualquier caso, la vía rápida no puede estar cerrada más de 30 segundos cada 3 minutos.

El proceso de cierre de un semáforo implica, estar 5 segundos en ámbar antes de pasar a rojo. Igualmente el cierre de paso por un paso de peatones requiere que durante 5 segundos el indicador de paso esté parpadeante (periodo parpadeo 500ms).

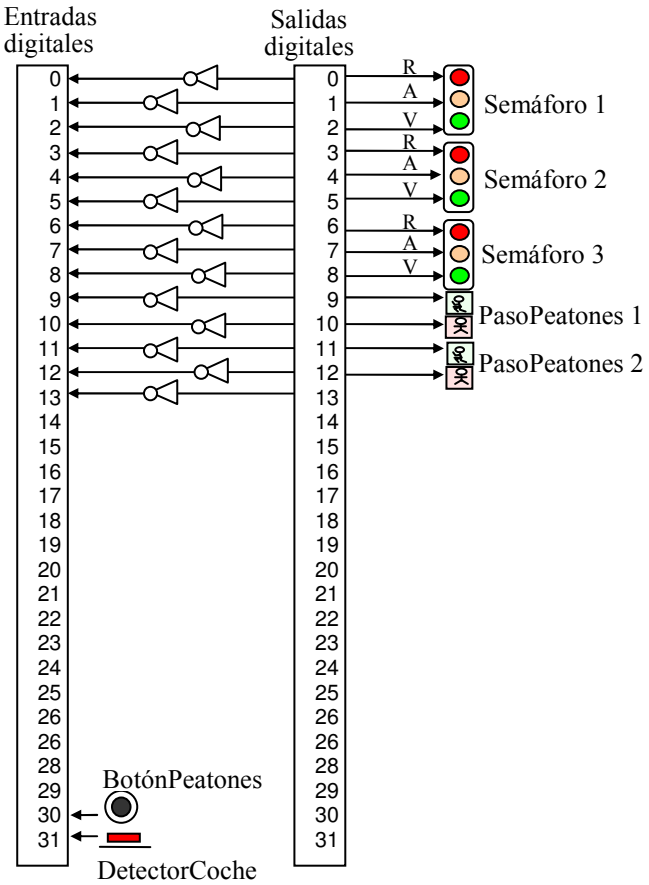
El comportamiento especificado para el sistema se describe mediante la siguiente máquina de estados. En ella los eventos son los timeout, la presencia del coche en la vía lenta y la pulsación del botón de paso de peatón. Las acciones de respuesta son el encendido o apagado de las luces de los semáforos y de los avisos de los pasos de peatones.



Control desde el computador

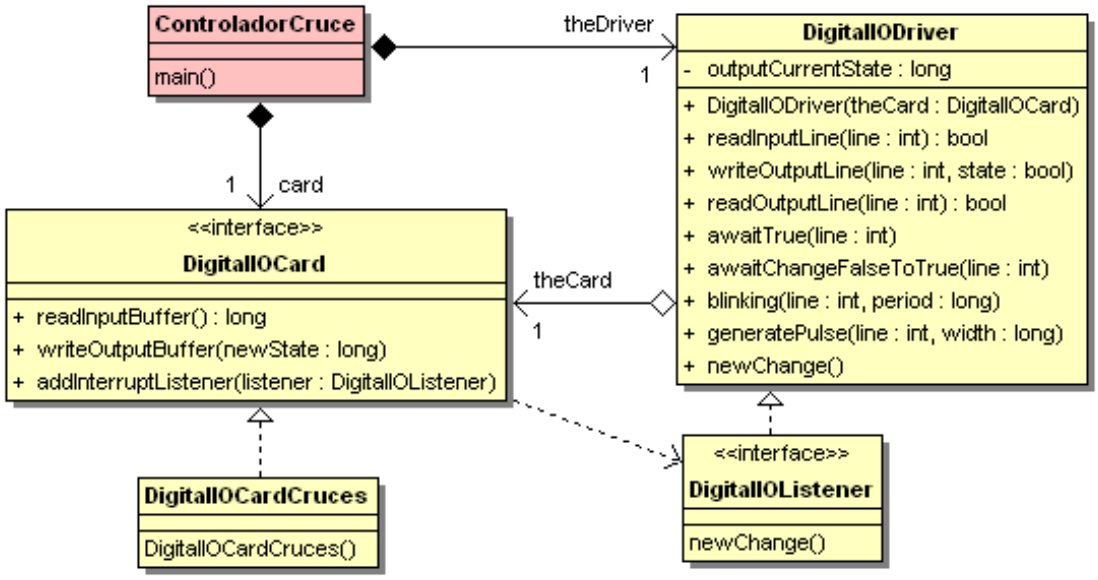
El control del sistema se realiza a través de una tarjeta DigitalIOCard, cuyos entradas y salidas digitales se han conectado al sistema, de acuerdo con el siguiente cableado:

Las 14 primeras líneas de salida D0 a D13 están conectadas a las entradas de igual índices a través de un inversor lógico. La razón de esta conexión es que así se pueden establecer las entradas desde el programa, y así mismo se puede quedar a la espera de que la salida cambie de true a false.



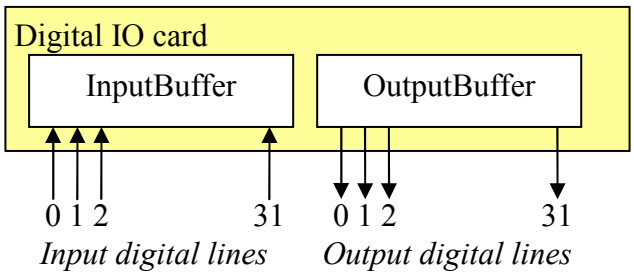
Criterio de diseño

En la figura siguiente se muestran los elementos que participan en la aplicación:



Interfaz DigitalIOCard=>La tarjeta DigitalIOCard permite leer el estado de de 32 líneas de entrada, y así mismo establecer el estado de 32 líneas de salida. Ambos tipos de líneas se numeran de 0 a 31.

A la tarjeta se accede a través de unos registros en los que se mapean los bits de las líneas digitales de entrada y salida. El estado de las 32 líneas de entrada (que se numeran de línea 0 a línea 31) se lee el registro InputBuffer, y el estado de las 32 líneas de salida (que también se numeran de línea 0 a línea 31) se establece escribiendo una palabra de 32 bits en el registro OutputBuffer.



El acceso a estos registros se realiza a través de clases que implementan la interfaz DigitalIOCard, que ofrece los siguientes métodos de acceso:

public long readInputBuffer() => Retorna en los 32 bits menos significativos del valor long, el estado de las 32 líneas de entrada.

public void writeOutputBuffer(long newState) => Establece el estado de las 32 líneas de salida al estado de los 32 bits menos significativos del parámetro *newState:long*.

public void addInterruptListener(DigitalIOListener listener) => Registra el objeto *listener* a su lista de usuarios. Para que el objeto pueda ser registrado debe implementar la interfaz DigitalIOListener, la cual le exige

que tenga definida la operación newChange(). Cada vez que cambia una de las líneas de entrada el objeto DigitalIOCard invoca sobre cada *listener* registrado en él como usuario, la operación newChange().

Clase DigitalIODriver: facilitar a futuras aplicaciones concurrentes el acceso seguro a la tarjeta de IO digital, esto es, leer a través de ella las entradas digitales y establecer las salidas digitales.

DigitalIODriver(card:DigitalIOCard) => Constructor que inicializa el driver y resetea (pone a 0 lógico) todas las líneas de salida, y se declara listener de los cambios en las líneas de entrada de la tarjeta. Se le pasa como argumento la tarjeta que va a controlar, y que debe ser un objeto que implemente la interfaz DigitalIOCard.

public boolean readInputLine(line:int) => Retorna true si la línea digital de entrada de índice *line* está establecida a 1 lógico, y false en el caso contrario. Lanza la excepción IndexOutOfRange si line está fuera del intervalo 0..31.

public void writeOutput(line:int, state:boolean) => Establece la línea de salida de índice *line* al estado lógico *state*, dejando las 31 líneas restantes sin cambiar. Dado que el estado de las líneas de salida no pueden leerse de la tarjeta, el estado de la salida debe mantenerse como una variable de estado en el driver (atributo *outputCurrentState*). Lanza la excepción IndexOutOfRange si line está fuera del intervalo 0..31.

Public bolean readOutputLine(line:int) => Retorna True si la línea de salida de indice line está establecida, y false en caso contrario. Lanza la excepción IndexOutOfRange si line está fuera del intervalo 0..31.

public awaitChangeFalseToTrue(line:int) => Suspende el thread que invoca el método, hasta que en la línea de entrada de índice *line* presenta una transición del estado false al estado true. Lanza la excepción IndexOutOfRange si line está fuera del intervalo 0..31.

public awaitTrue(line:int) => Suspende el thread que invoca el método hasta que la línea de entrada de índice *line* esté en el estado true. Lanza la excepción IndexOutOfRange si line está fuera del intervalo 0..31.

public blinking(line:int, period:int) => Comienza a generar una señal cuadrada en la línea de salida de índice "line"con *period/2* milisegundos en cada estado. Si se invoca con *period=0* representa la cancelación de la generación de la señal cuadrada en la correspondiente línea. Lanza la excepción IndexOutOfRange si line está fuera del intervalo 0..31.

public generatePulse(line:int, width:int) => Genera en la línea de salida de índice *line* un pulso de anchura *width* milisegundos, esto es, con independencia del estado en que se encuentre esa línea, pasará o permanecerá en el estado True dutante *width* milisegundos, y luego pasará a estado false. Lanza la excepción IndexOutOfRange si line está fuera del intervalo 0..31.

public newInputChange() => Será invocada por la DigitalIOCard cuando se produzca en ella un cambio en una de sus líneas de entrada digital. Es una señal de callback, esto es, que será invocada sólo si previamente el

DigitalIODriver se ha declarado como listener cliente de la DigitalIOCard.

Si se invocan una operación (*writeDigitalLine(), blinking() o generatePulse()*) relativa a una determinada línea de salida, queda cancelada la operación que se haya invocado previamente sobre esa misma línea *(blinking() o generatePulse()*).

Clase **DigitalIOCardCruce:** constiene una GUI que emula el sistema que se controla, e implementa la interfaz DigitalIOCard, por lo que puede ser controlada a través de la clase DigitalIODriver ya desarrollada en las prácticas previas. La clase ControladorCruce es la clase que debe ser desarrollada en esta práctica.

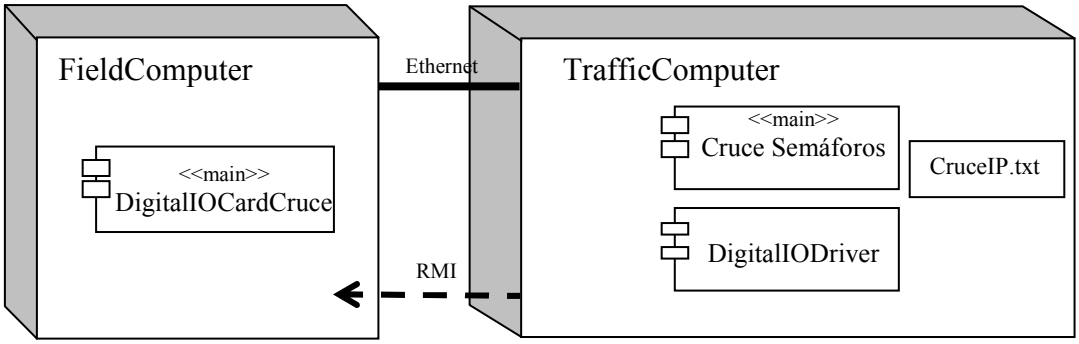
La temporización se realiza con la funcionalidad que ofrece el DigitalIODriver. En cualquier salida se puede generar un pulso de duración prevista. Y si está conectada a una entrada, se puede quedar a la espera de que termine la transición del pulso.

Aplicación Cruce Semáforos

Distribución de la aplicación Cruce Semáforos com RMI

Opción del Miércoles 15-Dic: Distribución del hardware DigitalIOCard.

En el caso se propone la distribución de la aplicación usando rmi, usando el siguiente plan de despliegue:



Criterio de diseño

El sistema debe operar sobre dos computadores:

- FieldComputer: Está localizado en la calle, y que dispone de las tarjetas de entrada/salida digital, que controlan las líneas físicas que leen los sensores y encienden y apagan las luces. En el se va a instanciar el objeto de la clase DigitalIOCardCruce que a través de su interfaz DigitalIOCard permite que una aplicación externa pueda leer líneas digitales de entrada, establecer el estado de líneas digitales de salida, y notifica al cliente que espera los cambios.
- TrafficComputer: Esta localizado en el centro de control de tráfico de la ciudad, y desde el se controla todo el tráfico de la ciudad, y en particular el cruce que se controla en esta práctica. En el se instancian todos los demás objetos que constituyen la aplicación de control del cruce.

Paso de objeto callback por parámetros.

A través del método *addInterruptListener*(DigitalIOListener listener) que ofrece el objeto DigitalIOCardCruce, los escuchantes mandan su referencia remota (callback), es decir, los objetos de tipo DigitalListener, interesados en recibir los cambios de la tarjeta digital. De esta forma, el objeto DigitalIOCardCruce, no necesitará buscar en el registro la referencia remota del DigitalIODriver.

Nota: En esta práctica hay que modificar:

- La clase **DigitalIOCardCruce**: se debe convertir en un servidor RMI que atienda las peticiones del Driver. Además es cliente, ya que ejecuta el método newChange() para avisar al driver de cambios producidos en la tarjeta.

- La clase **DigitalIODriver**: se debe convertir en un programa que constituya un cliente del servidor ofrecido por la instancia DigitalIOCruce. Esta clase lee el fichero "CrossingIP.txt" que contiene el IP del procesador FieldDriverComputer. Además es servidor, ya que ofrece el método newChange().

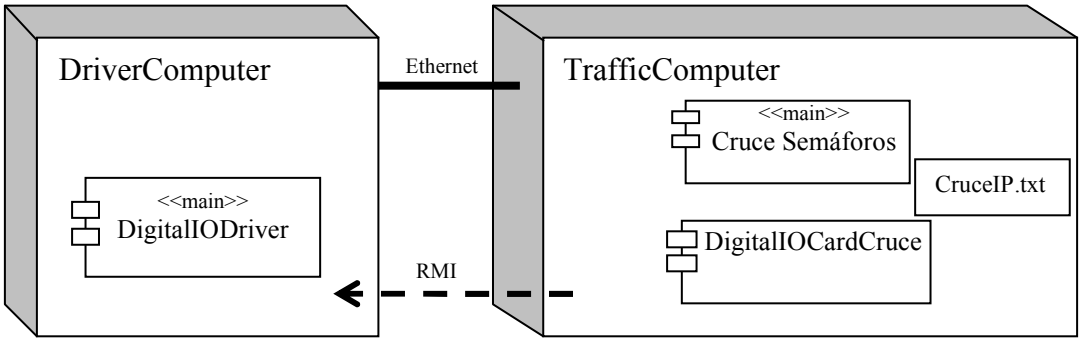
  Ejemplo de contenido del fichero CrossingIP.txt

  ```
  127.0.0.1
  ```

Distribución de la aplicación Cruce Semáforos con RMI

Opción del Jueves 16-Dic: Distribución del driver DigitalIODriver.

En el caso se propone la distribución de la aplicación usando rmi, usando el siguiente plan de despliegue:



Se considera que el driver de la tarjeta se encuentra distribuido, mientras que el resto, se encuentra instalado en un único computador.

Criterio de diseño

La distribución afecta al Driver, esto es, el objeto relativo al Driver, queda instanciado en el procesador DriverComputer, mientras que las clases de los servicios comunes se instancian en el procesador TrafficComputer.

Paso de objeto callback por parámetros.

A través del método *addInterruptListener*(DigitalIOListener listener) que ofrece el objeto DigitalIOCardCruce, los escuchantes mandan su referencia remota (callback), es decir, los objetos de tipo DigitalIOCardListener, interesados en recibir los cambios de la tarjeta digital. De esta forma, la tarjeta DigitalIOCardCruce, no necesita buscar en el registro la referencia al objeto remoto del DigitalIODriver.

Nota: En esta práctica hay que modificar:

- La clase **DigitalIODriver**: se debe convertir en un servidor RMI que atienda las peticiones del CrossingController. Además es un cliente de los servicios de la tarjeta.
- La clase **DigitalIOCard**: se debe convertir en un servidor RMI que atienda las peticiones del driver.
- La clase **CruceSemaforos**: se debe convertir en un programa que constituya un cliente del servidor ofrecido por la instancia DigitalIODriver. Esta clase lee el fichero "CrossingIP.txt" que contiene el IP del procesador DriverComputer.

  Ejemplo de contenido del fichero CrossingIP.txt

  ```
  127.0.0.2
  ```

- El programa principal CruceSemáforos no debe instanciar el driver.