Arduino

Unidad 4. Sistemas Reactivos

José Luis Poza Lujan Sergio Sáez Barona

Tecnologías Interactivas y Fabricación Digital
Universidad Politécnica de Valencia

Noviembre 2015

Índice

José L. Poza Lujan
Sergio Sáez Barona

Índice

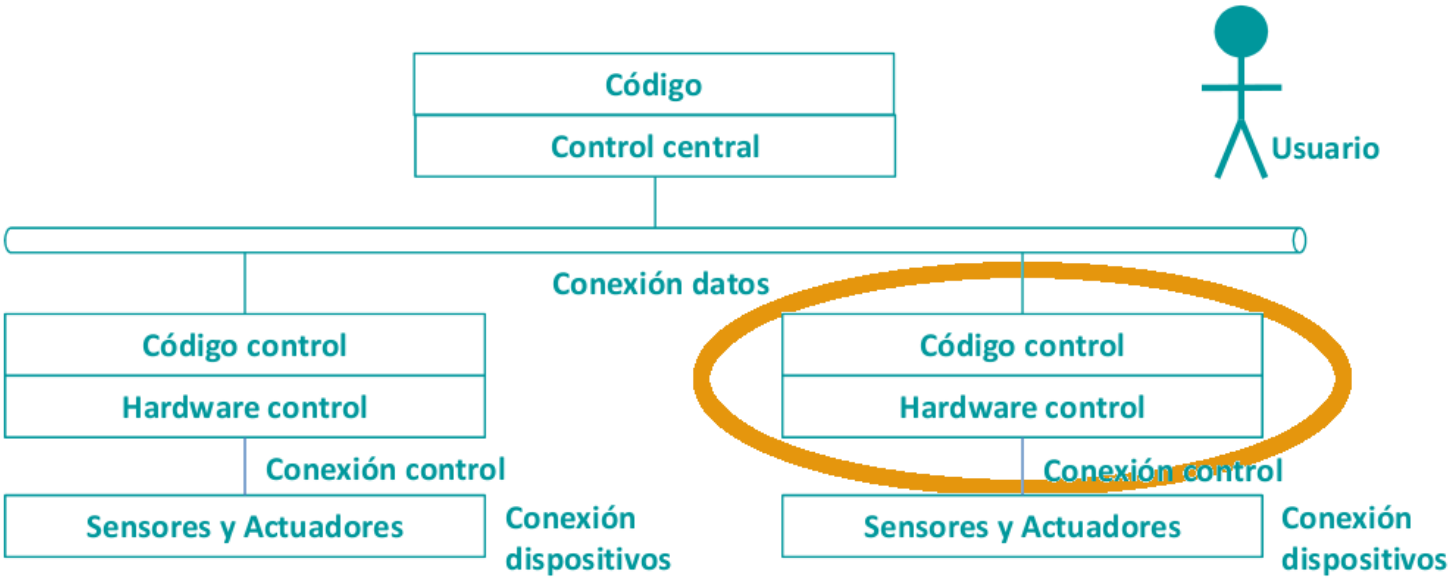
José L. Poza Lujan
Sergio Sáez Barona

Objetivos

- Comprender el diseño y la estructura de un sistema reactivo.
- Aprender a diseñar los nodos de un sistema de control distribuido.
- Comprender los diferentes niveles de *inteligencia* dentro un sistema de control distribuido.
- Aprender a implementar sistemas reactivos con Arduino.

José L. Poza Lujan
Sergio Sáez Barona

Arquitectura de un sistema distribuido



José L. Poza Lujan
Sergio Sáez Barona

Naturaleza de un nodo de control

Un nodo de un sistema de control distribuido recibe:

- estímulos externos del entorno a través de los sensores conectados directamente al nodo ...
  - mediante interrupciones,
  - mediante el muestreo periódico de los sensores,
- mensajes del *servidor de control* a través de la red de datos del sistema de distribuido,
- mensajes de otros nodos a través de las redes de control,
- peticiones de los controles remotos a disposición del usuario (infrorojos, RF, bluetooth, ...),

... y debe reaccionar a dichos eventos.

$\Rightarrow$ Se puede modelar como un sistema reactivo.

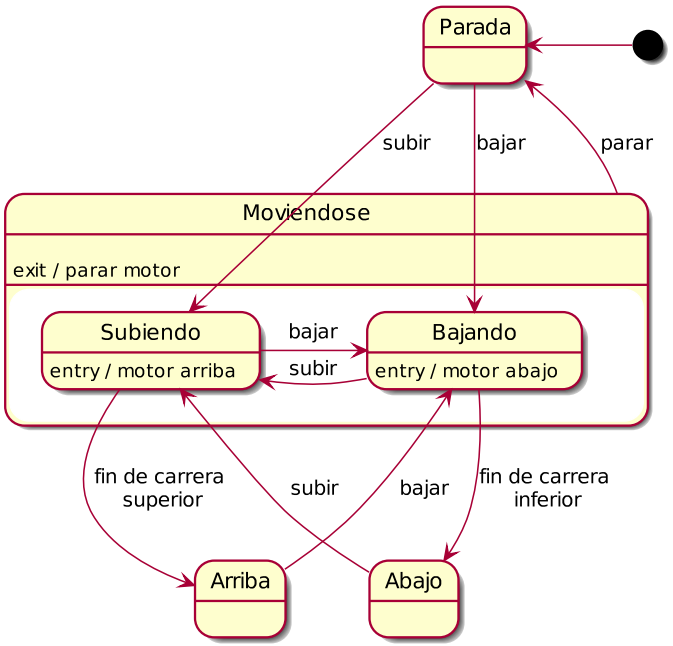
José L. Poza Lujan
Sergio Sáez Barona

Índice

José L. Poza Lujan
Sergio Sáez Barona

Definición de Sistemas Reactivos

- Un sistema reactivo es aquel que mantiene una continua interacción con su entorno, respondiendo ante los estímulos externos en función de su estado interno [wikipedia].
- Son sistemas complejos, pero que se pueden diseñar utilizando herramientas estructuradas como las máquinas de estados finitos (UML).



José L. Poza Lujan
Sergio Sáez Barona

Máquina de estados UML

- Es una extensión orientada a objetos de las máquinas de estados de Harel.
- Añade los estados jerarquicos, las regiones ortogonales, las acciones de entrada y salida, etc.
- Permite simplificar la programación de los sistemas reactivos haciendo que el manejo de cada evento dependa no sólo del evento en cuestion, sino también del estado y de condiciones sobre el estado interno.
- Una vez diseñado el sistema, la traducción del diagrama de la máquina de estados al código es *bastante* mecánica.

José L. Poza Lujan
Sergio Sáez Barona

Elementos principales

Un nodo del sistema que se comporta como un sistema reactivo se puede modelar mediante los siguientes elementos:

Estados

Describen una situación que perdura durante un periodo de tiempo del sistema.

Eventos

Describen los estímulos externos que pueden afectar al comportamiento del sistema.

Transiciones

Describen la respuesta del sistema a la ocurrencia de un evento.

Acciones

Modelan la actividad del sistema.

José L. Poza Lujan
Sergio Sáez Barona

Estados

Estado simple

Representa un estado *cualitativo* del sistema. El sistema debe estar en al menos uno de estos estados.

Estado compuesto

Permite agrupar varias estados, pero sólo uno de los subestados puede estar activo.

Estados concurrentes

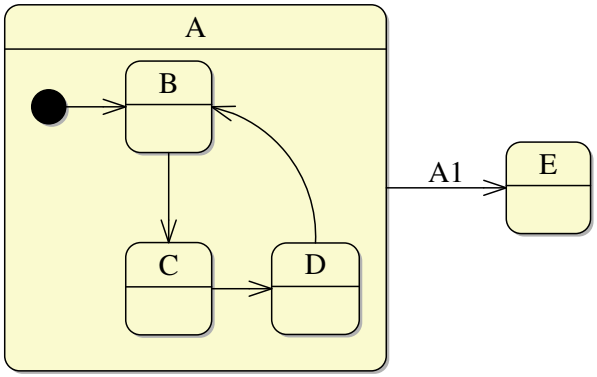
Permite agrupar varios *Estados Compuestos* o regiones, estándo activo un subestado de cada una de las regiones.

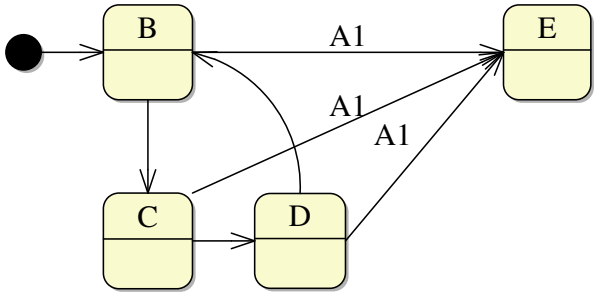
Pseudo-estados

Representan algunas elementos especiales del modelo del sistema.

José L. Poza Lujan
Sergio Sáez Barona

Estados compuestos





- El uso de estados compuestos o anidados permite definir la reacción ante un evento para un conjunto de estados.
- Elimina el problema de explosión de transiciones.
- En el ejemplo, si el sistema está en el estado es `B`, `C` o `D` y llega el evento `A1` el sistema cambia al estado `E`.

José L. Poza Lujan
Sergio Sáez Barona

Pseudo-estados

Estado inicial

Una transición desde este estado apuntará al estado inicial o al subestado de entrada de un estado compuesto.

Estado final

Una transición hasta este estado indica el fin del la actividad en el nivel en que se encuentra: un estado compuesto o el sistema completo.

Estado histórico

Similar al estado inicial salvo que al salir de un estado compuesto éste recuerda en qué subestado se encontraba y ese será el estado inical cuando se vuelva a entrar al estado compuesto.

Bifurcación

Trás este estado se produce una transición a alguno de los posibles estados de destino, dependiendo de las guardas.

José L. Poza Lujan
Sergio Sáez Barona

Eventos

Existen varios tipos de eventos:

Signal event Se envía el evento al subsistema de forma asíncrona.
El remitente no espera a que el evento se procese.

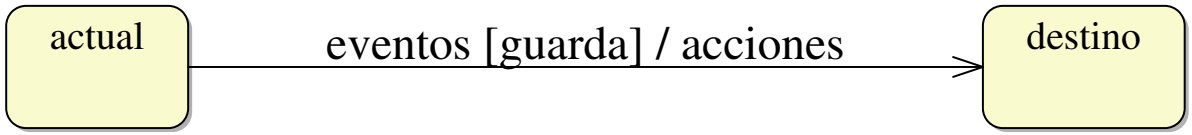
Call event El remitente y el destinatario se sincronizan.
El remitente no recupera el control hasta que el destinatario ha procesado el evento.

Time event Se dispara cuando un temporizador asociado a una transición expira. Se representa como `after N`

Completion event Se genera cuando se completa la actividad asociada a un estado.
Se utilizan para disparar las transiciones que sólo tienen una condición asociada (ver después).

iFab Tecnologías Interactivas y Fabricación Digital
Universitat Politècnica de València
José L. Poza Lujan
Sergio Sáez Barona

Transiciones



- Una transición viene determinada por:

  eventos Indica qué eventos externos[1] pueden disparar la transición.

  guarda Indica que condición se debe cumplir para que la transición se dispare.

  acciones Indica que acciones se deben llevar a cabo antes de cambiar al estado de destino.

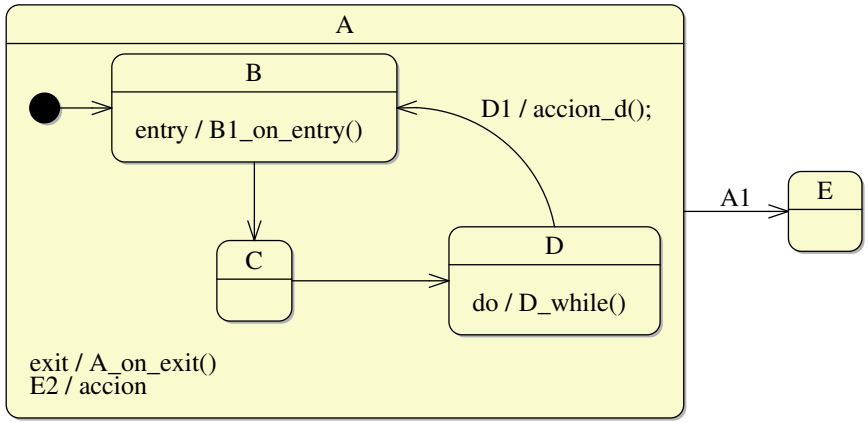
  destino El estado de destino de la transición. Si no hay un estado de destino, es una transición interna.

- Todos los elementos son opcionales, pero *suele* haber alguno de ellos.

[1] Lista de eventos separados por comas (,)

José L. Poza Lujan
Sergio Sáez Barona

Acciones



Transición Determina la acción asociada a una transición.

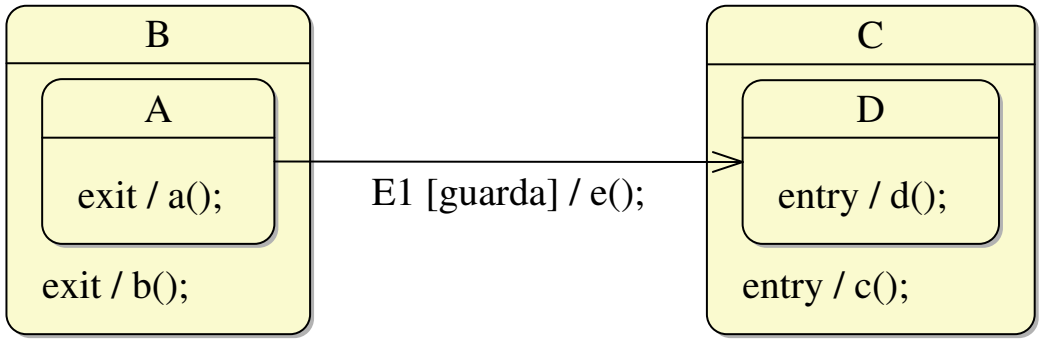
Entrada (`entry`) Permite definir la acción que se debe realizar al entrar en un estado.

Salida (`exit`) Permite definir la acción que se debe realizar al salir de un estado.

Actividad (`do`) Indica la acción que se debe ejecutar mientras se permanece en un estado.
¡Ojo! La acción se aborta al salir.

José L. Poza Lujan
Sergio Sáez Barona

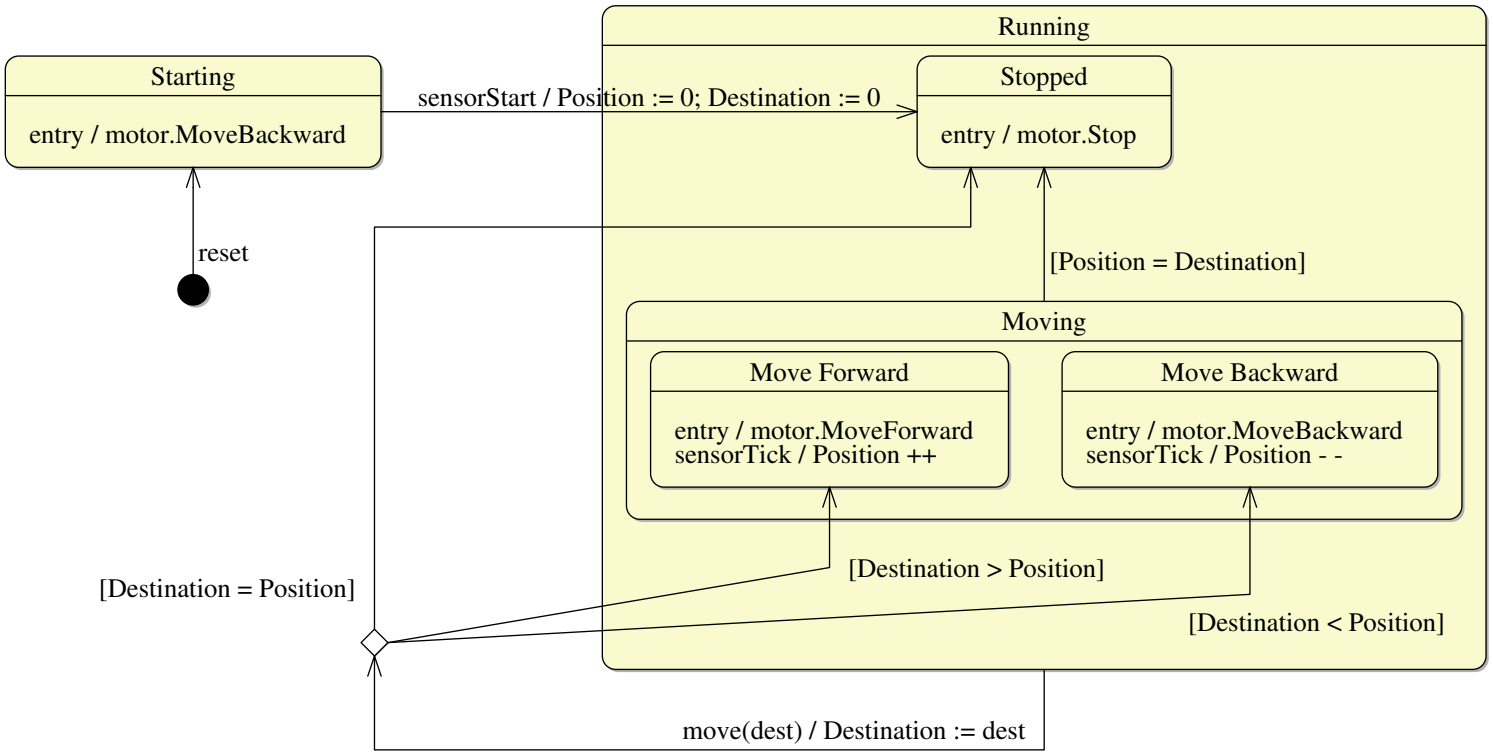
Secuencia de ejecución de una transición



- UML especifica que el comportamiento ante un evento es *run-to-completion*, es decir, una vez se comienza a procesar un evento no se atiende ningún otro hasta que se haya terminado de procesar.
- En el ejemplo, si el sistema está en el estado `A`, se produce el evento `E1` y se cumple la condición `guarda` se ejecutará:
  `a()` → `b()` → `e()` → `c()` → `d()`.

José L. Poza Lujan
Sergio Sáez Barona

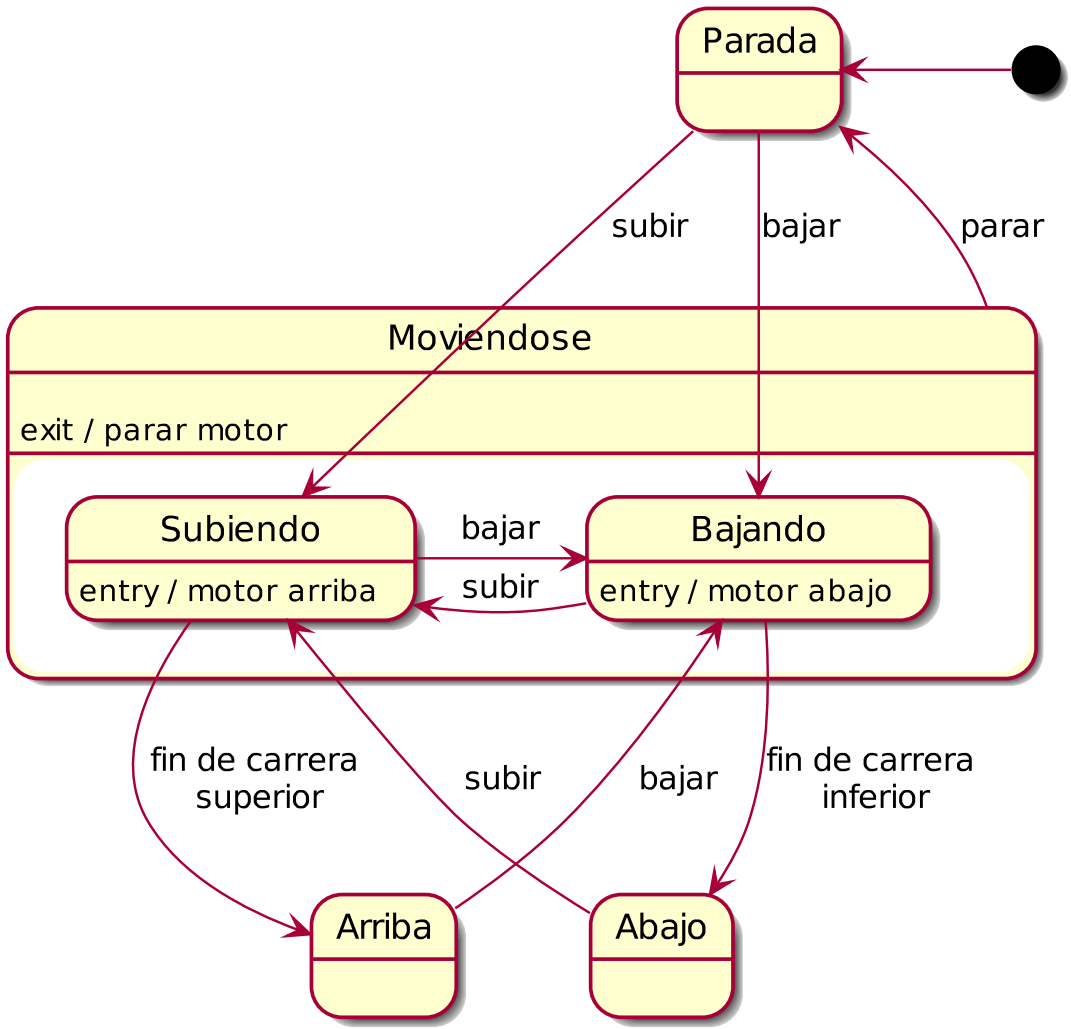
Ejemplos
Motor de un robot



José L. Poza Lujan
Sergio Sáez Barona

Ejemplos

Control de un motor de persiana



José L. Poza Lujan
Sergio Sáez Barona

Índice

José L. Poza Lujan
Sergio Sáez Barona

Diseño de los Nodos de Control

Utilizar las FSM con alguna restricción:

- Organizar el control como varias FSM:
  1. una FSM con la *inteligencia* de nivel superior que indica los **modos de funcionamiento**: *invierno*/*verano*, *noche*/*día*, ... y envía eventos a la otra FSM.
  2. otra FSM con el control fino de los niveles inferiores que indica el **estado interno** y la acción en curso: *bajando persiana*, *a la espera*, *movimiento detectado*, etc.
- Todos los eventos externos, eventos temporales, los mensajes del servidor o de otros nodos se modelan como un **evento interno**.

José L. Poza Lujan
Sergio Sáez Barona

Etapas del bucle de control de un nodo

1. Transformar los estímulos externos, comunicaciones, etc. en **eventos internos** dependiendo del estado actual.
   $\Rightarrow$ Ignorarlos si no son relevantes.
2. Llevar a cabo las **acciones** pertinentes en función del evento y del estado.
   Determinar si hay un **cambio de modo/estado**.
3. Informar al servidor central del nuevo estado, si fuera pertinente.

Dado que las diferentes comprobaciones de los estímulos, mensajes, el estado de las entradas utilizadas en las condiciones y/o las acciones a realizar pueden aparecer en múltiples estados y transiciones deben encapsularse en **funciones**.

José L. Poza Lujan
Sergio Sáez Barona

Estructura de un nodo

Esquema básico del bucle de control

```
void loop () {

  evento= determinar_evento();

  switch (estado) {
    case ESTADO_1:
      switch (evento) {
        case EVENTO_1:
          acciones_salida();
          accion_transicion();
          acciones_entrada();
          estado= NUEVO_ESTADO;
          break;
        case EVENTO_X:
          ...
          break;
        default:
      } /* end switch */
      break;
    case ESTADO_2:
      ...
      break;
    ...
  } /* end switch */

  enviar_estado(estado);

} /* end loop */
```


José L. Poza Lujan
Sergio Sáez Barona

Estructura de un nodo (cont.)

Definición de los estados (defines)

```
#define EST_INICIAL                 0
#define EST_PARADA                  1
#define EST_MOVIENDOSE              2
#define EST_MOVIENDOSE_SUBIENDO     3
#define EST_MOVIENDOSE_BAJANDO      4
#define EST_ARRIBA                  5
#define EST_ABAJO                   6
```

Definición de los estados (enum)

```
typedef enum {
  EST_INICIAL,
  EST_PARADA,
  EST_MOVIENDOSE,
  EST_MOVIENDOSE_SUBIENDO,
  EST_MOVIENDOSE_BAJANDO,
  EST_ARRIBA,
  EST_ABAJO
} estado_t;
```


iFab Tecnologías Interactivas y Fabricación Digital
UNIVERSITAT POLITÈCNICA DE VALÈNCIA
José L. Poza Lujan
Sergio Sáez Barona

Estructura de un nodo (cont.)

Definición de los eventos (defines)

```
#define EV_NINGUNO              0
#define EV_PARAR                1
#define EV_SUBIR                2
#define EV_BAJAR                3
#define EV_FIN_CARRERA_SUP      4
#define EV_FIN_CARRERA_INF      5
```

Definición de los eventos (enum)

```
typedef enum {
  EV_NINGUNO,
  EV_PARAR,
  EV_SUBIR,
  EV_BAJAR,
  EV_FIN_CARRERA_SUP,
  EV_FIN_CARRERA_INF
} evento_t;
```


José L. Poza Lujan
Sergio Sáez Barona

Estructura de un nodo (cont.)

Determinar el evento (eventos exclusivos)

```
void loop() {
  evento= EV_NINGUNO;

  if (evento == EV_NINGUNO) {
    activo= leer_final_carrera_sup();
    if (activo)
      evento= EV_FIN_CARRERA_SUP;
  } /* endif */

  if (evento == EV_NINGUNO) {
    activo= leer_final_carrera_inf();
    if (activo)
      evento= EV_FIN_CARRERA_INF;
  } /* endif */

  if (evento == EV_NINGUNO) {
    orden= leer_orden_remota();
    if (orden != NINGUNA)
      evento= extraer_evento_orden(orden);
  } /* endif */
  ...
} /* end loop */
```


José L. Poza Lujan
Sergio Sáez Barona

Estructura de un nodo (cont.)

Determinar el evento (dependiente del estado)

```
void loop() {
  evento= EV_NINGUNO;

  if (evento == EV_NINGUNO && estado == EST_SUBIENDO) {
    activo= leer_final_carrera_sup();
    if (activo)
      evento= EV_FIN_CARRERA_SUP;
  } /* endif */

  if (evento == EV_NINGUNO && estado == EST_BAJANDO) {
    activo= leer_final_carrera_inf();
    if (activo)
      evento= EV_FIN_CARRERA_INF;
  } /* endif */

  if (evento == EV_NINGUNO) {
    orden= leer_orden_remota();
    if (orden != NINGUNA)
      evento= extraer_evento_orden(orden);
  } /* endif */
  ...
} /* end loop */
```

Inconveniente: El manejo de eventos depende de los estados.

José L. Poza Lujan
Sergio Sáez Barona

Estructura de un nodo (cont.)

Determinar el evento (cola de eventos)

```
void loop() {
  n_eventos= 0;

  activo= leer_final_carrera_sup();
  if (activo)
    evento[n_eventos++]= EV_FIN_CARRERA_SUP;

  activo= leer_final_carrera_inf();
  if (activo)
    evento[n_eventos++]= EV_FIN_CARRERA_INF;

  orden= leer_orden_remota();
  if (orden != NINGUNA)
    evento[n_eventos++]= extraer_evento_orden(orden);
  } /* endif */
  ...
} /* end loop */
```

El tamaño máximo de la cola depende del máximo número de eventos concurrentes que se puedan dar.

José L. Poza Lujan
Sergio Sáez Barona

Estructura de un nodo (cont.)

Atención del evento (evento único)

```
void loop () {
  ...
  switch (estado) {
    case EST_PARADA:
      switch (evento) {
        case EV_SUBIR:
          motor_arriba();
          estado= EST_MOVIENDO_SUBIENDO;
          break;
        case EV_BAJAR:
          ...
        default:
      } /* end switch */
      break;
    case EST_MOVIENDO_SUBIENDO:
      switch (evento) {
        case EV_PARAR:
          parar_motor();
          estado= EST_PARADA;
          break;
        case EV_FIN_CARRERA_SUP:
          ...
        default:
      } /* end switch */
      break;
    ...
  } /* end switch */
  ...
} /* end loop */
```


José L. Poza Lujan
Sergio Sáez Barona

Estructura de un nodo (cont.)

Atención del evento (cola de eventos)

```
void loop () {
  ...
  for (i= 0; i<n_eventos; i++) {
    switch (estado) {
      case EST_PARADA:
        switch (evento[i]) {
          case EV_SUBIR:
            motor_arriba();
            estado= EST_MOVIENDO_SUBIENDO;
            break;
            ...
          default:
        } /* end switch */
        break;
      case EST_MOVIENDO_SUBIENDO:
        switch (evento[i]) {
          case EV_PARAR:
            parar_motor();
            estado= EST_PARADA;
            break;
            ...
          default:
        } /* end switch */
        break;
      ...
    } /* end switch */
  } /* endfor */
  ...
} /* end loop */
```


José L. Poza Lujan
Sergio Sáez Barona

Índice

José L. Poza Lujan
Sergio Sáez Barona

Conclusiones

- Un sistema de control distribuido puede estar constituido por muchos nodos.
- Cada nodo se puede considerar como un sistema reactivo independiente.
- Se pueden modelar mediante máquinas de estados UML con algunas restricciones.
- Trasladar el modelo a código es bastante mecánico.
  $\Rightarrow$ Resulta en un código claro y escalable.
  - Añadir nuevos mecanismos de control no afecta al comportamiento, sólo al código de determinar los eventos.
  - Cambiar el comportamiento del nodo no afecta al código de gestión eventos, sólo a las acciones a realizar para cda evento.

José L. Poza Lujan
Sergio Sáez Barona

Arduino

Unidad 4. Sistemas Reactivos

José Luis Poza Lujan Sergio Sáez Barona

Tecnologías Interactivas y Fabricación Digital
Universidad Politécnica de Valencia

Noviembre 2015