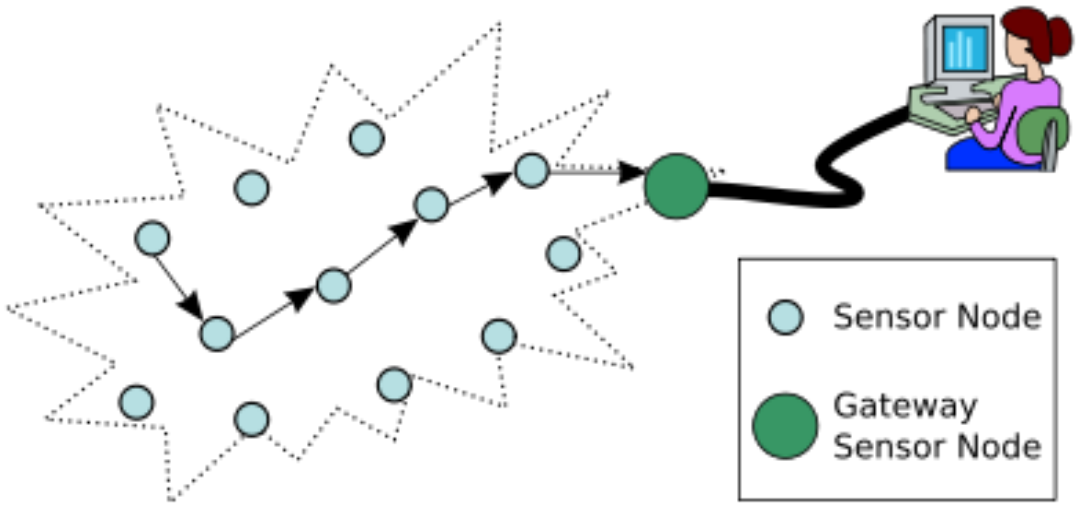
TinyOS & NesC

Introduzione alla programmazione delle WSN

Letterio Galletta

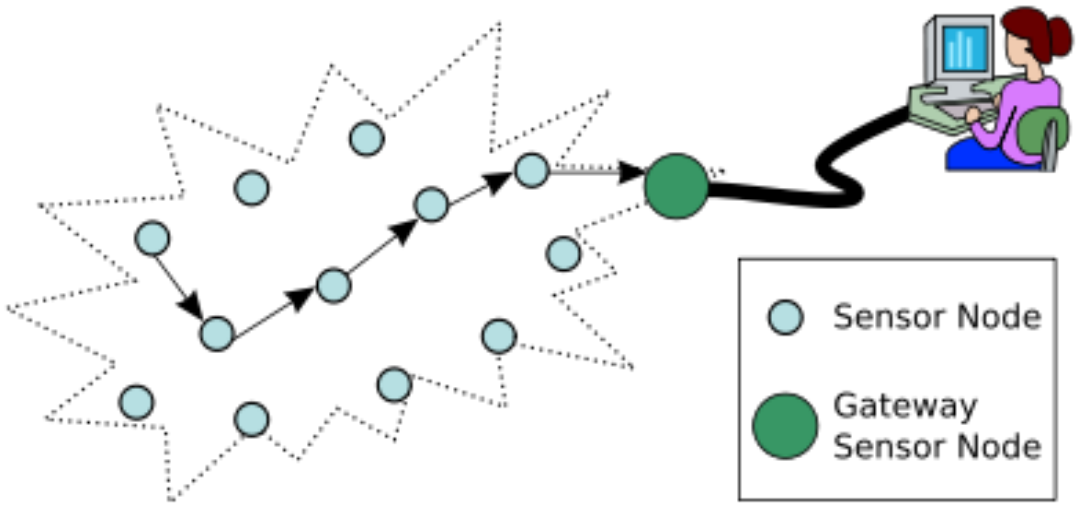
10 Maggio 2011

WSN



- Abbiamo visto: protocolli MAC, protocolli di routing, sicurezza, standard industriali (802.15.4 & Zigbee)

WSN



- Abbiamo visto: protocolli MAC, protocolli di routing, sicurezza, standard industriali (802.15.4 & Zigbee)
- L'obiettivo di questa lezione: introdurre gli strumenti per lo sviluppo di applicazioni

Hardware dei sensori

- Alimentazione a batteria
- Limitata capacità di memoria
  - es. 4KB (Mica, Mica2, Mica2dot, Micaz), 8KB (Iris)
- Limitata capacità di calcolo
  - es. 4Mhz (Mica), 8Mhz (Mica2, Mica2dot, Micaz,Iris)
- Interfaccia radio
- Trasduttori

Hardware dei sensori

- Alimentazione a batteria
- Limitata capacità di memoria
  - es. 4KB (Mica, Mica2, Mica2dot, Micaz), 8KB (Iris)
- Limitata capacità di calcolo
  - es. 4Mhz (Mica), 8Mhz (Mica2, Mica2dot, Micaz,Iris)
- Interfaccia radio
- Trasduttori

Abbiamo bisogno di strumenti adeguati!

Sistemi operativi per i sensori

Requisiti di un sistema operativo per i sensori:

- Dimensioni ridotte

Sistemi operativi per i sensori

Requisiti di un sistema operativo per i sensori:

- Dimensioni ridotte
- Basso carico computazionale

Sistemi operativi per i sensori

Requisiti di un sistema operativo per i sensori:

- Dimensioni ridotte
- Basso carico computazionale
- Gestione efficiente delle risorse

Sistemi operativi per i sensori

Requisiti di un sistema operativo per i sensori:

- Dimensioni ridotte
- Basso carico computazionale
- Gestione efficiente delle risorse
  - Ottimizzare il consumo energetico

Sistemi operativi per i sensori

Requisiti di un sistema operativo per i sensori:

- Dimensioni ridotte
- Basso carico computazionale
- Gestione efficiente delle risorse
  - Ottimizzare il consumo energetico
- Flessibilità e modularità

Sistemi operativi per i sensori

Requisiti di un sistema operativo per i sensori:

- Dimensioni ridotte
- Basso carico computazionale
- Gestione efficiente delle risorse
  - Ottimizzare il consumo energetico
- Flessibilità e modularità
  - Adeguarsi alle diverse piattaforme hardware e ai diversi scenari applicativi

Diversi progetti esistenti

Contiki

SwissQM

LiteOS

TinyOS

TinyOS

Caratteristiche

TinyOS

Caratteristiche

- Dimensioni fortemente ridotte

TinyOS

Caratteristiche

- Dimensioni fortemente ridotte
- Non proprio un sistema operativo, ma un ambiente di esecuzione implementato seguendo un modello ad eventi

TinyOS

Caratteristiche

- Dimensioni fortemente ridotte
- Non proprio un sistema operativo, ma un ambiente di esecuzione implementato seguendo un modello ad eventi
- Architettura component-based (ereditata dal NesC)

TinyOS

Caratteristiche

- Dimensioni fortemente ridotte
- Non proprio un sistema operativo, ma un ambiente di esecuzione implementato seguendo un modello ad eventi
- Architettura component-based (ereditata dal NesC)
  - Fornisce una ricca libreria di componenti predefiniti: protocolli di rete, servizi distribuiti, astrazioni hardware (timer, trasduttori)

TinyOS

Caratteristiche

- Dimensioni fortemente ridotte
- Non proprio un sistema operativo, ma un ambiente di esecuzione implementato seguendo un modello ad eventi
- Architettura component-based (ereditata dal NesC)
  - Fornisce una ricca libreria di componenti predefiniti: protocolli di rete, servizi distribuiti, astrazioni hardware (timer, trasduttori)
- Una tipica applicazione è implementata assemblando tra loro componenti

Linguaggio Network Embedded System C (NesC)

Variante del C per la programmazione dei motes

Restrizioni

Linguaggio Network Embedded System C (NesC)

Variante del C per la programmazione dei motes

Restrizioni

- Gestione della memoria statica (nessun heap)

Linguaggio Network Embedded System C (NesC)

Variante del C per la programmazione dei motes

Restrizioni

- Gestione della memoria statica (nessun heap)
- Restrizione sull'uso dei puntatori (no puntatori a funzioni)

Linguaggio Network Embedded System C (NesC)

Variante del C per la programmazione dei motes

Restrizioni

- Gestione della memoria statica (nessun heap)
- Restrizione sull'uso dei puntatori (no puntatori a funzioni)
- No ricorsione

Linguaggio Network Embedded System C (NesC)

Variante del C per la programmazione dei motes

Restrizioni

- Gestione della memoria statica (nessun heap)
- Restrizione sull'uso dei puntatori (no puntatori a funzioni)
- No ricorsione

Estensioni

Linguaggio Network Embedded System C (NesC)

Variante del C per la programmazione dei motes

Restrizioni

- Gestione della memoria statica (nessun heap)
- Restrizione sull'uso dei puntatori (no puntatori a funzioni)
- No ricorsione

Estensioni

- Strutturazione dell'applicazione a componenti

Linguaggio Network Embedded System C (NesC)

Variante del C per la programmazione dei motes

Restrizioni

- Gestione della memoria statica (nessun heap)
- Restrizione sull'uso dei puntatori (no puntatori a funzioni)
- No ricorsione

Estensioni

- Strutturazione dell'applicazione a componenti
- Modello di programmazione a eventi

Linguaggio Network Embedded System C (NesC)

Variante del C per la programmazione dei motes

Restrizioni

- Gestione della memoria statica (nessun heap)
- Restrizione sull'uso dei puntatori (no puntatori a funzioni)
- No ricorsione

Estensioni

- Strutturazione dell'applicazione a componenti
- Modello di programmazione a eventi
- Modello di concorrenza basato su task ed eventi

TinyOS & NesC

Alcune note sull'installazione

- L'ambiente di sviluppo per TinyOS è disponibile per Linux e Windows (Cygwin)
- Mac OS X non è ufficialmente supportato sebbene esistano dei port
- Tutto il necessario è disponibile sul sito sotto forma di pacchetti rpm
- Due requisiti
  1. Java 6 SDK
  2. Graphviz
- Configurare le variabili di ambiente (TOSROOT, TOSDIR, CLASSPATH, MAKERULES)
  - Basta aggiungere "`source /opt/tinyos-2.1.1/tinyos.sh`" in `.bashrc` o `.profile`
- Per verificare se le impostazioni sono corrette `tos-check-env`

TinyOS & NesC

Alcune note sull'installazione

- L'ambiente di sviluppo per TinyOS è disponibile per Linux e Windows (Cygwin)
- Mac OS X non è ufficialmente supportato sebbene esistano dei port
- Tutto il necessario è disponibile sul sito sotto forma di pacchetti rpm
- Due requisiti
  1. Java 6 SDK
  2. Graphviz
- Configurare le variabili di ambiente (TOSROOT, TOSDIR, CLASSPATH, MAKERULES)
  - Basta aggiungere "`source /opt/tinyos-2.1.1/tinyos.sh`" in `.bashrc` o `.profile`
- Per verificare se le impostazioni sono corrette `tos-check-env`

Installazione su Debian

Aggiungere alle proprie sorgenti

```
http://tinyos.stanford.edu/tinyos/dists/ubuntu lucid
                                    main
```

Componenti

- Unità software dotata di un proprio stato e che svolge un determinato compito

Componenti

- Unità software dotata di un proprio stato e che svolge un determinato compito
- I programmi NesC sono costituiti da un insieme di componenti

Componenti

- Unità software dotata di un proprio stato e che svolge un determinato compito
- I programmi NesC sono costituiti da un insieme di componenti
  - Aumenta la modularità e la flessibilità dell'applicazione

Componenti

- Unità software dotata di un proprio stato e che svolge un determinato compito
- I programmi NesC sono costituiti da un insieme di componenti
  - Aumenta la modularità e la flessibilità dell'applicazione
  - La composizione è fatta staticamente dal programmatore

Componenti

- Unità software dotata di un proprio stato e che svolge un determinato compito
- I programmi NesC sono costituiti da un insieme di componenti
  - Aumenta la modularità e la flessibilità dell'applicazione
  - La composizione è fatta staticamente dal programmatore
- I componenti possono essere divisi in 3 categorie

Componenti

- Unità software dotata di un proprio stato e che svolge un determinato compito
- I programmi NesC sono costituiti da un insieme di componenti
  - Aumenta la modularità e la flessibilità dell'applicazione
  - La composizione è fatta staticamente dal programmatore
- I componenti possono essere divisi in 3 categorie
  1. Hardware abstraction (es. led, timer)

Componenti

- Unità software dotata di un proprio stato e che svolge un determinato compito
- I programmi NesC sono costituiti da un insieme di componenti
  - Aumenta la modularità e la flessibilità dell'applicazione
  - La composizione è fatta staticamente dal programmatore
- I componenti possono essere divisi in 3 categorie
  1. Hardware abstraction (es. led, timer)
  2. Synthetic hardware

Componenti

- Unità software dotata di un proprio stato e che svolge un determinato compito
- I programmi NesC sono costituiti da un insieme di componenti
  - Aumenta la modularità e la flessibilità dell'applicazione
  - La composizione è fatta staticamente dal programmatore
- I componenti possono essere divisi in 3 categorie
  1. Hardware abstraction (es. led, timer)
  2. Synthetic hardware
  3. High-level compoment (es. protocolli di comunicazione)

Componenti

Interfacce

- Le interfacce sono l'unico "punto di accesso" per interagire con il componente

Componenti

Interfacce

- Le interfacce sono l'unico "punto di accesso" per interagire con il componente
- Un'interfaccia definisce un insieme di comandi e un insieme di gestori di eventi

Componenti

Interfacce

- Le interfacce sono l'unico "punto di accesso" per interagire con il componente
- Un'interfaccia definisce un insieme di comandi e un insieme di gestori di eventi
  - I comandi descrivono il servizio offerto

Componenti

Interfacce

- Le interfacce sono l'unico "punto di accesso" per interagire con il componente
- Un'interfaccia definisce un insieme di comandi e un insieme di gestori di eventi
  - I comandi descrivono il servizio offerto
  - I gestori degli eventi sono invocati per notificare a chi usa l'interfaccia il verificarsi di un evento

Componenti

Interfacce

- Le interfacce sono l'unico "punto di accesso" per interagire con il componente
- Un'interfaccia definisce un insieme di comandi e un insieme di gestori di eventi
  - I comandi descrivono il servizio offerto
  - I gestori degli eventi sono invocati per notificare a chi usa l'interfaccia il verificarsi di un evento
- Ogni componente definisce un insieme di interfacce esportate ed un insieme di interfacce utilizzate

Componenti

Interfacce

- Le interfacce sono l'unico "punto di accesso" per interagire con il componente
- Un'interfaccia definisce un insieme di comandi e un insieme di gestori di eventi
  - I comandi descrivono il servizio offerto
  - I gestori degli eventi sono invocati per notificare a chi usa l'interfaccia il verificarsi di un evento
- Ogni componente definisce un insieme di interfacce esportate ed un insieme di interfacce utilizzate
  - Invoca i comandi e implementa i gestori degli eventi di ogni interfaccia utilizzata

Componenti

Interfacce

- Le interfacce sono l'unico "punto di accesso" per interagire con il componente
- Un'interfaccia definisce un insieme di comandi e un insieme di gestori di eventi
  - I comandi descrivono il servizio offerto
  - I gestori degli eventi sono invocati per notificare a chi usa l'interfaccia il verificarsi di un evento
- Ogni componente definisce un insieme di interfacce esportate ed un insieme di interfacce utilizzate
  - Invoca i comandi e implementa i gestori degli eventi di ogni interfaccia utilizzata
  - Implementa i comandi e invoca i gestori degli eventi di ogni interfaccia esportata

Moduli e configurazioni

I componenti sono di due tipi

Moduli
Implementano i comandi delle interfacce che esportano e i gestori degli eventi delle interfacce che usano

Moduli e configurazioni

I componenti sono di due tipi

Moduli

Implementano i comandi delle interfacce che esportano e i gestori degli eventi delle interfacce che usano

Configurazioni

Moduli e configurazioni

I componenti sono di due tipi

Moduli

Implementano i comandi delle interfacce che esportano e i gestori degli eventi delle interfacce che usano

Configurazioni

- Definito assemblando componenti già esistenti

Moduli e configurazioni

I componenti sono di due tipi

Moduli

Implementano i comandi delle interfacce che esportano e i gestori degli eventi delle interfacce che usano

Configurazioni

- Definito assemblando componenti già esistenti
- In ogni applicazione c'è una configurazione globale che definisce le connessioni tra i componenti cui è composta

Esempio: Powerup

Hello World: il primo programma

```
#include <stdio.h>

int main(void)
{
  printf("Hello␣World!\n");
  return 0;
}
```

Powerup

- HelloWorld nel caso dei sensori
- Accende il primo led del sensore e si sospende

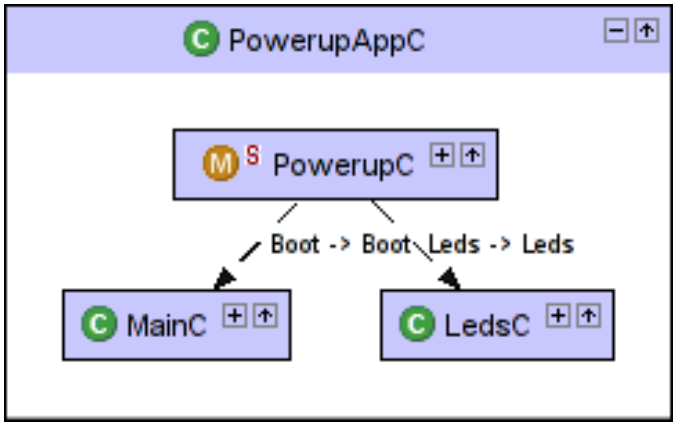
Powerup

Componenti

L'applicazione è formata da due componenti

PowerupC modulo che realizza la logica

PowerupAppC configurazione che collega `PowerupC` con i componenti forniti da TinyOS



PowerupC

```
module PowerupC
{
  uses interface Boot;
  uses interface Leds;
}
implementation
{
  event void Boot.booted()  {
    call Leds.led0On();
  }
}
```

Usa le interfacce `Boot` e `Leds`

- Implementa il gestore degli eventi `booted` richiesto da `Boot`
- Accende il led 0 invocando il comando `led0On` di `Leds`

Le interfacce Boot e Leds

Boot

```
interface Boot {
    event void booted();
}
```

Leds

```
interface Leds {
  command void led0On();

  command void led0Off();

  command void led0Toggle();

  ...
}
```

PowerupAppC

```
configuration PowerupAppC{}
implementation {
  components MainC, PowerupC, LedsC;

  MainC.Boot <- PowerupC;

  PowerupC -> LedsC.Leds;
}
```

Assembla l'applicazione usando i componenti predefiniti `MainC` e `LedsC`

- I componenti usanti durante l'assemblaggio sono dichiarati usando `components`
- L'assemblaggio dei componenti (wired) è fatto a livello di interfacce tramite gli operatori `<-` e `->`

LedsC

Esempio di componente predefinito

```
configuration LedsC {
  provides interface Leds;
}
implementation {
  components LedsP, PlatformLedsC;

  Leds = LedsP;

  LedsP.Init <- PlatformLedsC.Init;
  LedsP.Led0 -> PlatformLedsC.Led0;
  LedsP.Led1 -> PlatformLedsC.Led1;
  LedsP.Led2 -> PlatformLedsC.Led2;
}
```

Powerup

Compilazione

- TinyOS è provvisto di un sistema di build molto potente e flessibile basato su `Makefiles`

```
COMPONENT=PowerupAppC
include $(MAKERULES)
```

Powerup

Compilazione

- TinyOS è provvisto di un sistema di build molto potente e flessibile basato su `Makefiles`

```
COMPONENT=PowerupAppC
include $(MAKERULES)
```

- Per compilare basta usare `make <platform>` (es. `make micaz`)

Powerup

Compilazione

- TinyOS è provvisto di un sistema di build molto potente e flessibile basato su `Makefiles`

```
COMPONENT=PowerupAppC
include $(MAKERULES)
```

- Per compilare basta usare `make <platform>` (es. `make micaz`)
- Il risultato della compilazione è un file binario (`build/micaz/main.exe`)che include TinyOS e tutti i componenti dell'applicazione

Sensing

- Generalmente l’acquisizione di dati da un trasduttore richiede 2 operazioni:

Sensing

- Generalmente l'acquisizione di dati da un trasduttore richiede 2 operazioni:
  1. Inizializzazione del hardware

Sensing

- Generalmente l'acquisizione di dati da un trasduttore richiede 2 operazioni:
  1. Inizializzazione del hardware
  2. Acquisizione dei dati

Sensing

- Generalmente l'acquisizione di dati da un trasduttore richiede 2 operazioni:
  1. Inizializzazione del hardware
  2. Acquisizione dei dati
- In TinyOS un trasduttore è un componente che fornisce un'interfaccia per la lettura di dati (`Read`,`ReadStream`,`ReadNow`)

Sensing

- Generalmente l'acquisizione di dati da un trasduttore richiede 2 operazioni:
  1. Inizializzazione del hardware
  2. Acquisizione dei dati
- In TinyOS un trasduttore è un componente che fornisce un'interfaccia per la lettura di dati (`Read`,`ReadStream`,`ReadNow`)
- I dettagli implementativi del componente dipendono dalla specifica piattaforma e dalla sensorsboard utilizzata

Interfaccia `Read`

```
interface Read<val_t> {
    command error_t read();

    event void readDone( error_t result, val_t val );
}
```

- Interfaccia parametrica rispetto al tipo del dato letto
- L'operazione di `read` è implementata in modo asincrono (split-phase operation)

Split-phase operation

- Tecnica che realizza una forma comunicazione asincrona tra componenti

Split-phase operation

- Tecnica che realizza una forma comunicazione asincrona tra componenti
- Usata per eseguire operazioni bloccanti o di lunga durata

Split-phase operation

- Tecnica che realizza una forma comunicazione asincrona tra componenti
- Usata per eseguire operazioni bloccanti o di lunga durata

In cosa consiste?

Dividere l'operazione in due fasi

Split-phase operation

- Tecnica che realizza una forma comunicazione asincrona tra componenti
- Usata per eseguire operazioni bloccanti o di lunga durata

In cosa consiste?

Dividere l'operazione in due fasi

1. Invocazione (comando che termina subito)

Split-phase operation

- Tecnica che realizza una forma comunicazione asincrona tra componenti
- Usata per eseguire operazioni bloccanti o di lunga durata

In cosa consiste?

Dividere l'operazione in due fasi

1. Invocazione (comando che termina subito)
2. Restituzione del risultato (notifica di un evento)

Split-phase operation

- Tecnica che realizza una forma comunicazione asincrona tra componenti
- Usata per eseguire operazioni bloccanti o di lunga durata

In cosa consiste?

Dividere l'operazione in due fasi

1. Invocazione (comando che termina subito)
2. Restituzione del risultato (notifica di un evento)

Vantaggi

Split-phase operation

- Tecnica che realizza una forma comunicazione asincrona tra componenti
- Usata per eseguire operazioni bloccanti o di lunga durata

In cosa consiste?

Dividere l'operazione in due fasi

1. Invocazione (comando che termina subito)
2. Restituzione del risultato (notifica di un evento)

Vantaggi

- Non blocca il sistema

Split-phase operation

- Tecnica che realizza una forma comunicazione asincrona tra componenti
- Usata per eseguire operazioni bloccanti o di lunga durata

In cosa consiste?

Dividere l'operazione in due fasi

1. Invocazione (comando che termina subito)
2. Restituzione del risultato (notifica di un evento)

Vantaggi

- Non blocca il sistema
- Permette di ottimizzare l'uso della memoria

Esempio: `Sense`

Applicazione giocattolo che

- È formata da due componenti

Esempio: `Sense`

Applicazione giocattolo che

- È formata da due componenti
  1. Il modulo `SenseC`

Esempio: `Sense`

Applicazione giocattolo che

- È formata da due componenti
  1. Il modulo `SenseC`
  2. La configurazione `SenseAppC`

Esempio: `Sense`

Applicazione giocattolo che

- È formata da due componenti
  1. Il modulo `SenseC`
  2. La configurazione `SenseAppC`
- Periodicamente acquisisce un dato da `DemoSensorC` e ne visualizza i bit meno significativi attraverso i led

Esempio: `Sense`

Applicazione giocattolo che

- È formata da due componenti
  1. Il modulo `SenseC`
  2. La configurazione `SenseAppC`
- Periodicamente acquisisce un dato da `DemoSensorC` e ne visualizza i bit meno significativi attraverso i led

`DemoSensorC`

Esempio: `Sense`

Applicazione giocattolo che

- È formata da due componenti
  1. Il modulo `SenseC`
  2. La configurazione `SenseAppC`
- Periodicamente acquisisce un dato da `DemoSensorC` e ne visualizza i bit meno significativi attraverso i led

`DemoSensorC`

- Fornisce l'interfaccia `Read<uint16_t>`

Esempio: `Sense`

Applicazione giocattolo che

- È formata da due componenti
  1. Il modulo `SenseC`
  2. La configurazione `SenseAppC`
- Periodicamente acquisisce un dato da `DemoSensorC` e ne visualizza i bit meno significativi attraverso i led

`DemoSensorC`

- Fornisce l'interfaccia `Read<uint16_t>`
- Rappresenta un generico trasduttore la cui natura dipende dal sensore o dalla sensorbaord usata (es. sensore di luce, sensore di voltaggio)

SenseC

```
module SenseC
{
  uses {
    interface Boot;
    interface Leds;
    interface Timer<TMilli>;
    interface Read<uint16_t>;
  }
}
implementation
{
```

SenseC

```
 // sampling frequency in binary milliseconds
#define SAMPLING_FREQUENCY 100

event void Boot.booted() {
  call Timer.startPeriodic(SAMPLING_FREQUENCY);
}

event void Timer.fired()
{
  call Read.read();
}
```

SenseC

```
event void Read.readDone(error_t result, uint16_t data)
{
  if (result == SUCCESS){
    if (data & 0x0004)
      call Leds.led2On();
    else
      call Leds.led2Off();
    if (data & 0x0002)
      call Leds.led1On();
    else
      call Leds.led1Off();
    if (data & 0x0001)
      call Leds.led0On();
    else
      call Leds.led0Off();
  }
}
```

SenseAppC

```
configuration SenseAppC {}
implementation {

  components SenseC, MainC, LedsC, new TimerMilliC(),
             new DemoSensorC() as Sensor;

  SenseC.Boot -> MainC;
  SenseC.Leds -> LedsC;
  SenseC.Timer -> TimerMilliC;
  SenseC.Read -> Sensor;
}
```

Componenti generic

- Per default i componenti in TinyOS sono singleton

Componenti generic

- Per default i componenti in TinyOS sono singleton
  - Esiste una sola istanza per l'intera applicazione

Componenti generic

- Per default i componenti in TinyOS sono singleton
  - Esiste una sola istanza per l'intera applicazione
- Questo approccio va bene per componenti che sono astrazioni di funzionalità di basso livello (hardware)

Componenti generic

- Per default i componenti in TinyOS sono singleton
  - Esiste una sola istanza per l'intera applicazione
- Questo approccio va bene per componenti che sono astrazioni di funzionalità di basso livello (hardware)
- Spesso è necessario che ci siano più istanze di uno stesso componente (strutture dati)

Componenti generic

- Per default i componenti in TinyOS sono singleton
  - Esiste una sola istanza per l'intera applicazione
- Questo approccio va bene per componenti che sono astrazioni di funzionalità di basso livello (hardware)
- Spesso è necessario che ci siano più istanze di uno stesso componente (strutture dati)
- Generic component possono essere istanziati più volte (con `new`)

Componenti generic

- Per default i componenti in TinyOS sono singleton
  - Esiste una sola istanza per l'intera applicazione
- Questo approccio va bene per componenti che sono astrazioni di funzionalità di basso livello (hardware)
- Spesso è necessario che ci siano più istanze di uno stesso componente (strutture dati)
- Generic component possono essere istanziati più volte (con `new`)

```
generic configuration TimerMilliC() {
  provides interface Timer<TMilli>;
}
implementation {
  ...
}
```

Stack di comunicazione

- Active Message (AM) è la tecnologia di comunicazione di TinyOS

Stack di comunicazione

- Active Message (AM) è la tecnologia di comunicazione di TinyOS
  - comunicazione radio (mote-to-mote)

Stack di comunicazione

- Active Message (AM) è la tecnologia di comunicazione di TinyOS
  - comunicazione radio (mote-to-mote)
  - comunicazione seriale (pc-to-mote)

Stack di comunicazione

- Active Message (AM) è la tecnologia di comunicazione di TinyOS
  - comunicazione radio (mote-to-mote)
  - comunicazione seriale (pc-to-mote)
- Nel comunicazione radio AM fornisce un protocollo per la trasmissione e ricezione di pacchetti

Stack di comunicazione

- Active Message (AM) è la tecnologia di comunicazione di TinyOS
  - comunicazione radio (mote-to-mote)
  - comunicazione seriale (pc-to-mote)
- Nel comunicazione radio AM fornisce un protocollo per la trasmissione e ricezione di pacchetti
- Ogni pacchetto è caratterizzato da un AM Type (intero a 8 bit)

Stack di comunicazione

- Active Message (AM) è la tecnologia di comunicazione di TinyOS
  - comunicazione radio (mote-to-mote)
  - comunicazione seriale (pc-to-mote)
- Nel comunicazione radio AM fornisce un protocollo per la trasmissione e ricezione di pacchetti
- Ogni pacchetto è caratterizzato da un AM Type (intero a 8 bit)
  - svolge la stessa funzione del numero di porta in UDP

Stack di comunicazione

- Active Message (AM) è la tecnologia di comunicazione di TinyOS
  - comunicazione radio (mote-to-mote)
  - comunicazione seriale (pc-to-mote)
- Nel comunicazione radio AM fornisce un protocollo per la trasmissione e ricezione di pacchetti
- Ogni pacchetto è caratterizzato da un AM Type (intero a 8 bit)
  - svolge la stessa funzione del numero di porta in UDP
  - identifica il tipo di evento da generare nel nodo che riceve il pacchetto

message_t

- Un pacchetto è rappresentano dalla struttura `message_t`
- Per accedere ai campi di `message_t` si devono usare i comandi delle interfacce `Packet` e `AMPacket`

```
interface Packet {

   command uint8_t payloadLength(message_t* msg);

   command void* getPayload(message_t* msg, uint8_t len);
   ...

}
```

message_t

- Un pacchetto è rappresentano dalla struttura `message_t`
- Per accedere ai campi di `message_t` si devono usare i comandi delle interfacce `Packet` e `AMPacket`

```
interface AMPacket {

  command am_addr_t address();

  command am_addr_t destination(message_t* amsg);

  ...
}
```

Interfaccia `AMSend`

```
interface AMSend {
  command error_t send(am_addr_t addr,
       message_t* msg, uint8_t len);

  command error_t cancel(message_t* msg);

  event void sendDone(message_t* msg, error_t error);

  command uint8_t maxPayloadLength();

  command void* getPayload(message_t* msg, uint8_t len);

}
```

- la trasmissione di un pacchetto è effettuata tramite `send/sendDone`
- l'indirizzo di un nodo è stabilito durante l'installazione dell'applicazione sul sensore (generalmente `TOS_NODE_ID`)

Interfaccia `Receive`

```
interface Receive {

  event message_t* receive(message_t* msg,
   void* payload, uint8_t len);

}
```

- `receive` è invocato quando la radio riceve un pacchetto

Interfaccia `SplitControl`

- La radio prima di essere utilizzata deve essere attivata

```
interface SplitControl
{
  command error_t start();

  event void startDone(error_t error);

  command error_t stop();

  event void stopDone(error_t error);
}
```

- L'accensione della radio è fatta tramite la split-operation `start/startDone`
- Lo spegnimento della radio è fatta tramite la split-operation `stop/stopDone`

Componenti per la comunicazione radio

- `ActiveMessageC` è il componente che realizza lo stack di comunicazione su ogni specifica architettura
- Altri componenti per la comunicazione

  `AMSenderC` componente per l'invio dei messaggi
  `AMReceiverC` componente per la ricezione dei messaggi

Esempio: `BlinkToRadio`

Una semplice applicazione che svolge due compiti

1. Periodicamente (ogni 250 ms) incrementa un contatore e invia il suo valore ai vicini

Composta da due componenti

Esempio: `BlinkToRadio`

Una semplice applicazione che svolge due compiti

1. Periodicamente (ogni 250 ms) incrementa un contatore e invia il suo valore ai vicini
2. Quando riceve un messaggio da un vicino visualizza i bit meno significativi del suo contatore usando i led

Composta da due componenti

Esempio: `BlinkToRadio`

Una semplice applicazione che svolge due compiti

1. Periodicamente (ogni 250 ms) incrementa un contatore e invia il suo valore ai vicini
2. Quando riceve un messaggio da un vicino visualizza i bit meno significativi del suo contatore usando i led

Composta da due componenti

- Il modulo `BlinkToRadioC`

Esempio: `BlinkToRadio`

Una semplice applicazione che svolge due compiti

1. Periodicamente (ogni 250 ms) incrementa un contatore e invia il suo valore ai vicini
2. Quando riceve un messaggio da un vicino visualizza i bit meno significativi del suo contatore usando i led

Composta da due componenti

- Il modulo `BlinkToRadioC`
  - usa le interfacce `Boot`, `Leds`, `Timer`, `Packet`, `AMPacket`, `AMSend`, `Receive`, `SplitControl`

Esempio: `BlinkToRadio`

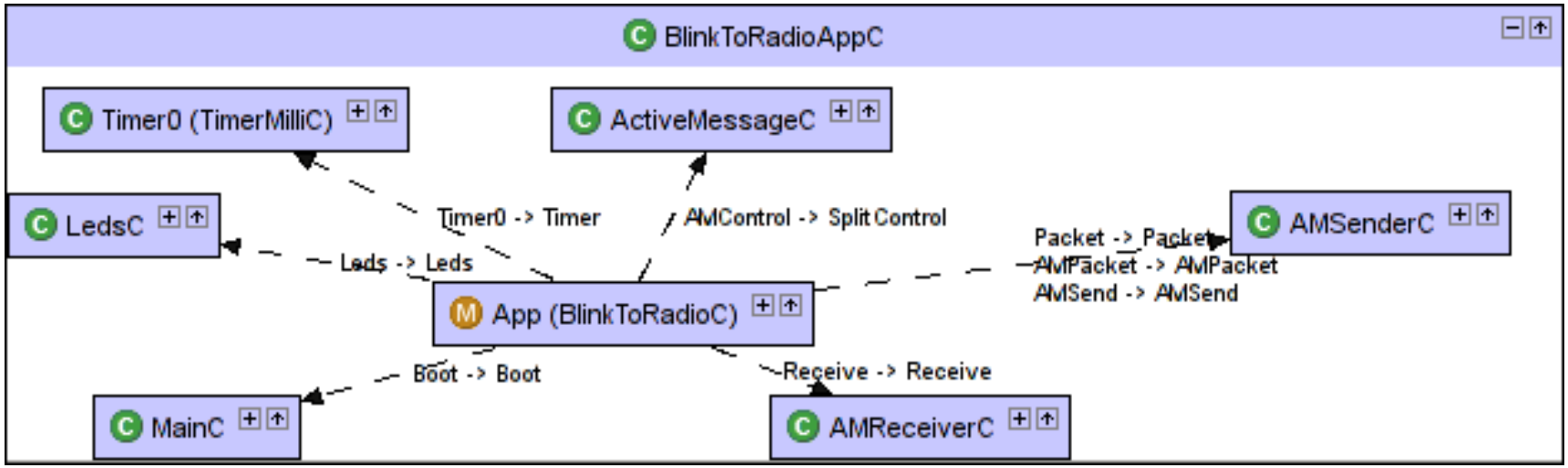
Una semplice applicazione che svolge due compiti

1. Periodicamente (ogni 250 ms) incrementa un contatore e invia il suo valore ai vicini
2. Quando riceve un messaggio da un vicino visualizza i bit meno significativi del suo contatore usando i led

Composta da due componenti

- Il modulo `BlinkToRadioC`
  - usa le interfacce `Boot`, `Leds`, `Timer`, `Packet`, `AMPacket`, `AMSend`, `Receive`, `SplitControl`
- La configurazione `BlinkToRadioAppC`

Struttura dell'applicazione



BlinkToRadioC

1. Inizializzare la radio
2. Avviare il timer
3. Periodicamente incrementare il contatore e inviare il suo nuovo valore ai vicini
4. Ogni volta che viene ricevuto un messaggio visualizzare i bit meno significativi del suo contenuto sui led

Inizializzare la radio e il timer

```
event void Boot.booted() {
  call AMControl.start();
}

event void AMControl.startDone(error_t err) {
  if (err == SUCCESS) {
    call Timer0.startPeriodic(TIMER_PERIOD_MILLI);
  }
  else {
    call AMControl.start();
  }
}
```

`AMControl` è un nome alternativo per `SplitControl`

Incremento del contatore e invio

```
event void Timer0.fired() {
  counter++;
  if (!busy) {
    BlinkToRadioMsg* btrpkt =  (BlinkToRadioMsg*)
       (call Packet.getPayload(&pkt, sizeof(BlinkToRadioMsg)));
    if (btrpkt == NULL) {
      return;
    }
    btrpkt->nodeid = TOS_NODE_ID;
    btrpkt->counter = counter;
    if (call AMSend.send(AM_BROADCAST_ADDR,
        &pkt, sizeof(BlinkToRadioMsg)) == SUCCESS) {
      busy = TRUE;
    }
  }
}
```

Ricezione di un messaggio

```
event message_t* Receive.receive(message_t* msg,
                 void* payload, uint8_t len){
  if (len == sizeof(BlinkToRadioMsg)) {
    BlinkToRadioMsg* btrpkt = (BlinkToRadioMsg*)payload;
    setLeds(btrpkt->counter);
  }
  return msg;
}
```

BlinkToRadioAppC

```
configuration BlinkToRadioAppC {
}
implementation {
  ...
   components BlinkToRadioC as App;
   components ActiveMessageC;
   components new AMSenderC(AM_BLINKTORADIO);
   components new AMReceiverC(AM_BLINKTORADIO);
  ...
   App.Packet -> AMSenderC;
   App.AMPacket -> AMSenderC;
   App.AMControl -> ActiveMessageC;
   App.AMSend -> AMSenderC;
   App.Receive -> AMReceiverC;
}
```

Comunicazione mote-pc

La tecnologia degli Active Message viene utilizzata anche per implementare la comunicazione tra sensore e pc

Radio	Seriale
`ActiveMessageC`	`SerialActiveMessageC`
`AMSenderC`	`SerialAMSenderC`
`AMReceiverC`	`SerialAMReceiveC`

BlinkToSerialAppC

```
configuration BlinkToSerialAppC {
}
implementation {
 ...
  components BlinkToRadioC as App;
  components SerialActiveMessageC;
  components new SerialAMSenderC(AM_BLINKTORADIO);
  components new SerialAMReceiverC(AM_BLINKTORADIO);
 ...
  App.Packet -> SerialAMSenderC;
  App.AMPacket -> SerialAMSenderC;
  App.AMControl -> SerialActiveMessageC;
  App.AMSend -> SerialAMSenderC;
  App.Receive -> SerialAMReceiverC;
}
```

Comunicazione pc-mote

TinyOS fornisce un insieme di API e di strumenti per costruire applicazione che interagiscono con la rete di sensori

- API Java
- Parser di pacchetti

API Java

Interfaccia per l'invio e la ricezione di messaggi

- `net.tinyos.message.MoteIF` classe base da utilizzare per inviare e ricevere dati dalla rete
- `net.tinyos.message.Message` classe che rappresenta un messaggio scambiato con la rete
- `net.tinyos.message.MessageListener` handler per gestire la ricezione di un messaggio
- `net.tinyos.packet.PhoenixSource` interfaccia che rappresenta il canale di comunicazione con la rete

Parser di pacchetti

- Fornisce un tool chiamato `mig`
- Genera una sottoclasse di `net.tinyos.message.Message` per il parsing dei messaggi dalla rete

```
nx_struct test_serial_msg {
  nx_uint16_t counter;
};
```

Parser di pacchetti

- Fornisce un tool chiamato `mig`
- Genera una sottoclasse di `net.tinyos.message.Message` per il parsing dei messaggi dalla rete

```
public class TestSerialMsg extends Message {
  public TestSerialMsg() {
    ...
  }

  public int get_counter() {
    ...
  }

  public void set_counter(int value) {
    ...
  }
  ...
}
```

Conclusione

Abbiamo introdotto le basi della programmazione in TinyOS

- Componenti e interfacce
- Sensing
- Comunicazione con AM
  - radio
  - seriale

Altri strumenti messi a disposizione da TinyOS

- Protocollo di dissemination di dati
- Protocollo di collection di dati
- Memorizzazione dati
- Tossim