Universidade Federal de Santa Catarina
Laboratório de Integração de Software e Hardware

Gerenciamento de Energia em Sistemas de Sensoriamento Remoto

Arliones Stevert Hoeller Junior

Lucas Francisco Wanner, Augusto Born de Oliveira,
Roger Kreutz Immich, Antônio Augusto Medeiros Fröhlich

arliones@lisha.ufsc.br
http://www.lisha.ufsc.br/~arliones

17 de Julho de 2006

Sistemas de Sensoriamento

- Dispositivos de sensoriamento remoto são pequenos com recursos limitados
  - Processadores de 8-bits a 3-4 Mhz
  - Limitações de memória (programa e dados)
  - Mecanismos de comunicação

Bateria = Tempo de vida do sistema

Infraestrutura de gerência de energia

- Suporte variado em hardware
  - Modos de operação (CPU, UART, ADC, etc)
  - Frequency and Voltage Scaling
  - Contadores de eventos
- Padrões para gerência: APM e ACPI
  - Interface Hardware/Software
  - Não utilizados em sistemas embarcados: complexidade
- Soluções para sistemas de sensoriamento
  - Específicas para cada contexto
  - Parciais, não contemplando todos dispositivos de hardware
  - Comprometem a portabilidade

Suporte *power-aware* em Sistemas de Sensoriamento

- "Sistemas operacionais" com HALs simples
- Não oferecem componentes de alto-nível
- Focam na gerência de consumo da CPU
- Migrações implementadas pela aplicação
- Resultado:
  - Eficácia contestável
    - CPU não consome tanta energia quanto periféricos
  - Portabilidade comprometida
    - Detalhes do hardware explícitos na aplicação

Exemplo: TinyOS

- Interface StdControl
  - Permite apenas ligar e desligar dispositivos
  - Modos de baixo consumo ignorados
- HPLPowerManagement
  - Granularidade fina na gerência da CPU
- MAC (radios)
  - Implementações *power-aware*
  - Falta de configurabilidade
- Problemas
  - Falta de padronização das interfaces
  - Configurabilidade do hardware inexplorada

Proposta

- API unificada
- Propagação hierarquica de mensagens
- API em componentes de alto-nível

```
file.power(OFF);
communicator.power(OFF);
display.power(OFF)
```

- Formalização das migrações de modo de operação através de Redes de Petri

API de gerência de energia

- Interface simples e uniforme
  - `power(Mode s)`
  - `Mode power()`
- Modos de operação configuráveis:
  - Componentes exportam todos modos possíveis
  - Programador seleciona equivalentes aos modos globais (FULL, LIGHT, STANDBY, OFF)

Implementação

- *Application-Oriented System Design* (AOSD)
  - Metodologia multi-paradigma
  - Sistemas otimizados para aplicações
- Interface de gerência e controle de instâncias
  - Consumo energia é característica não-funcional [Lohmann, 2005]
  - Modelado como aspecto, implementado com recursos de metaprogramação estática

Implementação: API

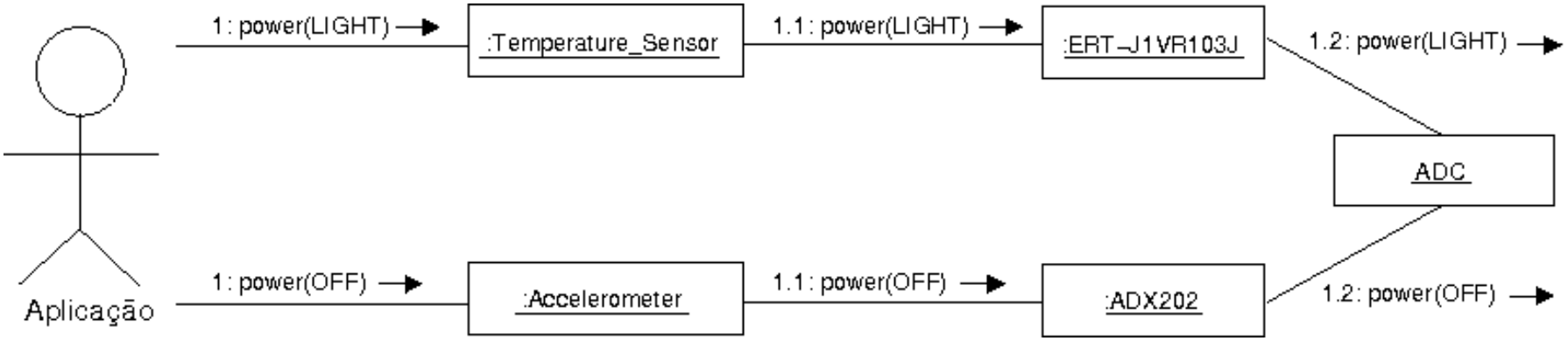
```
template<typename T>
class Power_Manager: public T {
  //...
  public:
    void methodX() {
      if(_op_mode > _prev_op_mode)
        T::power(_prev_op_mode);
      T::methodX();
    }

    char power() { return _op_mode; }

    void power(char mode) {
      _prev_op_mode = _op_mode;
      T::power(mode);
      _op_mode = mode;
    }
  //...
};
```

Implementação: modos de operação configuráveis

```
template<>
struct Traits<AVR8>: public Traits<void> {

    static const bool Power_Management = true;

    static const char FULL = AVR8::FULLY;

    static const char LIGHT = AVR8::IDLE;

    static const char STANDBY = AVR8::POWER_SAVE;

    static const char OFF = AVR8::POWER_DOWN;

};
```

Propagação de mensagens

- Componentes organizados hierarquicamente
- Programador usa abstrações de alto-nível (e.g., `Thread`, `Temperature_Sensor`)
- Mensagens de troca de modo de operação são propagadas através da hierarquia



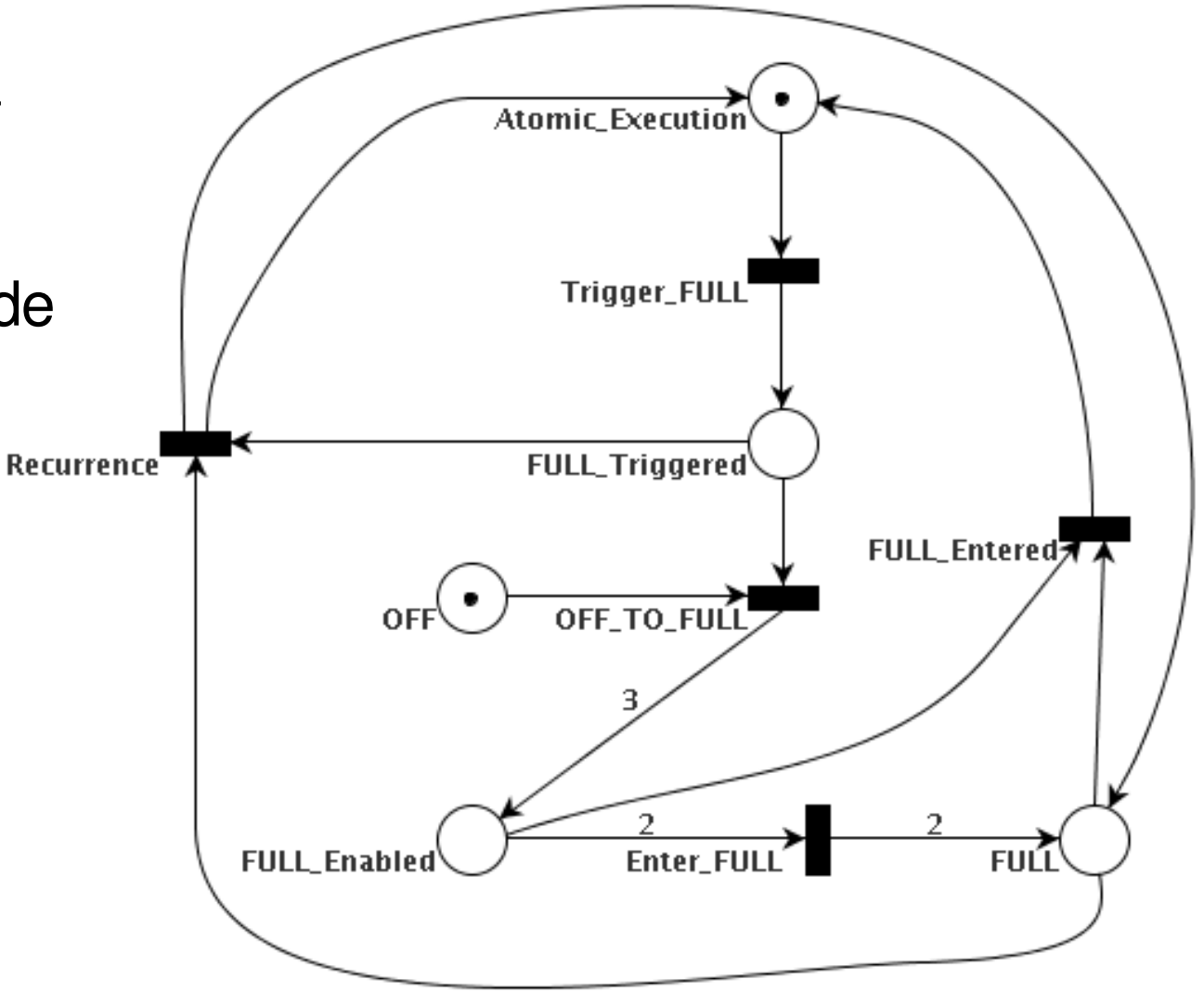
Redes de Transição de modos de operação

- Redes de Petri
- Definem condições para as transições
- Representam as propagações de mensagens

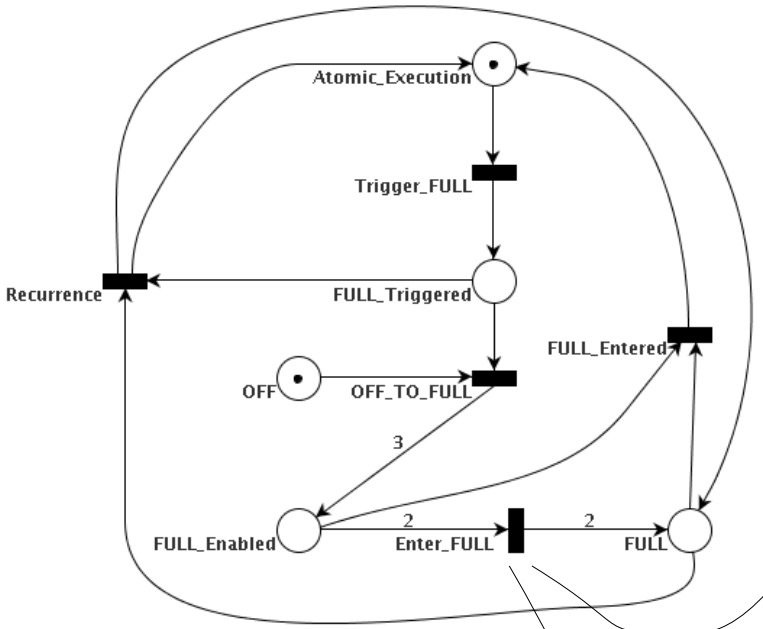
- Estrutura generalizada de transição
  - OFF, STANDBY, LIGHT, FULL
  - Grafo de alcançabilidade finito => análise dos estados possíveis

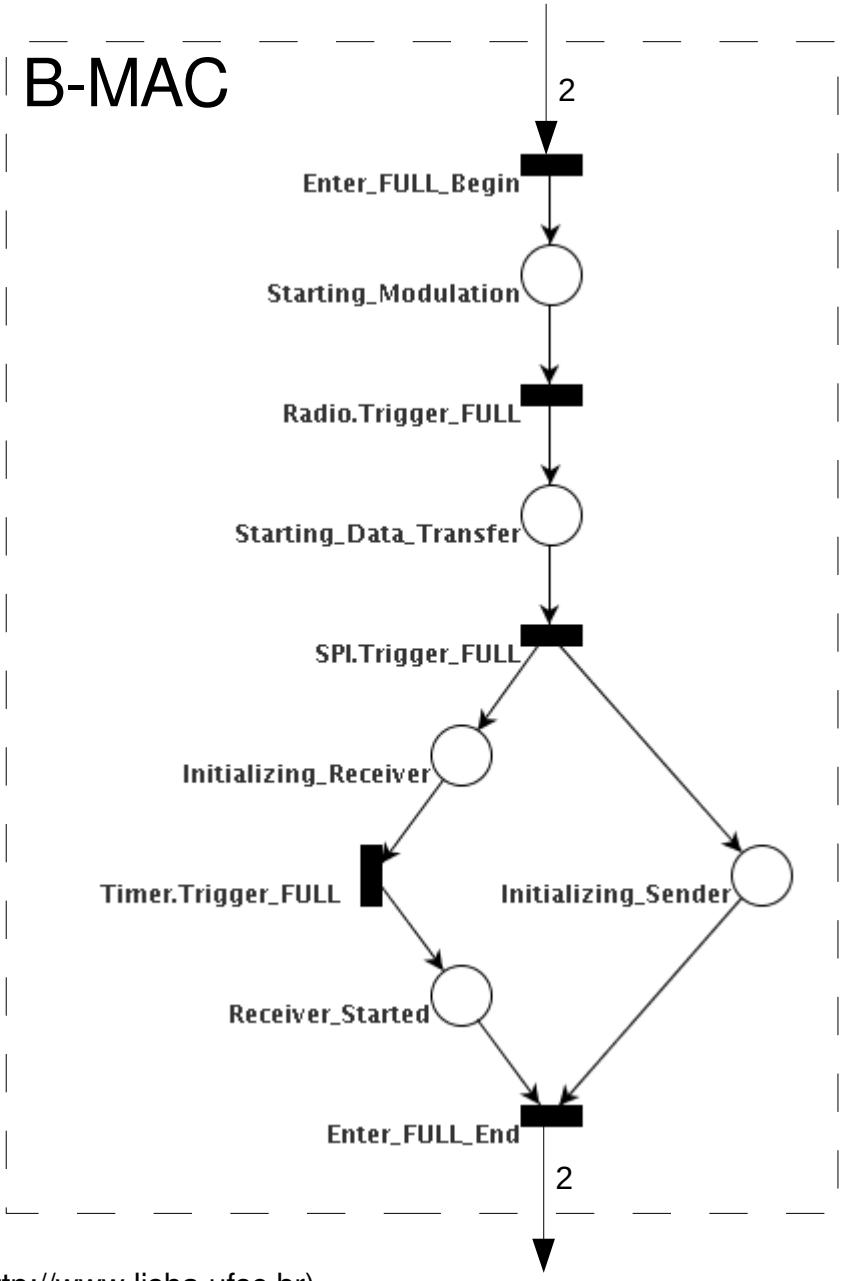
Rede generalizada de transição

- Versão simplificada
- A versão completa cobre todos modos de operação



Redes Hierárquicas

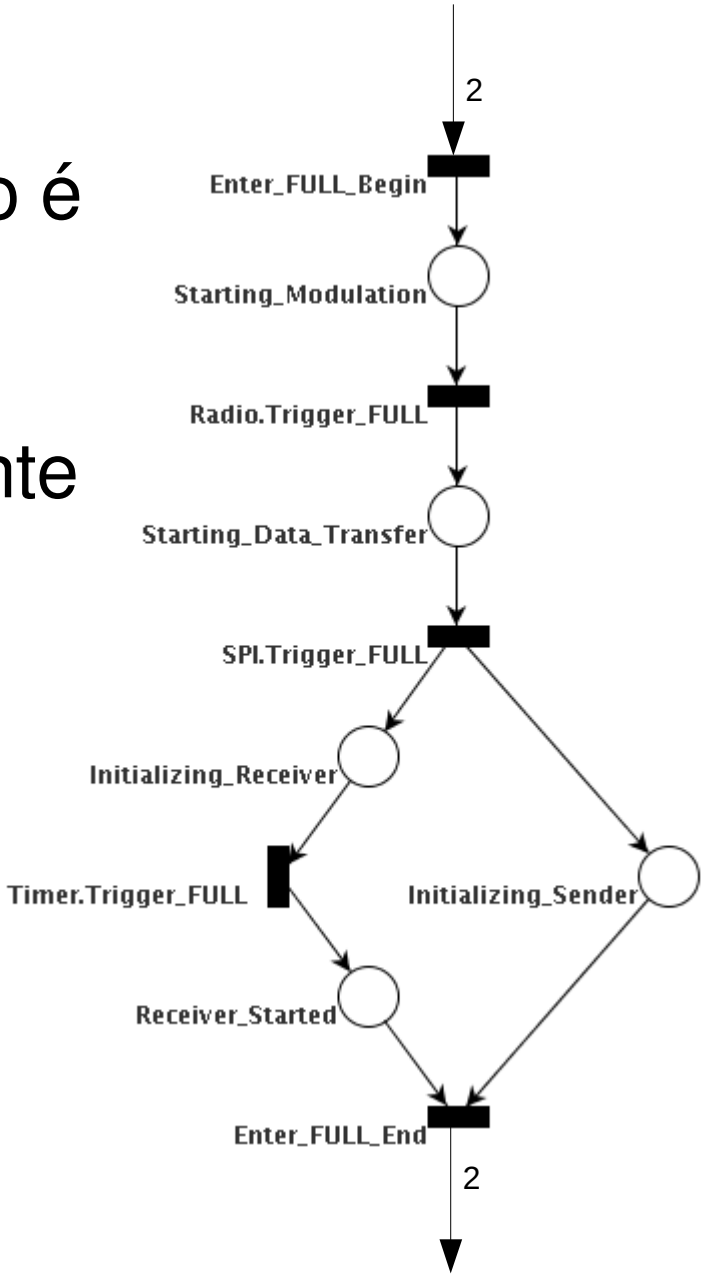




Implementação

- Redes de transição:
  - Análise em tempo de execução é desnecessária
  - Métodos de troca de modo de operação inferidos estaticamente a partir das redes

```
BMAC::power(int mode) {
  switch(mode) {
  case FULL:
    radio.power(FULL);
    spi.power(FULL);
    timer.power(FULL);
    break;
  //...
}
```

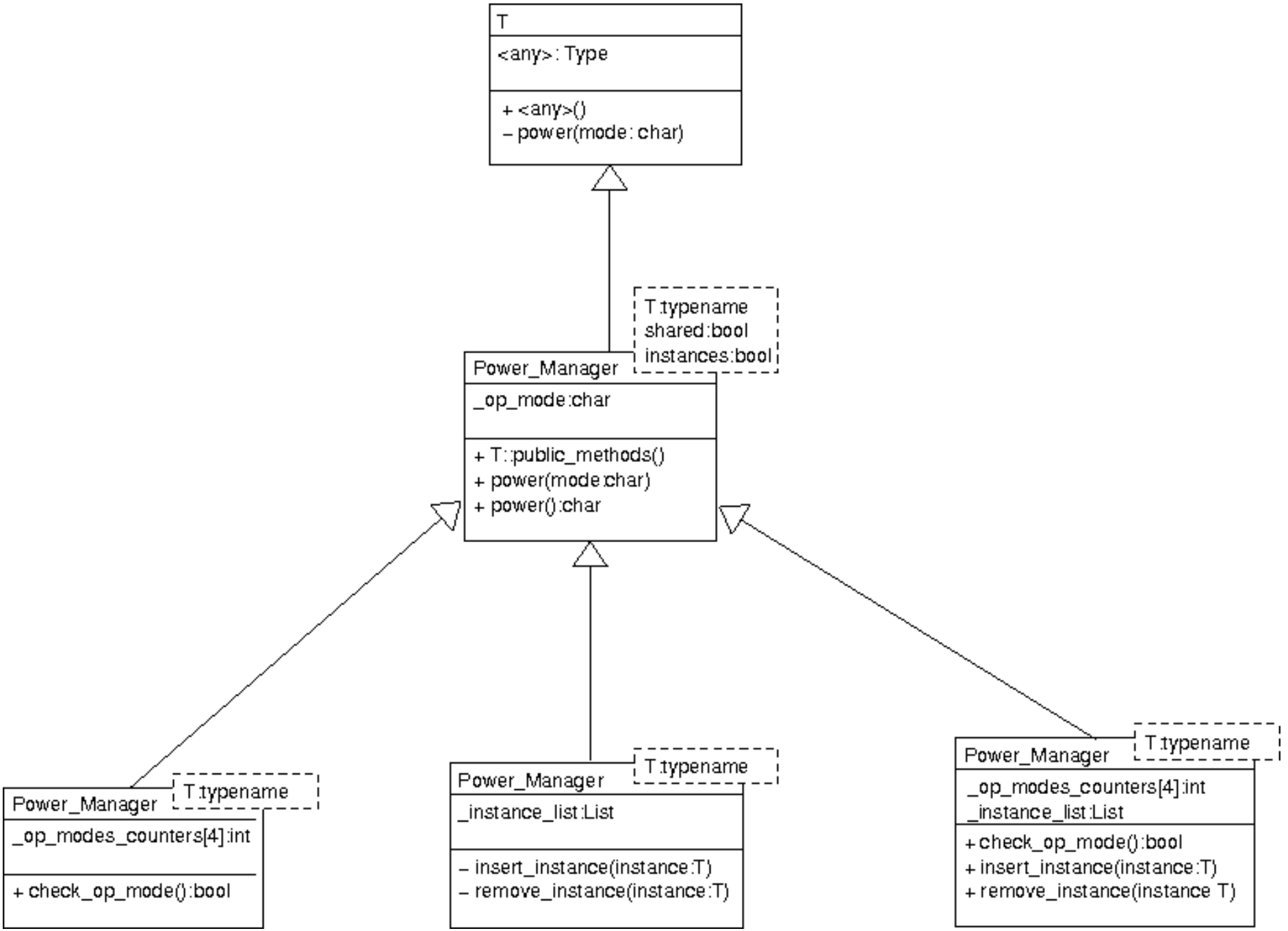


Problema: Concorrência

- Dispositivos de hardware compartilhados
  - Ex.: ADC compartilhado por vários sensores
- Solução: democracia
  - Um contador por modo de operação
  - Contadores armazenam o número de "usuários" do dispositivo
  - Maioria de instâncias requerida para migrar o dispositivo

$$P_j = \frac{(opmode[j])}{(\sum opmode)}$$

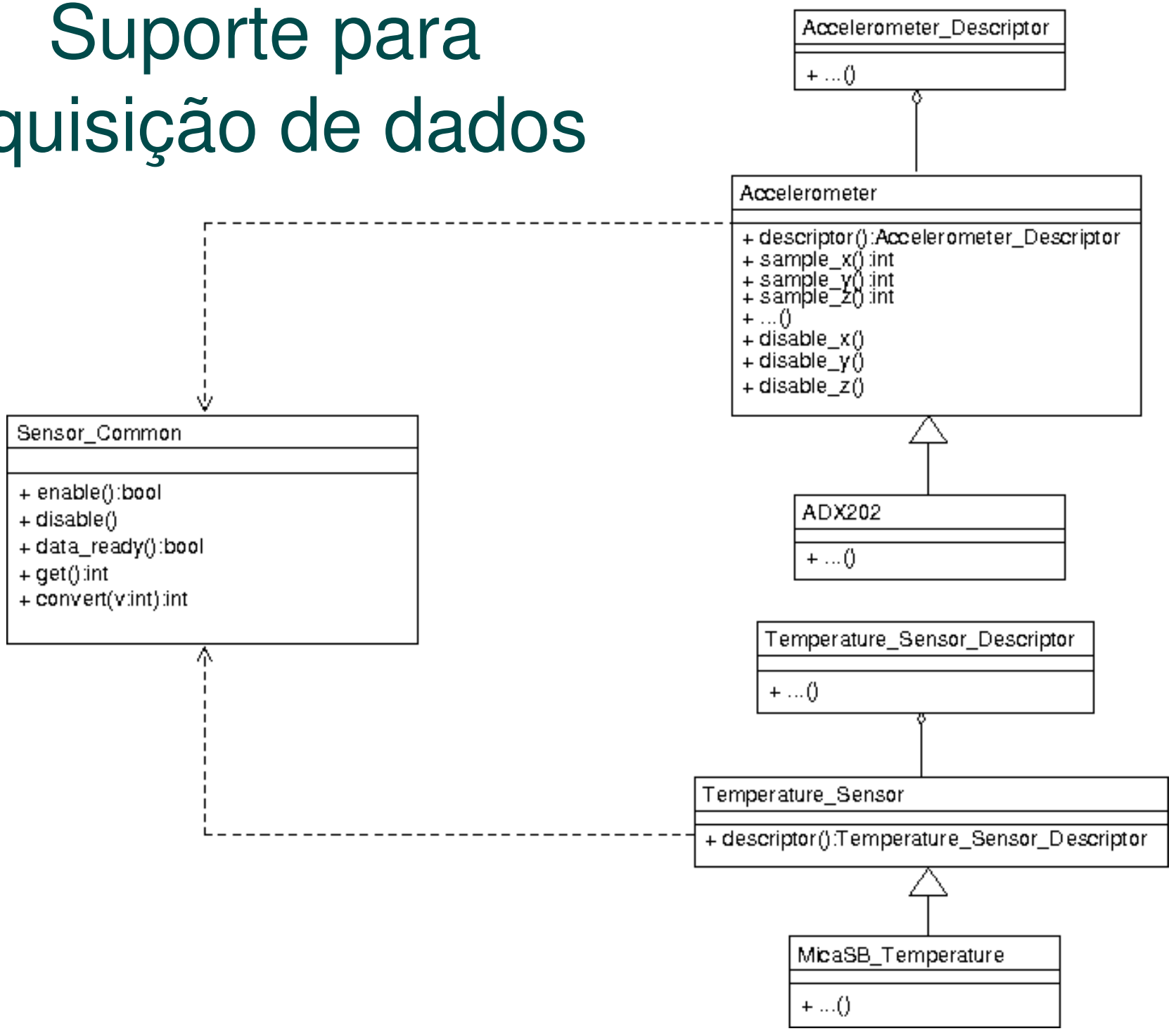
- *j* é o modo pretendido
- *opmode* é o conjunto de contadores
- Se $P_j \geq 0{,}5$ o modo de operação é entrado


T
<any>: Type
+ <any>()
– power(mode: char)
Power_Manager
T:typename
shared:bool
instances:bool
_op_mode:char
+ T::public_methods()
+ power(mode:char)
+ power():char
Power_Manager
T:typename
_op_modes_counters[4]:int
+ check_op_mode():bool
Power_Manager
T:typename
_instance_list:List
– insert_instance(instance:T)
– remove_instance(instance:T)
Power_Manager
T:typename
_op_modes_counters[4]:int
_instance_list:List
+ check_op_mode():bool
+ insert_instance(instance:T)
+ remove_instance(instance T)

Estudo de Caso: Termômetro

- Aplicação
  - Enviar uma medida de temperatura pela porta serial a cada segundo
- Hardware
  - Microcontrolador ATMega16
  - Termistor de 10 KΩ
- Estrutura de gerência de energia
  - Implementada no **EPOS**
  - Componentes utilizados: `Alarm`, `System`, `UART` e **`Temperature_Sensor`**

Suporte para aquisição de dados



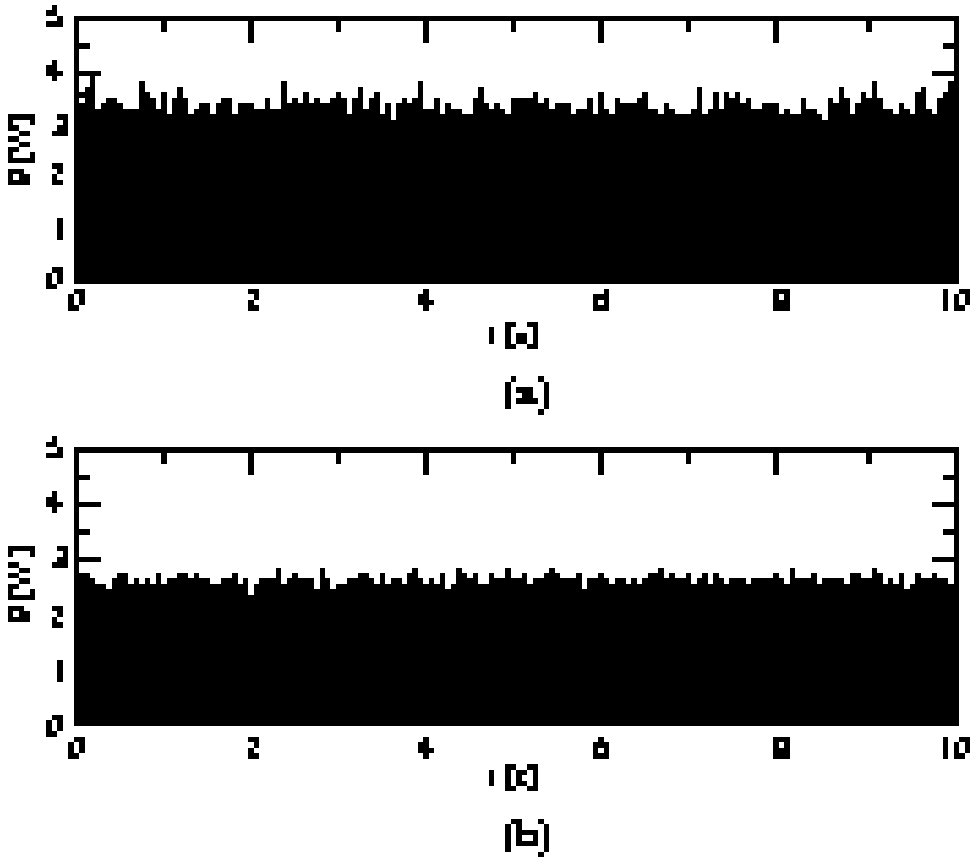
Exemplo de código-fonte

```
#include <system.h>
#include <sensor.h>
#include <uart.h>
#include <alarm.h>

using namespace System;

System sys;
Temperature_Sensor sensor;
UART uart;

void alarm_handler() {
    uart.put(sensor.sample());
}

int main() {
    Handler_Function handler(& alarm_handler);
    Alarm alarm(1000000,& handler);
    while(1) {
        sys.power(STANDBY);
    }
}
```

Medições de consumo

- Kit STK-500
- Termistor de 10 KΩ

consumo
38.1 % menor



Consumo: 3.96 J

Consumo: 2.45 J

Conclusão

- Gerência de energia dirigida pela aplicação em sistemas de sensoriamento
  - Sem comprometer memória e processamento
  - Independente de plataforma
- Configurabilidade dos modos de operação
  - Aplicações têm suas necessidades cobertas
- API uniforme
  - Programação da aplicação facilitada
  - Portabilidade garantida

Obrigado!

Arliones Stevert Hoeller Junior
Pesquisador Associado
arliones@lisha.ufsc.br
http://www.lisha.ufsc.br
+55 (48) 3331-9516 ext.14

Exemplo Complexo

```
MP3_Player player;
Display display;

//...

void main() {
  //...
  char * file_name = select_file();
  display.power(LIGHT);
  play_file(file_name);
  //...
}

//...

void play_file(char * file_name) {
  File file(file_name);
  player.load(file);
  file.power(OFF);
  player.play();
}
```

Acessos a API