Energy Efficiency for IoT Software in the Large

(Projeto Serrapilheira)

Francisco Sant'Anna

francisco@ime.uerj.br

www.ceu-lang.org

About us...

- Research areas:
  - Programming Languages
  - Distributed Systems
- Programming Language “Céu”
  - Embedded platforms
  - Wireless Sensor Networks (WSNs)
  - Games

Energy Efficiency for IoT

- 14 billion “traditional” network-connected devices (e.g., mobile phones & smart TVs).
- 50 billion by 2020 with the IoT (e.g., smart bulbs & fitness wearables).
- **Most energy consumed in standby mode.**
- Network standby is one of the six fronts on G20's Energy Efficiency Action Plan
    - https://www.iea-4e.org/projects/g20

Project Goals

1. Address energy efficiency through extensive use of standby.
2. Target constrained embedded architectures that form the IoT.
3. Provide standby mechanisms at the **programming language level** that scale to all applications.
4. Support **transparent**/non-intrusive standby mechanisms that reduce barriers of adoption.

Project Non-Goals

- Adaptive Computing
  - QoS (e.g., resolution, frame rate, accuracy)
  - Behavior (e.g., switch UI, disable functionalities)
  - *Don't take advantage of standby modes (goal 1)*
- Energy-Aware Network Protocols
  - Low-power listening
  - Small periods of duty cycles
  - *Only for networked parts, not automatic (goals 3 & 4)*
- Complex Hardware Architectures
  - Microprocessors, MMU, OS-based, Smartphones
  - *Not constrained embedded platforms (goal 2)*

Our Approach

- Enforce idle states of execution
  - Céu enforces a reactive model of execution
- Infer deepest sleeping mode
  - Céu has a semantics amenable to analysis
- Put device to sleep
  - Céu has a energy-aware runtime
- Only awake from interrupts
  - Céu provides interrupt service routines (ISRs)

A “Hello world!” in Céu

- Blinking a LED
  1. *on ↔ off every 1s*
  2. *stop after button press*
  3. *restart after 2s*

```
input  on/off PIN_02;
output on/off PIN_13;

loop do
    par/or do
        loop do
            await 1s;
            emit  PIN_13(on);
            await 1s;
            emit  PIN_13(off);
        end
    with
        await PIN_02;
    end
    await 2s;
end
```

Lines of execution
=
Trails (in Céu)

A “Hello world!” in Céu

- Blinking a LED
  1. *on ↔ off every 1s*
  2. *stop after button press*
  3. *restart after 2s*
- Vocabulary
  - Time & I/O (`await`, `emit`, etc)
- Compositions
  - seq, loop, par *(trails)*
    - At any level of depth
  - ~~state variables / communication~~

```
input  on/off PIN_02;
output on/off PIN_13;

loop do
   par/or do
      loop do
         await 1s;
         emit  PIN_13(on);
         await 1s;
         emit  PIN_13(off);
       end
   with
      await PIN_02;
   end
   await 2s;
end
```

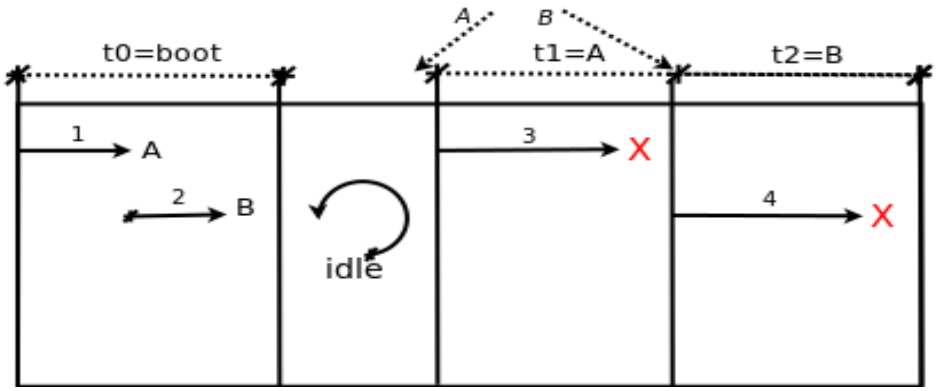
Overview of Céu

- Reactive
  - environment in control: *events*
- Imperative
  - sequences, loops, assignments
- Concurrent
  - multiple lines of execution: *trails*
- Synchronous
  - trails synchronize at each external event
  - **trails are always awaiting**
- Deterministic
  - always yields the same outcome for a given timeline

Synchronous Execution Model

1. Programs starts in the “boot reaction” in a single trail.
2. Trails execute until they await or terminate. This step is known as “reaction chain” and always executes in bounded time.
3. An input event occurrence awakes **all** trails awaiting that event. Repeat “step 2”.

```
par/and do
   <...>        // 1
   await A;
   <...>        // 3
with
   <...>        // 2
   await B;
   <...>        // 4
end
```



`<...>` *are trail segments that do not await (e.g. assignments, calls)*

Unbounded Execution

```
var int v=0;
await A;
loop do
   v = v + 1;
end
await B;
```

Is `await B` *ever reached?*

- Breaks the synchronous/reactive model

Bounded execution

- Céu ensures bounded execution
  - All loops must `await` or `break`

```
loop do
  if <cond> then
    break;
  end
end
```

```
loop do
  if <cond> then
    break;
  else
    await A;
  end
end
```

- *Limitation: heavy computations*

Project In Practice...

- Hardware infrastructure
  - Off-the-shelf Arduinos (ATMega328, Cortex-M0)
- Software infrastructure
  - Implement an energy-aware runtime for Céu
  - Rewrite device drivers in Céu (timers, SPI, Radio)
- Applications
  - Rewrite existing IoT applications in Céu
    - Time to rewrite
    - Coding “aesthetics”
    - Energy consumption

An example

```
// application.ino

void loop () {
    delay(3600000);
    int v = analogRead();
    <performs-some-action>
}
```

```
// wiring_analog.c

int analogRead () {
    <...>
    while (bit_is_set(<...>)) {}
    return <...>;
}
```

```
// application.ceu

output none ADC_REQUEST;
input  int  ADC_DONE;
#include "adc.ceu" // driver implementation

every 1h do
    emit ADC_REQUEST;
    var int v = await ADC_DONE;
    <performs-some-action>;
end

.
```

[a] Application: no explicit energy management

Research Topics

- Language level
  - Primitive support for interrupt service routines
  - Energy-aware runtime system
- HAL level
  - Event-driven syscall API
  - Driver infrastructure
  - Synchronous/Asynchronous interaction
- Application level
  - IoT applications
  - Evaluation (e.g., energy consumption)

Current Status

- SenSys'13, ACM Embedded Networked Sensor Systems
  - *Safe System-level Concurrency on Resource-Constrained Nodes*
- TECS'17, ACM Transactions on Embedded Computing Systems
  - *The Design and Implementation of the Synchronous Language Céu*
- GSoC'17, Google Summer of Code 2017
  - *Interrupt-based drivers and libraries for Céu-Arduino*
- Drivers
  - Arduino UNO (ATMega328): Timer/Analog/Serial/SPI
  - Arduino Zero (ARM Cortex M0+): Timer
- Energy-Aware Runtime
  - Basic Sleep Mode

Financiamento Serrapilheira

- 60/2000 projetos receberam até R$100k por 1 ano
- 12/60 projetos receberão até R$1M por mais 3 anos

- Nosso projeto é de 70k
  - 10k para viagens e equipamentos
  - 60k para bolsas
    - 1-2 alunos de graduação
    - 1-2 alunos de mestrado

Energy Efficiency for IoT Software in the Large

(Projeto Serrapilheira)

Francisco Sant'Anna

francisco@ime.uerj.br

www.ceu-lang.org