TinyOS

Saymon Castro de Souza

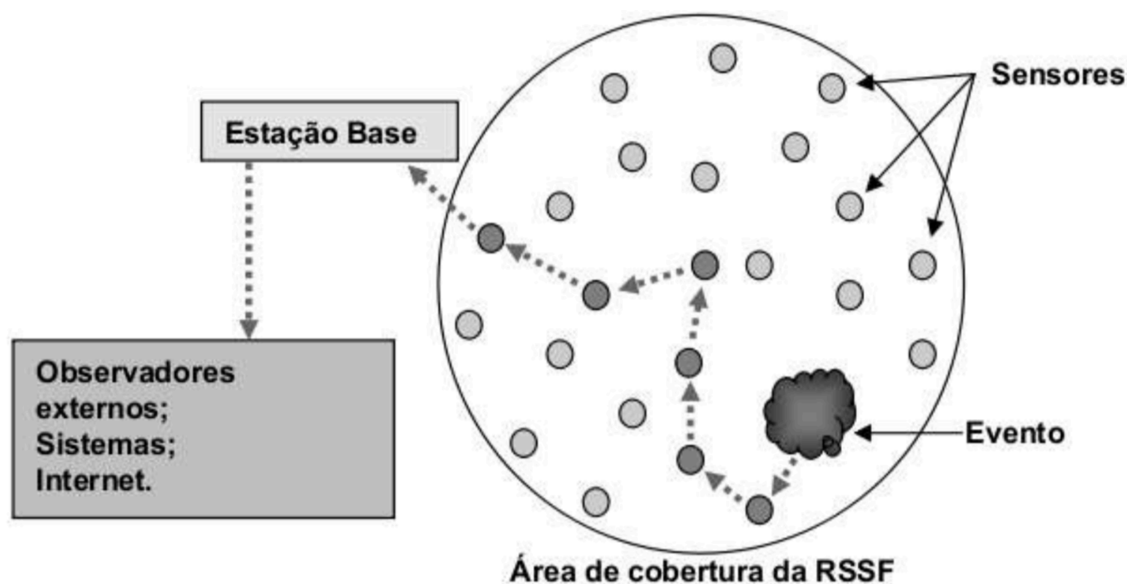
Orientador: Prof. Dr. José Gonçalves Pereira Filho

Agenda

- Introdução
- nesC
- TinyOS
- Preparação do ambiente
- Implementação

RSSF

- Formadas por um grande número de pequenos **sensores inteligentes**.
  - Objetivo: Detectar e transmitir alguma característica do meio físico.



RSSF

- Desafios:
  - Energia
    - Dispõe apenas de uma bateria com vida limitada, dando uma vida bastante curta ao nó.
    - Nós são espalhados em regiões geográficas tornando difícil a reposição de uma possível bateria esgotada ou mesmo de um nó danificado.

RSSF

- Desafios
  - Recursos limitados
  - Comunicação sem fio
  - O sistema operacional deve ser eficiente em termos de consumo de memória, processador e energia.

RSSF

- Desafios
  - Robustez: uma vez instalado no ambiente, a rede de sensores deve ser executada sem intervenção humana por meses ou anos.
  - Diversas implementações de serviços: as aplicações deveriam ser capazes de escolher entre múltiplas implementações de, por exemplo, roteamento multihop.
  - Adaptabilidade aos requisitos das aplicações: As aplicações possuem requisitos muito diferentes em termos de tempo de vida, comunicação, sensibilidade, etc.

Exemplo nós sensores

- Telosb
  - Microcontrolador TI MSP430, 16 bits
  - 10kB de RAM, 48kB de Flash
  - Rádio CC2420, padrão IEEE802.15.4, 250kbps
- Micaz
  - Microcontrolador Atmel ATmega128, 8 bits
  - 4kB de RAM, 128kB de Flash
  - Rádio CC2420, padrão IEEE802.15.4, 250kbps

Exemplo de nós sensores

Micaz

Telosb

nesC

nesC

- Linguagem orientada a componentes com um modelo de execução baseado em eventos

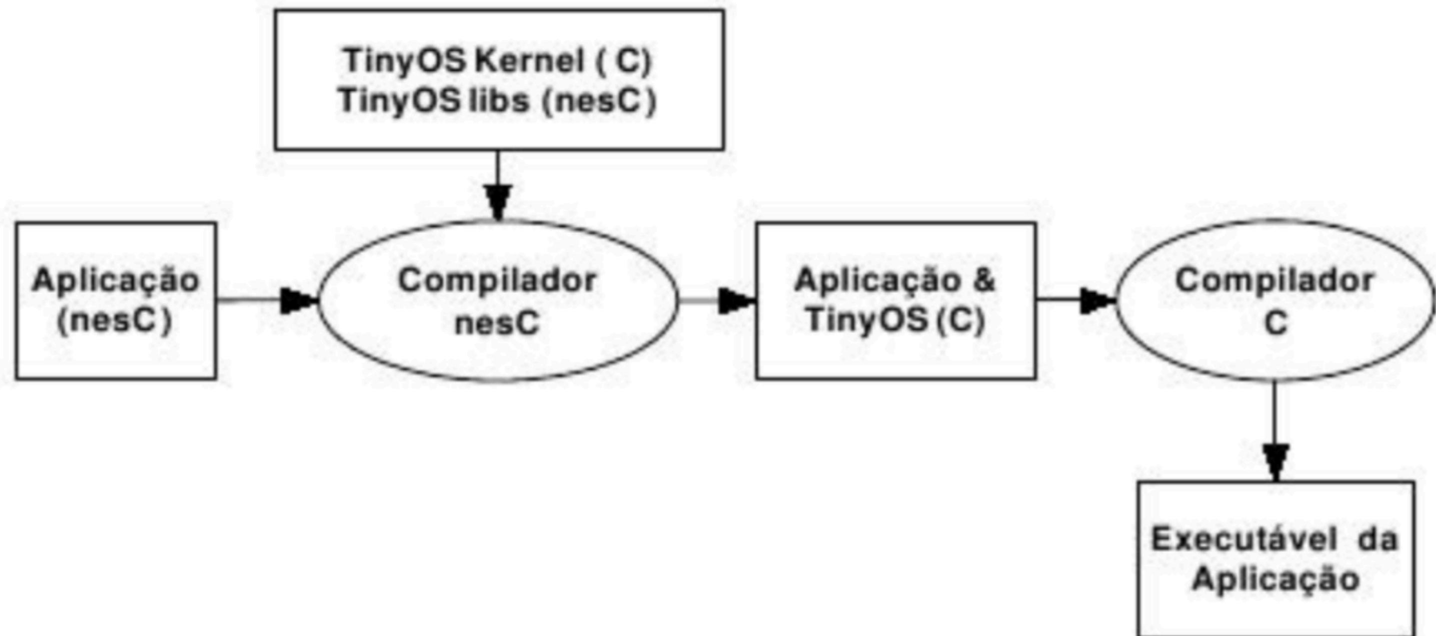
nesC

- Os componentes encapsulam estado e interagem através de interfaces bem definidas
  - Semelhante à O. O., porém não há herança, alocação dinâmica ou *Dynamic Dispatch.*
- Não suporta recursos que inviabilizem análise estática (ponteiros, alocação de memória dinâmica...)

nesC

- Modelo orientado a componentes
  - Possível construir a aplicação utilizando um conjunto de componentes existentes e adicionando algum código extra necessário à execução do objetivo final da aplicação.
  - Assim, ao invés de ser um SO de propósito geral, TinyOS comporta-se como uma framework que permite a construção de um novo TinyOS específico, evitando o uso de componentes que não são necessários para a execução da aplicação.

Processo de geração de código executável.



TinyOS

TinyOS

- TinyOS é um sistema Operacional dirigido a eventos projetado para redes de sensores sem fio que possui recursos muito limitados
- O modelo de programação adotado prioriza fortemente o tratamento dessas restrições em detrimento da simplicidade oferecida para o desenvolvimento de aplicações.
- TinyOS atualmente é usado em vários projetos científicos e comercias.

TinyOS - características

- Arquitetura baseada em componentes
- Concorrência baseada em Tarefas e Eventos
- Operações divididas em fase

TinyOS - características

- Arquitetura baseada em **componentes**
  - TinyOS disponibiliza um conjunto de componentes de sistemas reusáveis que a aplicação deve conectar através de uma especificação.
    - SO segmentado em peças “encaixáveis”.
    - Facilita o projeto de aplicações
    - Facilita comunicação entre módulos
    - Encapsulamento
    - Elimina componentes não utilizados

TinyOS - características

- Concorrência baseada em **Tarefas e Eventos**
  - Tarefas são mecanismos computacionais adiados.
  - Uma vez escalonada, uma tarefa é executada até terminar.
  - Como não há preempção entre as tarefas, o código executado por elas deve ser curto.
  - O escalonador de tarefas é configurado para escalonar as tarefas para execução sempre que o processador torna-se disponível.
  - A política padrão é FIFO (First In First Out).

TinyOS - características

- Concorrência baseada em **Tarefas e Eventos**
  - Um **evento** sinaliza o término de um serviço, como por exemplo, o envio de uma mensagem.

TinyOS - características

- **Operações divididas em fase**
  - Todas as operações de longa duração devem ser dividas em fases.
  - Início e término das tarefas são sinalizados
  - Exemplo: envio de um pacote.
    - Um componente pode invocar o comando *send* para iniciar a transmissão de uma mensagem de rádio.
    - O componente de comunicação, ao término da transmissão, gera o evento *sendDone.*

Instalação do ambiente

Instalação do ambiente

- Virtualbox
  - Ubuntu 12.04.5
  - Atualizar as listas do repositório
  - Instalar os pacotes de compilação essenciais
  - Instalar os pacotes de compilação essenciais do python
  - Instalar o tinyos-2.1.2
  - Alterar as variáveis de ambiente

Implementação

nesC

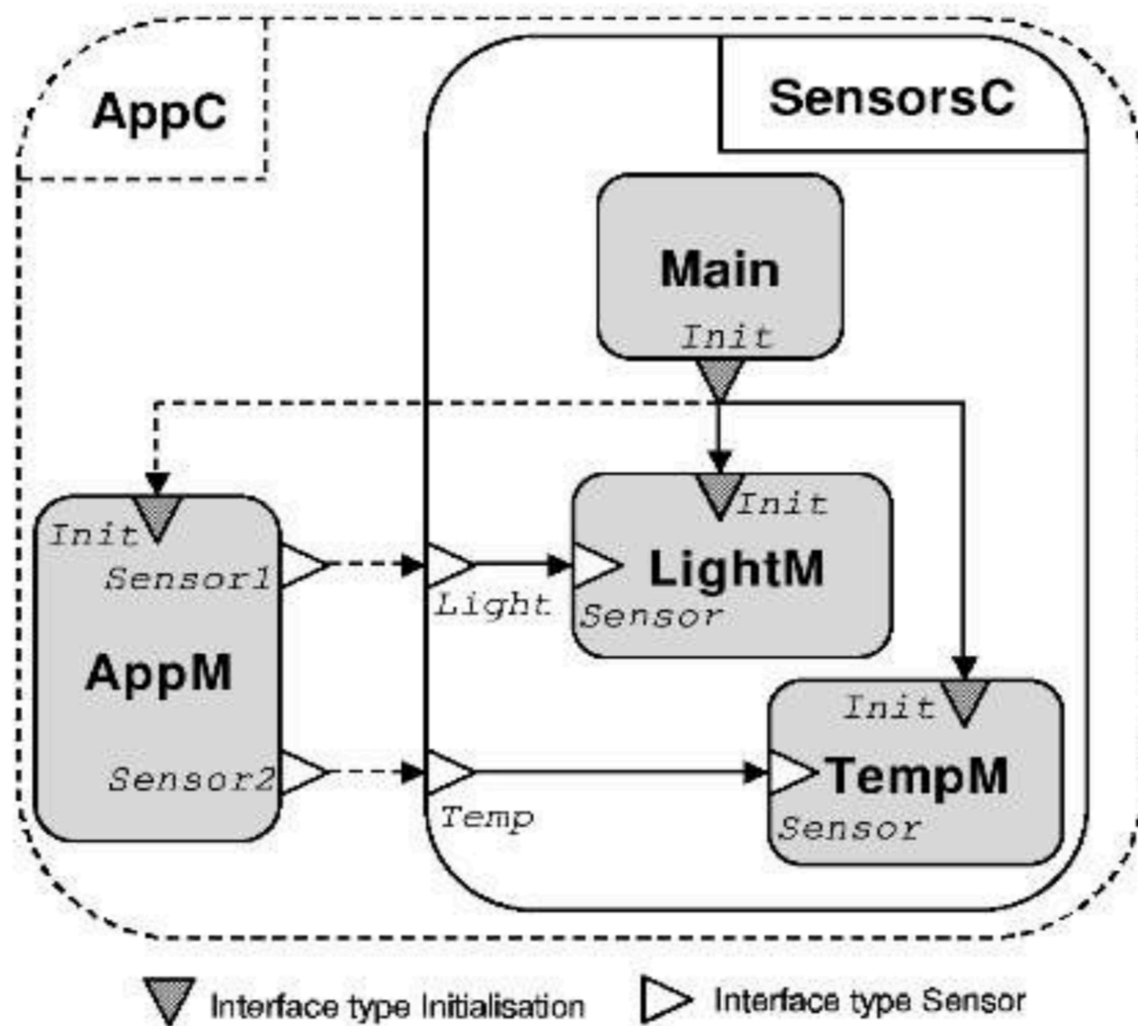
- Extensão da linguagem C
- Criada para incorporar os conceitos e o modelo de execução do TinyOS
- Logo, é orientado a eventos e formado por um conjunto de componentes

nesC

- Principais conceitos:
  - Separação de construção e composição
  - Especificação do comportamento de componentes através de um conjunto de interfaces
  - Componentes são ligados (“wired”) através das interfaces e configurações

nesC

- NesC define dois tipos de **componentes**:
  - **Módulos:** Componentes implementados com código NesC.
  - **Configurações:** Componentes implementados através da ligação de componentes uns aos outros.



AppC
SensorsC
Main
Init
Init
LightM
Light
Sensor
Sensor1
Init
AppM
Sensor2
Init
TempM
Temp
Sensor
Interface type Initialisation
Interface type Sensor

As interfaces definem as interações entre um componente provedor de serviço e um componente que usará esse serviço.

```
interface Initialize {
    command void init();
}
```

Comandos são invocações que o usuário pode fazer para o componente provedor do serviço.

```
interface Sense {
    command void sense();
    event void senseDone();
}
```

Split-phase operations

Eventos são do provedor ao usuário

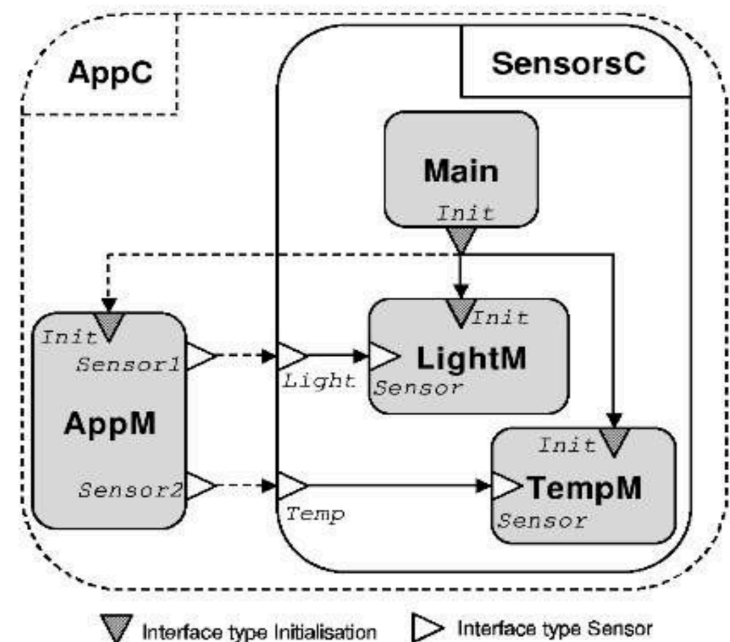
```
module { ... }
implementation {
        int sum = 0;
        task void startSensing() {
                call Sensor1.sense();
        }
        command void Init.init(){
                post startSensing();
        }
        event void Sensor1.senseDone(int val) {
                sum += val;
                call Sensor2.sense();
        }
        event void Sensor2.senseDone(int val){
                sum += val;
        }
}
```

nesC

- Configurações
  - São componentes cuja função é a de ligar (*wire*) componentes uns aos outros de acordo com as interfaces fornecidas e usadas por esses componentes

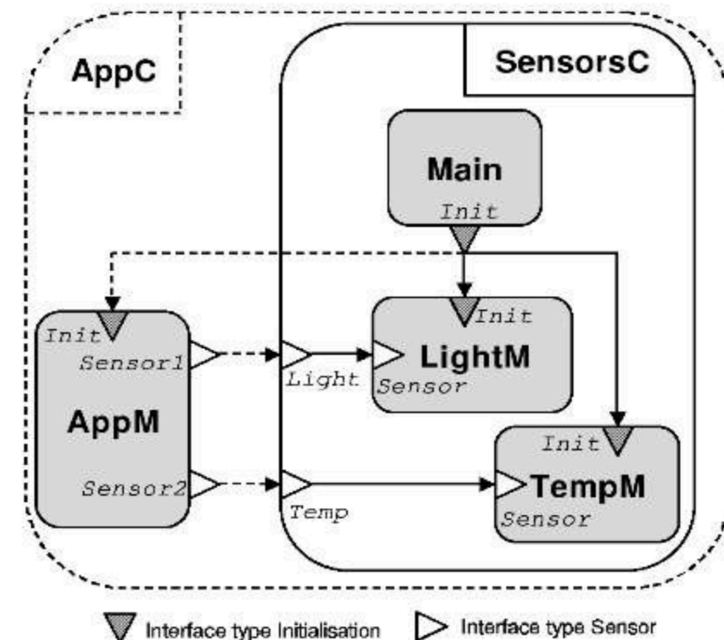
```
configuration SensorsC{
        provides interface Sense as Light;
        provides interface Sense as Temp;
}
implementation {
        components Main, LightM, TempM;

        Main.Init -> LightM.Init;
        Main.Init -> TempM.init;
        Light = LightM.Sensor;
        Temp = TempM.Sensor;
}
```



```
Configuration AppC { }
implementation {
        components Main, AppM, SensorsC;

        Main.Init → AppM.Init;
        AppM.Sensor1 -> SensorsC.Light;
        AppM.Sensor2 -> SensorsC.Temp;
}
```



nesC

- Em resumo, um componente deve fornecer implementações para os comandos das interfaces que ele fornece e para os eventos das interfaces que ele usa.

```
module { ... }
implementation {
    int sum = 0;
    task void startSensing() {
        call Sensor1.sense();
    }
    command void Init.init(){
        post startSensing();
    }
    event void Sensor1.senseDone(int val) {
        sum += val;
        call Sensor2.sense();
    }
    event void Sensor2.senseDone(int val){
        sum += val;
    }
}
```

Obrigado !