NesC y TinyOS
"The nesC Language: A Holistic Approach to **N**etworked **E**mbedded **S**ystems"

Tomado de paper de:

D. Gay, P. Levis, R. Behren, M. Welsh, E. Brewer, D. Culler

In Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation (PLDI)

Introducción

- Avances en redes inalámbricas e integración de sensores permiten desarrollar nodos pequeños, flexibles y de bajo costo que interactúan con su ambiente a través de sensores, actuadores e interfaces de comunicaciones.
- Motes: nombre coloquial dado a sistemas que integran CPU, memoria, comunicación de radio frecuencia u óptica, sensores basados en MEMs (Micro-Electro-Mechanical system) de bajo consumo, etc.

¿Qué es nesC?

- Un lenguaje de programación de sistemas para **N**etworked **E**mbedded **S**ystem, como los motes.
- Es una extensión de C
  - C tiene control directo del hardware
  - Muchos programadores ya conocen C
  - nesC provee chequeos de seguridad ausentes en C
- Permite análisis completo del programa durante compilación
  - Detecta carreras críticas => Elimina errores potenciales
  - Agresivo en inlining de funciones => Optimización
- Lenguaje estático (para permitir lo anterior)
  - No hay manejo dinámico de memoria
  - No hay punteros a funciones
  - Así el grafo de llamadas y acceso a variables es completamente conocidos para el compilador
- Soporta y refleja el diseño de TinyOS
  - Basado en concepto de componentes
  - Soporta directamente el modelo de concurrencia conducido por eventos
  - Considera el problema de acceso a datos compartidos vía definición de secciones atómicas y palabra reservada norace.

Desafíos abordados por NesC

- Conducido por eventos: reacciona ante interacciones con el ambiente (No procesamiento batch o interactivo). Las aplicaciones gráficas son conducidas por eventos: cuando se presiona una opción (sería el evento), el programa reacciona a ella.
- Recursos limitados: poca memoria, bajo costo, bajo consumo.
- Confiabilidad: No hay mecanismo de recuperación de fallas fuera de reboot automático
- Requerimiento de tiempo real soft: Ej: consultar sensor, atender radio.

TinyOS

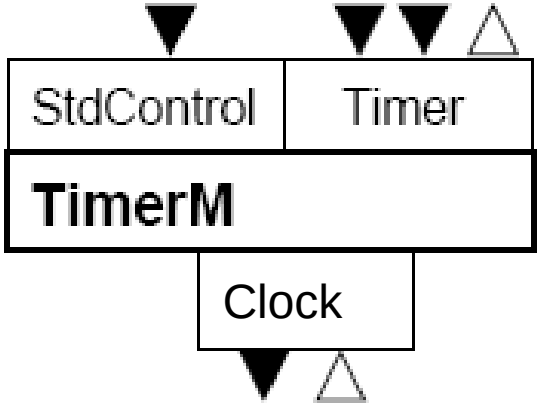
- Arquitectura basada en componentes
  - Una aplicación cablea componentes reusables según la aplicación
- Concurrencia basada en Tareas y eventos
  - Tareas & eventos corren hasta terminar, pero un evento puede interrumpir (preempt) la ejecución de una tarea u otro evento
  - Tareas no pueden interrumpir otra tarea o evento
- Operaciones Split-phase (fase partida): para operaciones no bloqueante
  - El inicio de un comando retorna inmediatamente y un evento avisa el término

Propiedades Claves de TinyOS

- Todos los recursos son conocidos de manera estática (es decir a tiempo de compilación)
- No es un SO de propósito general: las aplicaciones son construidas a partir de un conjunto de componentes reusables sumadas a código específico de la aplicación
- límite HW/SW puede variar dependiendo de la aplicación y plataforma de HW => descomposición flexible es requerida

Componentes

- Una componente puede ser vista como una componente electrónica discreta (un timer, un contador, etc).
- Una componente provee (“pines de la componente) y usa interfaces
  - Las interfaces son el único punto de acceso a una componente
- Una interfaz modela algún servicio a través de comandos (por ejemplo, envío de un mensaje) o eventos (por ejemplo, envío concluido).
- El proveedor de un comando implementa los comandos, mientras el usuario (llamador) implementa los eventos.




```
module TimerM {
  provides {
    interface StdControl;
    interface Timer;
  }
  uses interface Clock as Clk;
} ...
```

Interfaz

- Interfaces bidireccionales soportan ejecución de fase partida o dividida (split-phase)

```
interface Clock {
  command result_t setRate(char interval, char scale);
  event result_t fire();
}

interface Send {
  command result_t send(TOS_Msg *msg, uint16_t length);
  event result_t sendDone(TOS_Msg *msg, result_t success);
}

interface ADC {
  command result_t getData();
  event result_t dataReady(uint16_t data);
}
```

Implementación de Componentes

- Hay dos tipos de componentes: módulos & configuraciones
- Módulos proveen código de aplicación e implementan una o más interfaces.
- Configuraciones cablean componentes
  - Conectan interfaces usadas por componentes a interfaces provistas por otros
  - Notar definición recursiva, similar a la estructura de datos árbol. Los módulos son como las hojas del árbol, las componentes son los nodos interiores del árbol.

Módulos: Ejemplo, aplicación Surge

- Surge: Cada segundo obtiene la lectura de un sensor y envía un mensaje con ese valor

```
module SurgeM {
  provides interface StdControl;
  uses interface ADC;
  uses interface Timer;
  uses interface Send;
}
implementation {
  uint16_t sensorReading;

  command result_t StdControl.init() {
    return call Timer.start(TIMER_REPEAT, 1000);
  }

  event result_t Timer.fired() {
    call ADC.getData();
    return SUCCESS;
  }

  event result_t ADC.dataReady(uint16_t data) {
    sensorReading = data;
    ... send message with data in it ...
    return SUCCESS;
  }
  ...
}
```

Se debe implementar cada comando ofrecido

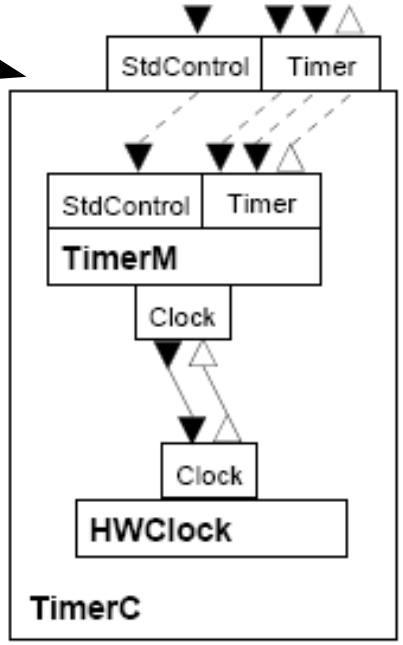
Y cada evento recibido

Configuraciones

- Cablea las componentes, en este caso, TimerM y HWClock

Mapeo con =

Cableado con ->

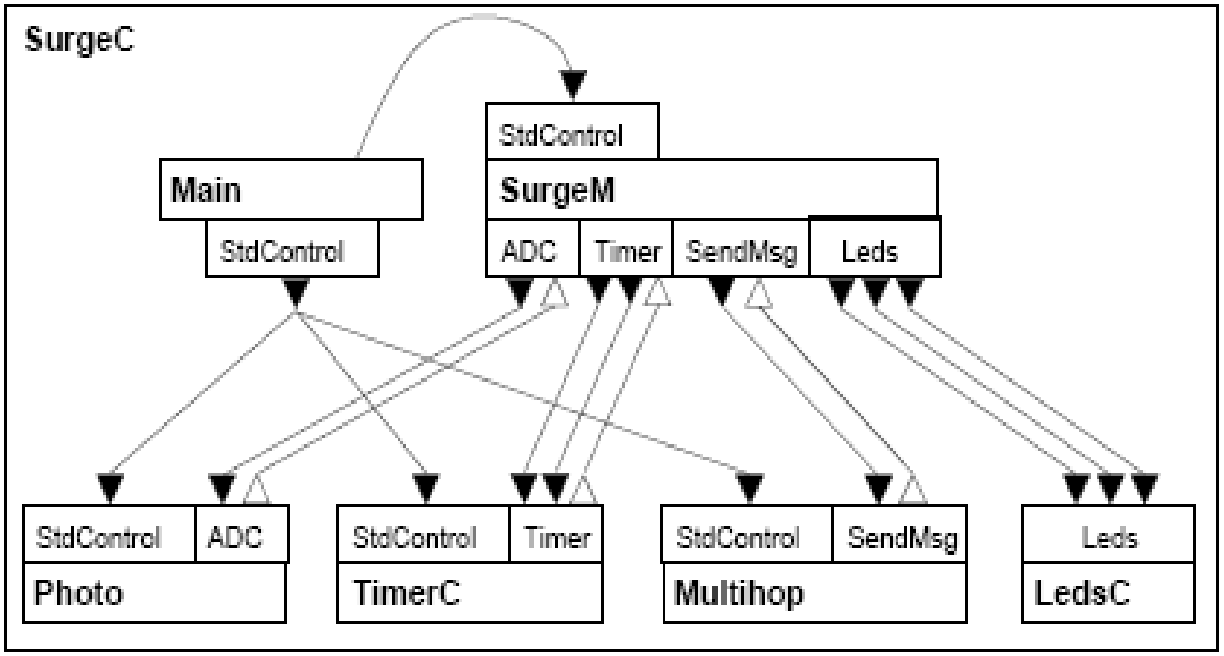



```
configuration TimerC {
  provides {
    interface StdControl;
    interface Timer;
  }
}
implementation {
  components TimerM, HWClock;

  StdControl = TimerM.StdControl;
  Timer = TimerM.Timer;

  TimerM.Clk -> HWClock.Clock;
}
```

Ejemplo: Configuración SurgeC



Concurrencia y atomicidad

- Código asincrónico (AC): es el alcanzable por al menos un manejador de interrupción
- Código sincrónico (SC): es alcanzable sólo por tareas
  - Consecuencia de esta definición: Código sincrónico es atómico con respecto a otro código sincrónico
- Aún hay una carrera potencial entre AC y SC
  - Cualquier actualización a estado compartido hecha desde un AC
  - Cualquier actualización a estado compartido hecha desde SC que es también actualizado desde AC
- Solución: Invariante libre de carrera
  - Cualquier actualización a variable compartida es hecha por SC u ocurre dentro de una sección atómica

atomic y norace

```
module SurgeM { ... }
implementation {
  bool busy;
  norace uint16_t sensorReading;

  event result_t Timer.fired() {
    bool localBusy;
    atomic {
      localBusy = busy;
      busy = TRUE;
    }
    if (!localBusy)
      call ADC.getData();
    return SUCCESS;
  }

  task void sendData() { // send sensorReading
    adcPacket.data = sensorReading;
    call Send.send(&adcPacket, sizeof adcPacket.data);
    return SUCCESS;
  }

  event result_t ADC.dataReady(uint16_t data) {
    sensorReading = data;
    post sendData();
    return SUCCESS;
  }
  ...
}
```

Deshabilita interrupciones

Habilita interrupciones

- Atomic
  - Deshabilita interrupciones
  - Dentro de una sección atómica no podemos llamar a comandos o señalar eventos.
  - Si una variable x es accedida por un AC, cualquier acceso a x fuera de una sección atómica es un error de compilación
  - Una sección atómica debería ser corta (las interrupciones están deshabilitadas!).

norace

- Si un programador sabe que una variable generadora de posible carrera crítica no lo será, la declara norace

Evaluación

- El paper presenta la prueba de tres aplicaciones TinyOS:
  - Surge, Maté, TinyDB
- Núcleo (Core) de TinyOS consiste de 172 componentes
  - 108 módulos y 64 configuraciones
- Agrupa componentes necesarias, vía interfaces nesC bidireccionales, para una aplicación específica.

Application	Modules	OS Modules (% of full OS)	Lines	OS Lines (% of full OS)
Surge	31	27 (25%)	2860	2160 (14%)
Maté	35	28 (25%)	4736	2524 (17%)
TinyDB	65	38 (35%)	11681	4160 (28%)

Efecto de inlining

- Impacto de inlining en código y desempeño

App	Code size		Code reduction	Data size	CPU reduction
	inlined	*noninlined*			
Surge	14794	16984	12%	1188	15%
Maté	25040	27458	9%	1710	34%
TinyDB	64910	71724	10%	2894	30%

Problemas Pendientes *(en esa época)*

- Localización estática de memoria
  - Ventajas
    - Permite análisis fuera de línea de los requerimientos de memoria
  - Problema:
    - No hay localización dinámica
- Obliga código corto en secciones atómicas y manejadores de comando y eventos
- Poco soporte del lenguaje para programación de tiempo real

Contiki OS alternativa a TinyOS

- Si bien esta presentación cubre TinyOS, Contiki OS es otro actor relevante.
- Se trata de un OS multi-hebras, a diferencia de TinyOS que es conducido por eventos.
- Contiki está escrito en C.
- Contiki implementa un stack TCP/IP usando pocos recursos.
- Dio origen a Contiki-NG (new generation) pensado para Internet of Things.
- Herramientas de desarrollo incluyen Cooja, un simulador de redes de nodos Contiki y uns subsistema para desarrollo de interfaces gráficas.