SAPIENZA
UNIVERSITÀ DI ROMA

Laboratorio di Sistemi Wireless

7 Maggio 2012

A. Cammarano, A.Capossele, D. Spenza

SAPIENZA
UNIVERSITÀ DI ROMA

Contatti

Cammarano: cammarano@di.uniroma1.it
Capossele: capossele@di.uniroma1.it
Spenza: spenza@di.uniroma1.it

Tel: 06-49918430
Room: 333

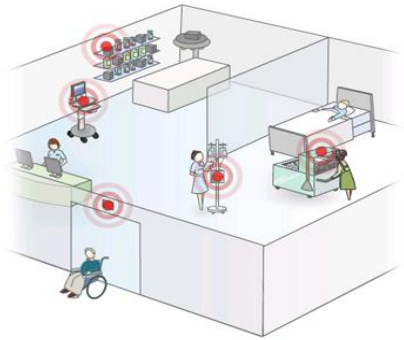
Slides: www.dsi.uniroma1.it/~spenza/

Outline

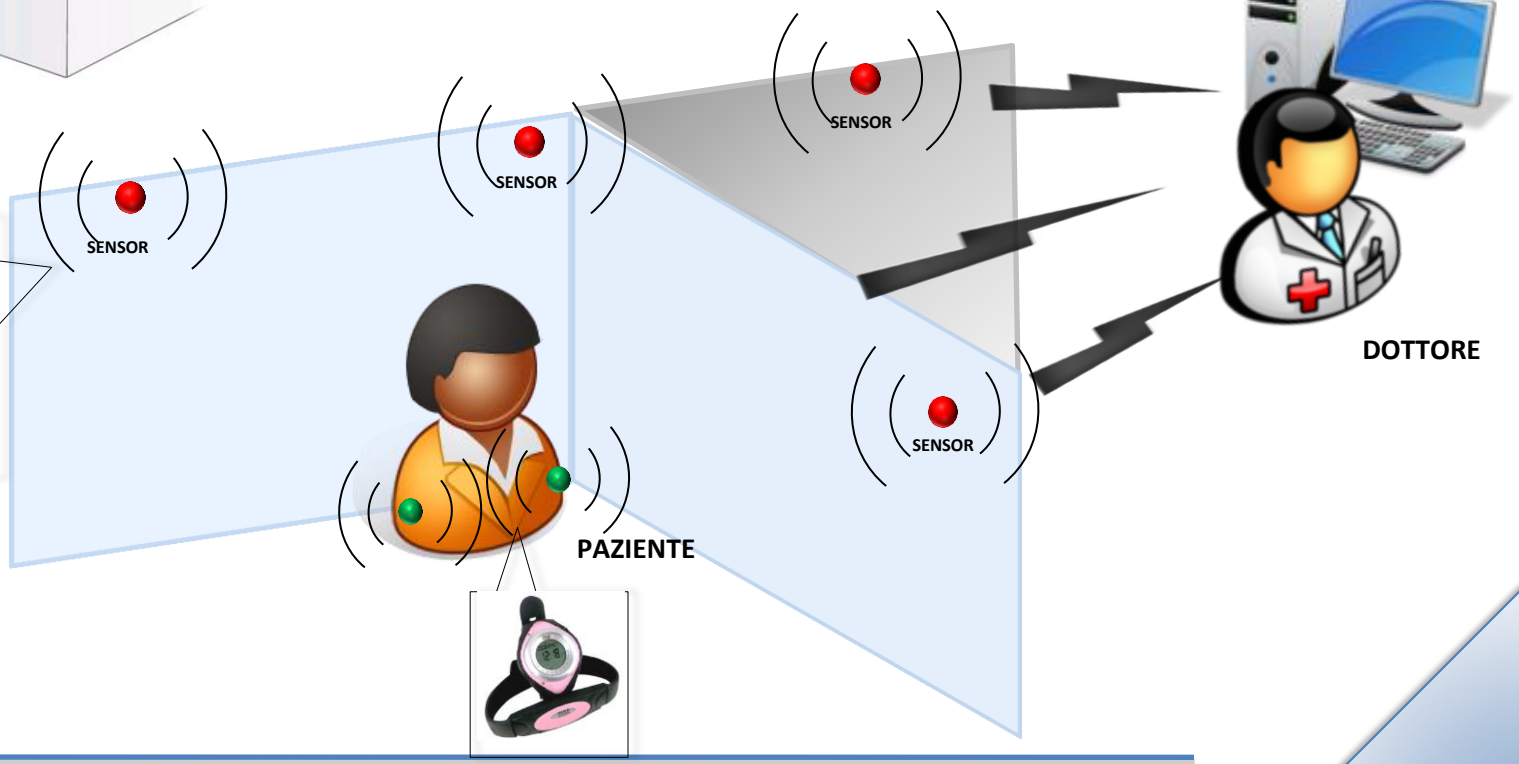
- SENSES lab
- WSN: introduzione e scenari applicativi
- Linguaggio NesC
- TinyOS
- Una semplice applicazione: Blink

- Grande interesse stanno riscuotendo sistemi noti come *Wireless Sensor Networks*, basati su comunicazione wireless
  - Le WSN trovano applicazioni sempre nuove in settori che vanno dalla telemedicina, alla protezione ambientale, al monitoraggio dei beni culturali, ecc…
- Sicurezza – Harvesting – Cognitive Network

CHIRON



Correlazione tra lo stato del pazionete e le condizioni ambientali



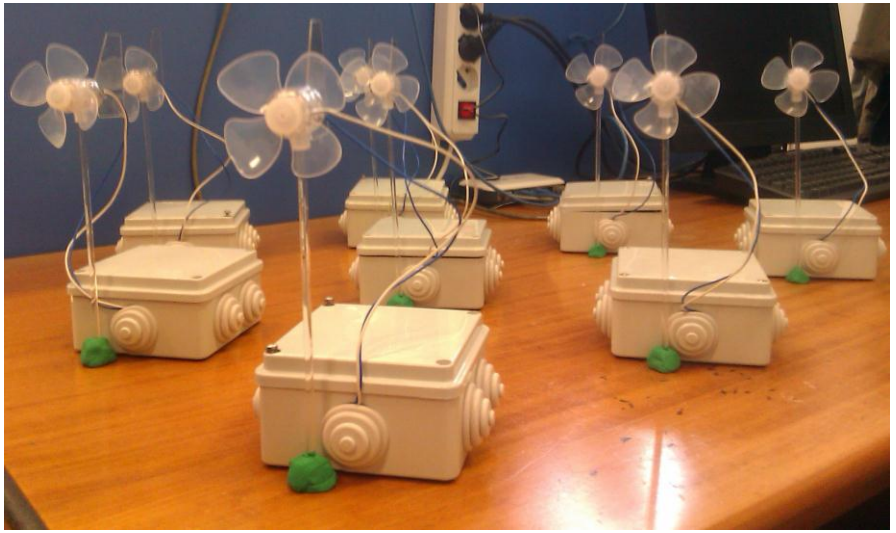
SENSORI AMBIENTALI

- **Luce**
- **Temperatura**
- **Umidità**
- **Inquinamento**

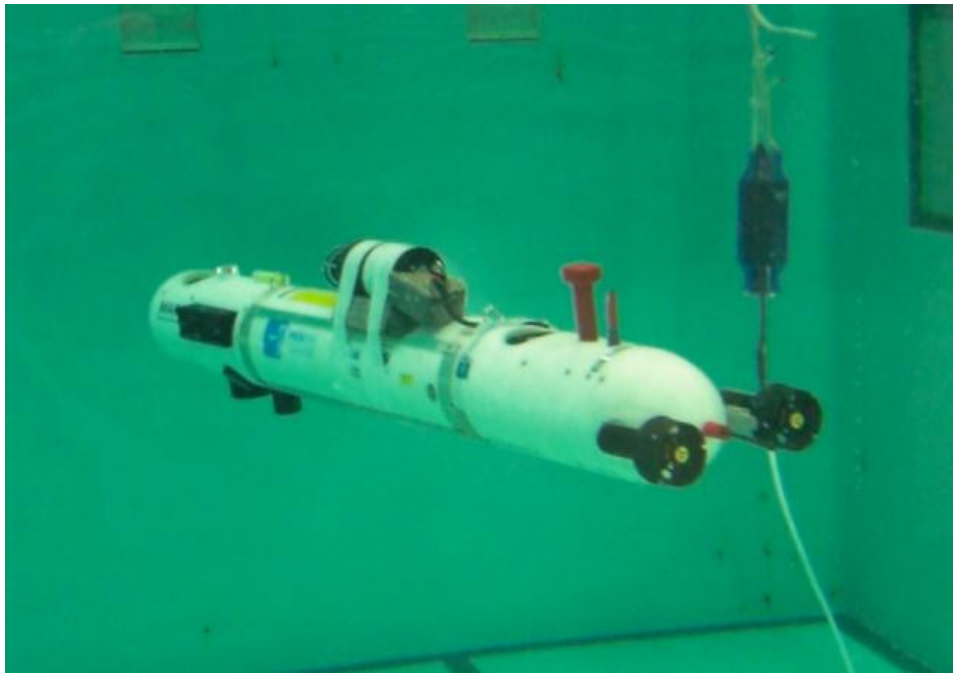
GENESI

Green Wireless Sensor Networks

Utilizzo di fonti di energia rinnovabile
per estendere la vita della rete



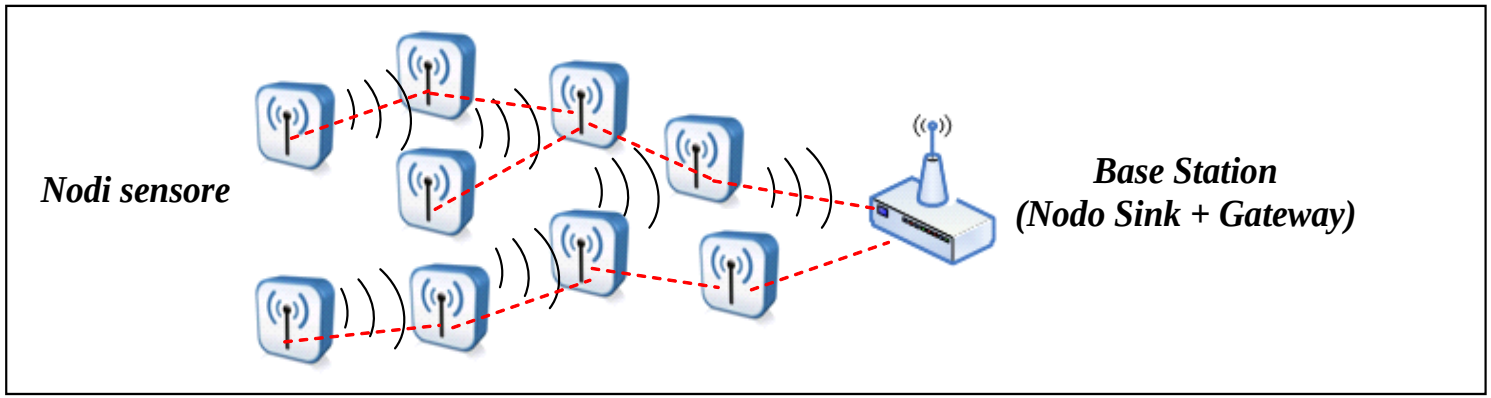
Underwater sensor networks



Comunicazione acustica

- Monitoraggio olio, gas, CO2
- Prevenzione disastri naturali
- Composizione chimica fondali
- …

Architettura



- Reti *wireless* formate da dispositivi autonomi, i *nodi sensore*, distribuiti nello spazio e dotati di sensori, che cooperano per fornire una funzione di monitoraggio dell'ambiente circostante
  - Elementi chiave
    - Nodo sensore (*nodo*, *mote*) e *Base Station*
    - Comunicazioni *wireless a corto raggio (multi-hop)*

Caretteristiche

- A prescindere dalla specifica realizzazione e dai singoli componenti utilizzati per i nodi, le WSN presentano caratteristiche peculiari e distintive
  - Autoconfigurazione
  - Scalabilità
  - Robustezza
  - Flessibilità
  - Basso costo
  - Efficienza energetica

Problematiche

- Modello del sistema
  - Nodi fisici vs. Componenti funzionali
  - Computazione locale vs. Comunicazione
- Architettura hardware dei nodi
  - Microprocessore/Microcontrollore
    - IBM 8051, Atmel ATmega128L, XScale PXA271, TI MSP430,…
  - Chipset per la comunicazione e la corrispondente antenna
    - ChipCon CC1100 e CC2420
  - Bus di comunicazione locale
    - SPI, I2C o proprietari

Problematiche

- Protocolli di comunicazione
  - I protocolli di comunicazione impiegati nelle WSN sono diversi ed eterogenei
    - Ai livelli più bassi, ci si appoggia a protocolli standard
      - IEEE 802.15.4, 6lowpan, Bluetooth, ecc…
  - Un aspetto essenziale è la riconfigurabilità dinamica della topologia della rete che, in molti casi reali, è soggetta a cambiamenti
    - Specifici algoritmi di routing per WSN
      - SPIN (Sensor Protocols for Information via Negotiation), Directed Diffusion, Rumor Routing, Q-RC (Q-learning Routing and Compression), ecc…

Risparmio energetico

- Nodo
  - Progettazione componenti e architettura hw/sw
  - Meccanismi per la (auto)gestione del consumo energetico
- Rete
  - Protocolli *energy-aware*
- Sistema
  - Applicazioni *energy-aware*

Sicurezza

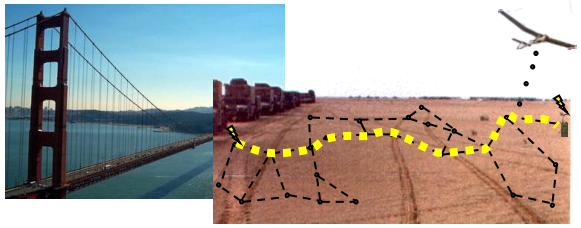
- Nodo
  - Algoritmi di crittografia
- Rete
  - Sistemi crittografici
  - *Intrusion Detection & Monitoring Systems*
- Sistema
  - Sistemi crittografici

Altre applicazioni

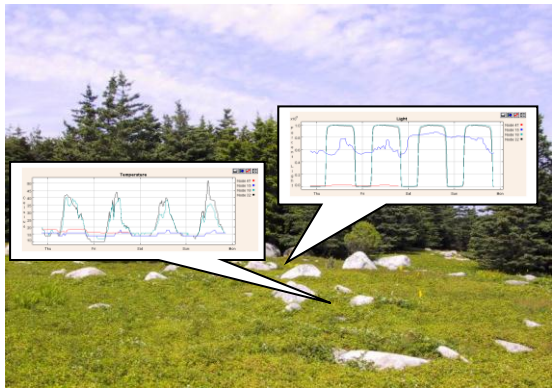
- Campi applicativi
  - Militari
  - Ambientali
  - Mediche
  - Domotiche
  - Commerciali
  - Industriali
  - Civili
  - Ecc…

Altre applicazioni

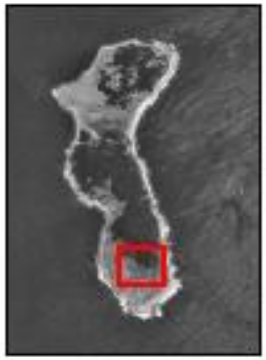
- Monitoraggio ambientale
- Monitoraggio strutturale



- Esempi
  - Rilevamento incendi/alluvioni
  - Monitoraggio frane
  - Agricoltura di precisione
  - Monitoraggio qualità acqua/aria
  - Monitoraggio animali protetti
  - Monitoraggio ponti/edifici



Altre applicazioni



- Un esempio..
  - Great Duck Island
    - Piccola isola degli USA, Maine
      - Studio sui Petrel, una particolare specie di uccello marino
    - 2002
      - Disposizione di 32 mote con vari sensori
      - Monitoraggio del microclima attorno ai nidi
      - Informazioni veicolate verso la rete Internet in tempo reale

Altre applicazioni

- Monitoraggio continuo di pazienti
  - In ambulatorio o in ospedale
  - In casa per pazienti cronici o anziani
- Adattamento dell'ambiente al paziente
- Raccolta di dati clinici

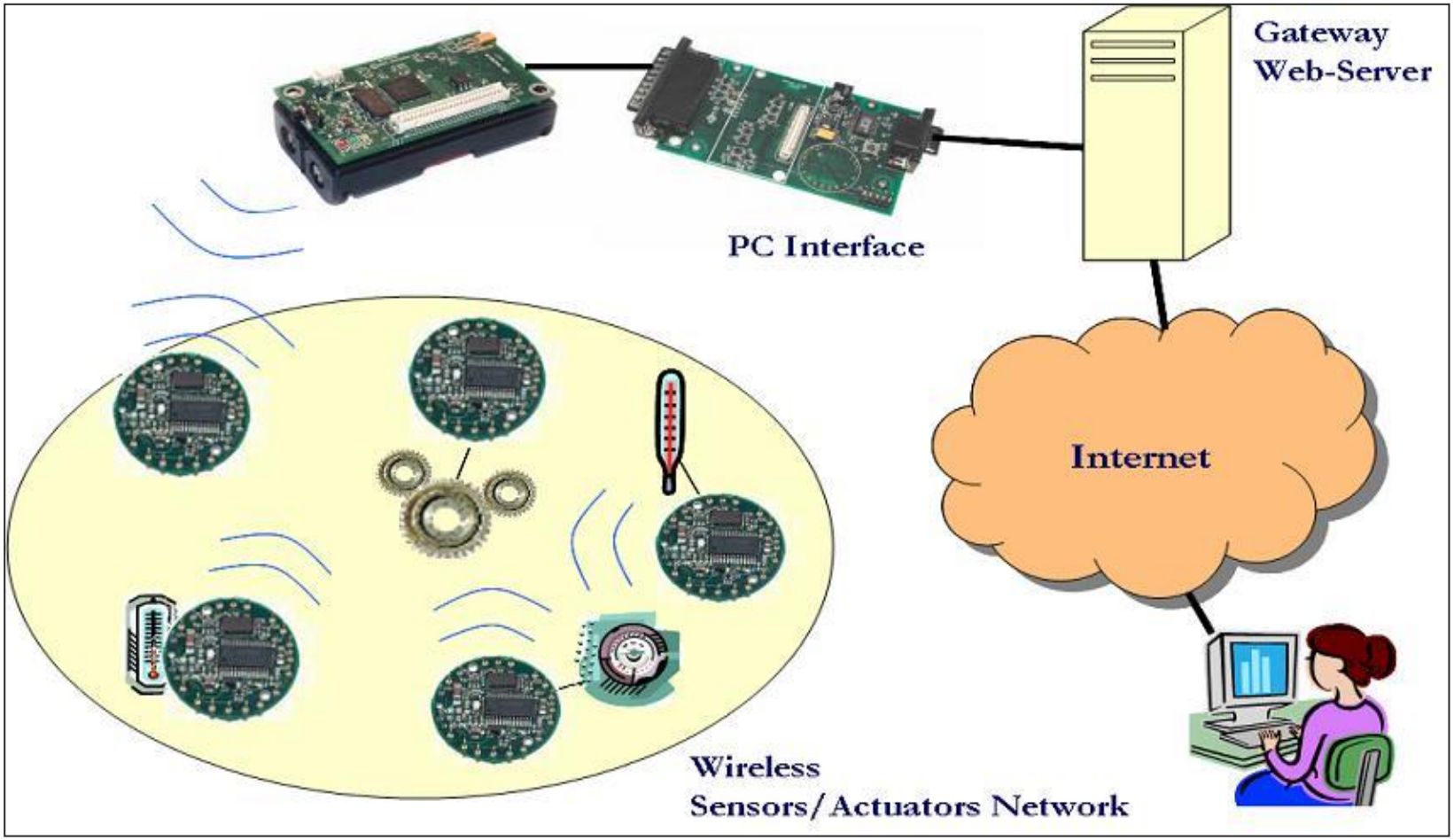
Altre applicazioni

- Gestione locale e remota della casa
  - Automazione degli elettrodomestici
  - Luminosità
    - Interruttori wireless
  - Temperatura
    - Termostati wireless
- Controllo della casa

Altre applicazioni

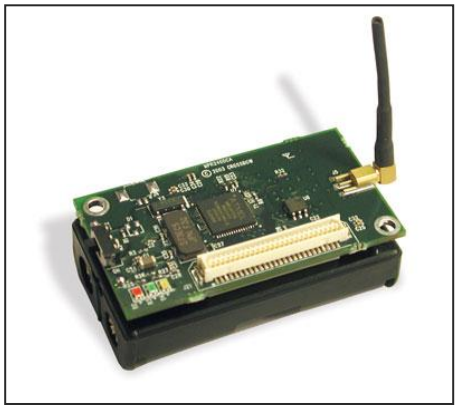
- Monitoraggio di attrezzature e munizioni
- Sorveglianza del campo di battaglia
  - Tipico ambiente ostile
- Riconoscimento del tipo di attacco
  - Nucleare, biologico, chimico

Tecnologie Hardware



Mica2

- Elaborazione: microcontrollore Atmel ATmega128L
  - MPU: 8-bit RISC (0-8 MHz)
  - Memoria
    - Programma: 128K Bytes Flash
    - Lavoro: 4K Bytes SRAM
  - ADC, UART, GPIO, I2C, SPI, Timer



- Comunicazione: *Transceiver* Chipcon CC1000
  - 868/915 MHz, 38.4 kbps, range 30-100 m)
- Memorizzazione locale: Flash 512 KB

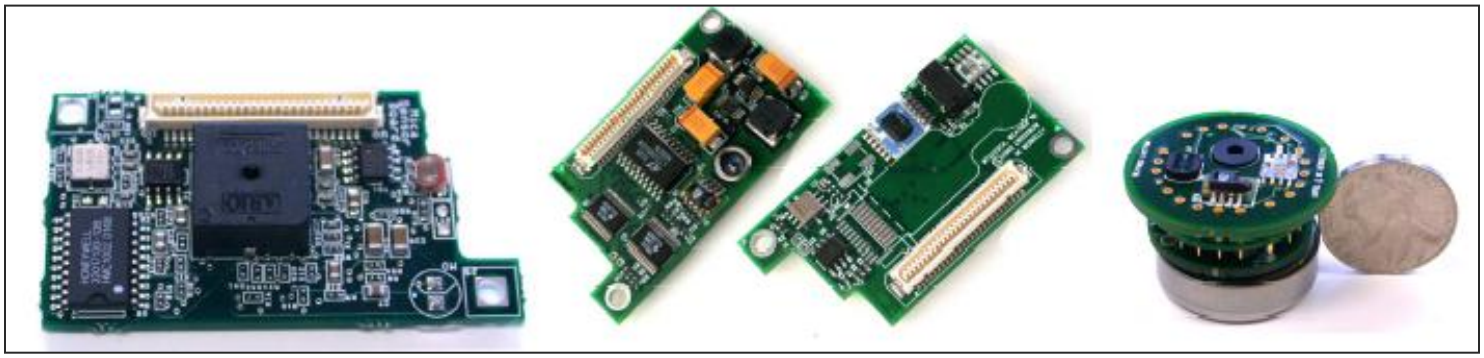
TelosB

- Elaborazione: microcontrollore TI MSP430
  - MPU: 16-bit RISC (0-8 MHz)
  - Memoria
    - Programma: 48K Bytes Flash
    - Lavoro: 10K Bytes SRAM
  - ADC, UART, GPIO, I2C, SPI, Timer

- Comunicazione: *Transceiver* Chipcon CC2420
  - IEEE 802.15.4 (2,4 GHz, 250 kbps, range 20-100 m)
- Memorizzazione locale: Flash 1024 KB

Sensor Board

- Differenti tipologie di “sensori”
  - Luce, temperatura, pressione, umidità
  - Accelerometri, magnetometri, misuratori di distanza
  - Microfono, foto/videocamera, ricevitori GPS



Altre applicazioni

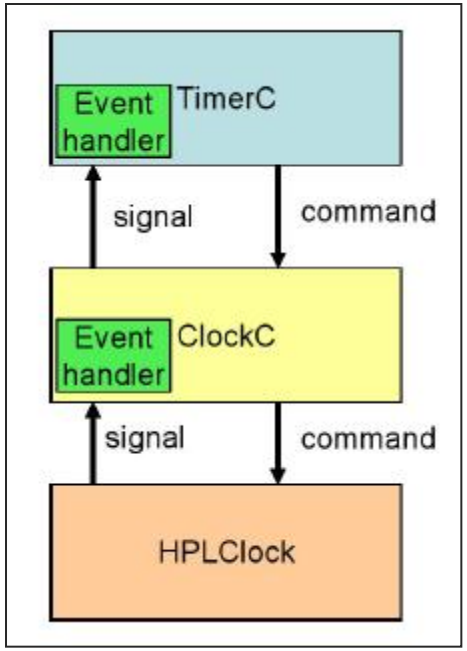
- Base Station
  - Collegamento *wired* PC-nodo (*wireless* con altri nodi)
    - Parallela, seriale (*MIB 510/520*), ethernet

nesC

- Estensione linguaggio C per *networked embedded systems*
  - Programmazione per componenti
    - Moduli e Interfacce
    - Configurazioni
    - Collegamenti
  - Programmazione orientata agli eventi
    - Comandi utilizzati/offerti
    - Eventi segnalati/gestiti
      - Corrispondono ad interruzioni hw o sw

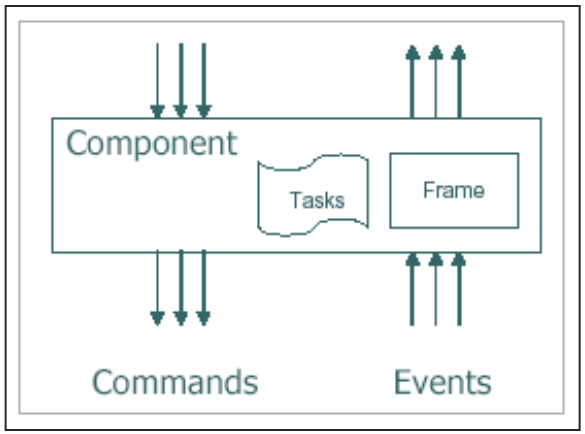
nesC

- I componenti utilizzatori sono collegati (*wired*) ai fornitori
  - Si crea una gerarchia di componenti
  - I comandi
    - Vanno verso il basso
    - Ritornano al chiamante
  - Gli eventi
    - Vanno verso l'alto
    - Ritornano al segnalatore



nesC

- Oltre a comandi/eventi esiste il concetto di *task* che introduce due livelli di priorità nell'esecuzione del codice
  - Eventi
    - Codice ad alta priorità
  - Task
    - Codice a bassa priorità



nesC

- Task
  - Computazione medio-lunga in background
  - Atomici rispetto ad altri task
  - Interrotti (*preemption*) dagli eventi
- Eventi
  - Alta reattività
  - Breve durata (demandano lavoro ai task)
- Task ed eventi possono richiamare comandi che tipicamente danno luogo ad una esecuzione di tipo *split-phase*
  - Questo permette un parallelismo a grana fine

- Es. Componente di tipo Modulo

```
module XYZ1
{
  provides interface Interface1 as I1;
  provides interface Interface2;
  …
  uses interface Interface3 as I3;
  uses interface Interface2;
  …
}
implementation
{
 command void I1.cmd1() {
     …
  }

  event void Interface2.ev1() {
     …
  }
}
```

- Es. Componente di tipo Configurazione

```
configuration XYZ
{
 …
}
implementation
{
  components XYZ1, XYZ2;

  …
  XYZ1.Interface1 -> XYZ2.Interface1;
  XYZ1.Interface2 -> XYZ2;
  …
}
```

TinyOS

- Si tratta di una libreria di componenti nesC che offre alcune funzionalità tipiche di un sistema operativo
  - Scheduler
  - Driver
    - Componenti per leggere dati da sensori
    - Componenti per inviare comandi ad attuatori
    - Componenti per gestire le comunicazioni radio
  - Power Management
    - Mantenere i dispositivi HW nel più basso stato di potenza dissipata possibile
  - No concetti di *kernel*, processi, gestione della memoria

TinyOS

- Per ogni applicazione esiste inoltre la configurazione *top-level* che contiene il componente *MainC*
  - Fornisce i servizi base di TinyOS (≈ 200 Bytes)

```
configuration BlinkAppC
{
}
implementation
{
  components MainC, BlinkC, LedsC;
  …

  BlinkC -> MainC.Boot;
  …
  BlinkC.Timer0 -> Timer0;
  …
  BlinkC.Leds -> LedsC;
}
```

TinyOS

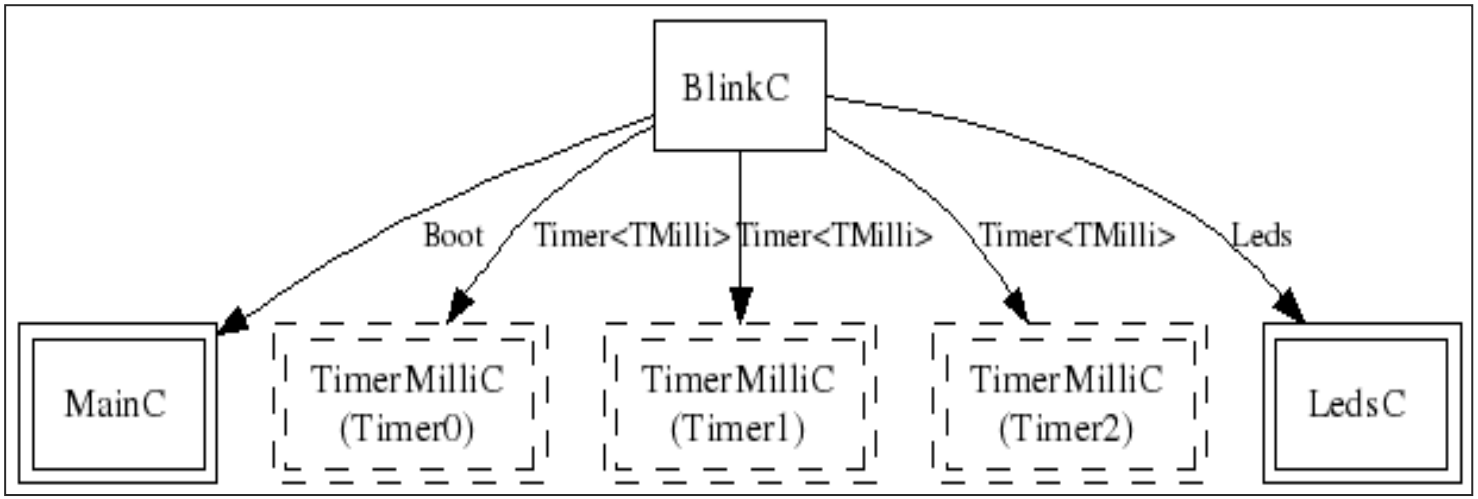
- La filosofia di sviluppo di una applicazione è del tipo
  - ***Hurry Up and Sleep!!!***
    - Ovvero cercare di rimanere nello stato di *sleep* il più possibile per risparmiare energia
    - Quando il nodo è risvegliato da un evento, cercare di eseguire il lavoro associato il più in fretta possibile e poi tornare a dormire
      - *Interrupt-driven & Split-phase*

Esempio - Blink

- Fa lampeggiare i 3 led di un nodo sensore
  - I led lampeggiano a frequenze di 1Hz, 2Hz, e 4Hz
- Componenti applicativi
  - *BlinkAppC (Configuration)*
  - *BlinkC (Module)*
- Componenti di sistema
  - *MainC, LedsC, TimerMilliC*

Esempio - Blink

- BlinkAppC: grafo dei componenti



	Singleton	Generic
Module	BlinkC	VirtualizeTimerC
Configuration	MainC	TimerMilliC (Timer0)

Esempio - Blink

- BlinkAppC.nc

```
configuration BlinkAppC
{
}
implementation
{
  components MainC, BlinkC, LedsC;
  components new TimerMilliC() as Timer0;
  components new TimerMilliC() as Timer1;
  components new TimerMilliC() as Timer2;


  BlinkC -> MainC.Boot;

  BlinkC.Timer0 -> Timer0;
  BlinkC.Timer1 -> Timer1;
  BlinkC.Timer2 -> Timer2;
  BlinkC.Leds -> LedsC;
}
```

Esempio - Blink

- BlinkC.nc

```
#include "Timer.h"

module BlinkC
{
  uses interface Timer<TMilli> as Timer0;
  uses interface Timer<TMilli> as Timer1;
  uses interface Timer<TMilli> as Timer2;
  uses interface Leds;
  uses interface Boot;
}
implementation
{
```

Esempio - Blink

- BlinkC.nc

```
event void Boot.booted()
{
  call Timer0.startPeriodic( 250 );
  call Timer1.startPeriodic( 500 );
  call Timer2.startPeriodic( 1000 );
}
```

Esempio - Blink

- BlinkC.nc

```
event void Timer0.fired()
  {
    dbg("BlinkC", "Timer 0 fired @ %s.\n", sim_time_string());
    call Leds.led0Toggle();
  }

  event void Timer1.fired()
  {
    dbg("BlinkC", "Timer 1 fired @ %s \n", sim_time_string());
    call Leds.led1Toggle();
  }

  event void Timer2.fired()
  {
    dbg("BlinkC", "Timer 2 fired @ %s.\n", sim_time_string());
    call Leds.led2Toggle();
  }
}
```

Esempio – Blink counter

- BlinkC.nc

```
uint8_t counter = 0;

event void Boot.booted()
{
  call Timer0.startPeriodic( 1024 );
}
```

	8 bits	16 bits	32 bits	64 bits
signed	int8_t	int16_t	int32_t	int64_t
unsigned	uint8_t	uint16_t	uint32_t	uint64_t

Esempio – Blink counter

- BlinkC.nc

```
event void Timer0.fired()
{
    counter++;
    if (counter & 0x1) {
      call Leds.led0On();
    }
    else {
      call Leds.led0Off();
    }
    if (counter & 0x2) {
      call Leds.led1On();
    }
    else {
      call Leds.led1Off();
    }
    if (counter & 0x4) {
      call Leds.led2On();
    }
    else {
      call Leds.led2Off();
    }
}
```