

Tutorial NesC

Conceptos Básicos

¿Que es NesC?

- Lenguaje de programación orientado a componentes
- El usuario crea un componente ayudándose de otros ya creados
 - Sintaxis C
 - Semántica
 - Implementación de Interfaces (POO)
 - Modelo de Eventos (POE)

Componentes

- Utilizan interfaces de componentes ya existentes
- Proporcionan interfaces para poder ser utilizadas por otros componentes

Estructura de un Componente

- Físicamente
 - 2 ficheros: (por convenio)
 - Configuración e Implementación (miaplicacion.nc)
 - Módulos (miaplicacionM.nc)
 - Pueden ser más si incluimos librerías (.h)
- Lógicamente
 - 3 partes
 - Configuración
 - Implementación
 - Módulos

Configuración

- Por lo general para nosotros no va a contener nada
- Pero todas las parten han de estar, aunque estas estén vacías
- Su función, es la de la configuración del componente (utilizado generalmente para crear librerías)

Implementación

- Comúnmente se denomina WIRING
- No se corresponde a lo que nosotros pensamos por implementación
- Si la implementación de mi aplicación utiliza interfaces, estas han de ser proporcionada por otro componente
- Se decide que la interfaz que usa una aplicación es la que proporciona un componente

Implementaci3n (2)

- Ejemplo
 - Implementation{
Components Main, MiAplicaci3n;
Main.StdControl -> MiAplicaci3n.StdControl;
}

M3dulos

- Se corresponde a lo que pensamos nosotros por implementaci3n.
- Consiste en escribir en C lo que queremos que haga nuestra aplicaci3n
- Estructura
 - Module PracticaM{
 - Provides
 - Uses
 - Implementation

Provides

- En este punto se especifican las interfaces que va a proporcionar nuestro componente.
- Si nosotros proporcionamos una interfaz tenemos que **tener implementadas** las funciones que dice dicha interfaz

Uses

- Los las interfaces que utiliza nuestro componente.
- No se sabe quien es el que proporciona dicha interfaz (establecido en el WIRING)
- Si usamos una interfaz
 - podemos llamar a sus métodos
 - TENEMOS que implementar los eventos que se van a producir por el hecho de utilizar la interfaz
 - Habrá que realizar el WIRING en el fichero correspondiente para indicar quien proporciona la interfaz

Implementation

- En esta sección es donde realmente se programa el comportamiento de nuestra aplicación
- Como mínimo
 - Variables globales
 - Funciones de las interfaces que proporciono
 - Eventos de las interfaces que utilizo

Resumen de la estructura

MiAplicacion.nc

```
includes paquete;  
  
configuration MiAplicacion{ }  
  
implementation {  
  components  
    Main,  
    MiAplicacionM;  
  
  Main.StdControl ->  
  MiAplicacionM.StdControl;  
}
```

MiAplicacionM.nc

```
Includes paquete;  
Module MiAplicacionM{  
  provides{  
    interface StdControl;  
  }  
  uses {  
    interface SendMsg;  
  }  
}  
  
Implementation {  
  // Código C  
}
```

paquete.h

```
enum {  
  PKT_REQ = 1,  
  PKT_HELLO = 2  
};  
  
typedef struct paq{  
  int address;  
  int pkt_type;  
  .....  
} paquete;
```

Componente VS Aplicación

- Un componente puede proporcionar interfaces para poder ser utilizadas por otros que hagan de aplicación
- Un componente puede ser una aplicación y para ello ha de proporcionar una interfaz especial StdControl.

Tipos de datos

- Tipos de C
- uint16_t , es un entero sin signo de 16 bit
- uint8_t , lo mismo de 8 bit
- bool , es un boolean (TRUE , FALSE)
- result_t , es un boolean pero (SUCCES , FAIL)

Proporcionando la Interfaz StdControl

- Obliga a tener las siguiente funciones
 - Void Init()
 - Se ejecutara al arrancar la mota
 - Inicialización de variables globales, etc ...
 - Llamada a los metodos init() de los componentes que utilizo (no todos sino los que sean necesarios)
 - Void start()
 - Se ejecutara despues del init y cuando la mota pase de off a on (pensar en ahorro de energia)
 - Arranque de temporizadores
 - Llamada a los metodos start() de los componentes que utilizo (no todos sino los que sean necesarios)
 - Void stop()
 - Se ejecutara cuando se apague la mota o se suspenda (estado idle)
 - Llamar a los metodos stop de los componentes que utilizo (no todos sino los que sean necesarios)

Componentes que Proporciona TinyOs

- Tipos de componentes
 - Primitivos : los proporciona TinyOs
 - Compuestos: los proporciona una librería o una aplicación

Componentes para InterNetworking

- Los componentes primitivos para redes
- Se basan en un paquete que tiene el formato:
 - TOSMsg
 - Dirección de destino
 - (No lleva la de origen)
 - Datos
 - CRC
 - Longitud
 - Etc ...
 - TOSMsgPrt (puntero a un TOSMsg)

Componente GenericComm y GenericCommPromiscuous

- Son componentes para enviar y recibir paquetes por radio o UART
- Proporcionan la interfaz SendMsg y RecibeMsg
- Se comportan como un switch (si recibo un paquete con dirección UART lo envío por el serie y si recibo otra dirección lo envío por radio)

Interfaz SendMsg

- Si uso la interfaz SendMsg de un componente GenericComm puedo usar la función
 - Send(addr , long_datos , TOSMsg)
 - Envía el paquete TOSMsg (ha de ser una variable global) a la dirección indicada
 - YO solo relleno del paquete TOSMsg el campo data (con una estructura de mi tipo de mensaje personal) -> Nivel superior
 - Las direcciones especiales
 - TOS_BCAST_ADDR – Dir. de broadcast de red
 - TOS_LOCAL_ADDRESS – La dirección de Mi mota (localhost)
 - TOS_UART_ADDR– La dirección del puerto COM
- Tengo que implementar el evento SendDone()

Interfaz ReceiveMsg

- Si utilizo la interfaz ReceiveMsg de un componente GenericComm no puedo usar ninguna función pero si que TENGO que implementar el evento de recibir un mensaje
 - Recibir un mensaje en el caso en que la dirección destino del mensaje coincida con mi TOS_LOCAL_ADDRESS o sea TOS_BCAST_ADDR
 - En el caso de ser el Componente GenericCommPromiscuous Recibiré siempre todos los mensajes, vaya a quien vayan (muy interesante)

Interfaces Parametrizadas

- Se utiliza para poder tener diferentes instancias de una interfaz
- Por ejemplo, las interfaces SendMsg y ReceiveMsg del componente GenericComm son parametrizadas
- Yo le dije que para este tipo de paquete use esta implementación del evento recibe y para el otro tipo, esta otra.

Componente TimerC

- Ofrece funciones de temporización mediante la interfaz Timer parametrizada (para poder tener muchos timer) -> de momento 10 como maximo
- Si utilizo la interfaz Timer de este componente tengo que implementar el evento fired() que se llamara cada x tiempo
- Para arrancar un timer llamo a start()
 - Start(tipo, tiempo(ms))
 - Tipo : TIME_REPEAT o TIMER_ONE_SHOT
- Para pararlo llamo a stop()

Componente ADCC

- Se utiliza para realizar una conversión Analógico – Digital, en definitiva, vale para obtener los valores de los sensores que posee la mota.
 - Temperatura
 - Humedad
 - Etc ...
- Proporciona las interfaces: ADC y ADCCControl

Interfaz ADC

- Si utilizo la interfaz ADC puedo llamar a las siguientes funciones
 - `getData();`
 - Devuelve el dato del sensor mediante un unico muestreo
 - `GetContinuousData()`
 - Devuelve el dato del sensor mediante una sucesión continua de muestreos.
- Obliga a implementar el evento
 - `dataReady(uint16_t data9)`
 - Es invocado cuando el datos esta preparado para ser leído

Estrucuta de Directorios de TinyOS

- /apps – Estan los componentes que son aplicaciones
- /tos/interfaces – Estan las interfaces que ofrecen los componentes (seguro que por lo menos estan TODAS las que ofrecen los componentes primitivos)
- /tos/system – Estan los componentes primitivos del TinyOs
- /tos/types – Estan los ficheros que contiene los tipos de datos (Pej: AM.h contiene la estructura de un mensaje TOSMsg)

¿Donde Hago mi aplicación?

- En el directorio /apps/
 - Asi puedo utilizar los makes generales para las aplicaciones
- Tendre un makefile especifico de mi aplicación
 - Estructura:
PLATFORMS=telos pc
COMPONENT=MiAplicacion // IMPORTANTE
PFLAGS= // ruta de las librerias que queremos usar
include ../Makerules

Tutorial TOSSIM

Simulación

Como lo ejecuto

- Compilo para PC : make pc
- /build/pc tendre el ejecutable
- Invocacion: ./ejecutable [parametros] nºde motas [&]
 - -nodbgout – No debug por la stdout
 - -gui – Espera a TinyViz
 - -b – Arranca las motas en n segundos

Como funciona

- Ejecuta el numero de motas que le he indicado
- Proporciona una interfaz de simulación
- Proporciona dos puertos COM virtuales
 - Tossim-uart
 - Todo el trafico que se mande al UART_ADDR sera enviado a este puerto com virtual
 - Tossim-radio
 - Todo el trafico que se mande por el aire (a cualquier dirección) será recibido por este puero com virtual
- Proporciona un método de debugging
 - Dbg(nivel," ",...) al estilo printf
 - NIVEL : DBG_USR1 , DBG_USR2, etc ...
- Esto permite utilizar utilidades externas

Utilidades

- TinyViz (/net/tinyos/sim/tinyviz)
- Listener (/net/tinyos/tools/Listen.class)
- SerialForwarder (/net/tinyos/sf/SerialForwarder.class)
- Se encuentran en /tools/java/
- Se arrancan desde este PATH

TinyViz

- Utilidad de interfaz grafica de simulación
- Se utiliza en conjunción con la opción `-gui` del ejecutable
- Permite un debugging muy sencillo mediante MATCH de mensajes de debug.
- `Net.tinyos.sim.TiniViz`
- Habilita de forma automatica el SerialForwarder

SerialForwarder

- Asocia un puerto de comunicaciones a un puerto TCP concreto
- Es una pasarela entre puertos, si recibo de uno lo envio por el otro y viceversa
- Se puede ver como un gateway entre comm y TCP
- P. Ej: Si recibo por `tossim-radio` envio por TCP 8080
- `Net.tosim.sf.SerialForwarder` puerto

Listener

- Muestra los mensajes en modo "raw" de un puerto específico
- Permite ver que es lo que se esta recibiendo

Cosas a Tener en Cuenta

- Envio de Paquete con cola
- ADC necesita un noarc
- Arquitectura (Base+Normales)
- Aplicación java