Research Progress

We start with an abstract of our project roughly based on the original proposal since the core research idea did not change. Then, we then discuss the progress of each specific goal enumerated in the original proposal and summarize what we achieved and still expect to achieve. We conclude with the main achievements of the first year and discuss the problems we encountered and how our overall approach to conduct the research evolved during this period.

Abstract

Effective use of low-power standby will play a fundamental role in energy efficiency for the expected 50 billion IoT devices by 2020. This research project aims to address the software challenges, as observed by the International Energy Agency, towards an energy-efficient IoT: to ensure that devices employ the lowest possible modes of standby, and that devices remain in longest possible periods of standby.

Given the projected scale of the IoT and the role of power standby controlled by software towards energy efficiency, this research project has the following goals:

- Address energy efficiency through extensive use of standby in software.
- Target constrained embedded microcontroller architectures that form the IoT.
- Provide standby mechanisms at the programming language level that scale to all applications.
- Support transparent/non-intrusive standby software mechanisms that reduce barriers of development.

The bulk of our research is based on the design of Céu, a new reactive programming language, which we have been working for the past 9 years. The language is grounded on the synchronous concurrency model, which trades power for reliability and has a simpler model of time that suits most requirements of IoT applications. In this model, all reactions to the external world are guaranteed to be computed in bounded time, ensuring that applications always reach an idle state amenable to standby mode.

Specific Goals - Original Proposal

In the original proposal, we enumerated three specific goals with associated risks and expected contributions:

Goal 1 - IoT Hardware Infrastructure

- **Risks:** very low (1/5)

- **Expected contributions:** (minor scientific contributions)
- **Progress:** 75%

We need an energy-efficient IoT hardware infrastructure that allows our software infrastructure to control it and take advantage of low-power modes of operation.

We adopted third-party Arduino hardware platforms with builtin low power support and did not expect much risks associated with this choice. We also selected IoT peripherals (e.g., radio transceivers and real-time clocks) that can operate in sleep modes to consume less energy.

Using this infrastructure, the automatic software mechanisms from our proposal can now control the hardware to reach almost 100% energy efficient without compromising the functionalities of the applications.

We want even more control over the hardware components and intend to prototype a customizable and affordable low-power development board as a side project. Last year, we submitted this idea to the main Brazilian scientific funding agency (*CNPq Universal call*), but we did not receive the grant. The proposal is attached as the file "cnpq-universal-18.pdf".

Goal 2 - IoT Software Infrastructure

- **Risks:** high (4/5)
- **Expected contributions:**
  - An energy-aware programming language that integrates with the surrounding physical environment.
- **Progress:** 60%

The core idea of our project is on the design of Céu, a new programming language that uses standby effectively:

- A programmer writes **any** IoT application in Céu unaware of standby or energy constraints.
- The compiler of Céu generates a program that puts the IoT device to sleep automatically and in the most efficient standby mode for as long as possible.

Goal 2.1 - Design and Implementation

We designed Céu from the very beginning to be promptly reactive to the surrounding environment. Basically, whenever a stimulus occurs (e.g., a sensor detects a change, or a radio message arrives), the program must be ready to react to that input. Céu guarantees that reactions always execute in finite time so that upcoming inputs are also readily handled.

A key insight is that if the period between subsequent stimuli is much higher than the associated reaction times, then the application can be in standby most

of the time. As an example, if a radio message arrives every second and the time to process each message is in the order of a few milliseconds, then the device can sleep almost 100% of the time. Follows the general approach we employed to address this insight:

- Enforce idle states of execution after each reaction
- Put the device in standby mode after each reaction
- Only awake from hardware interrupts on external stimuli

We wrote two work-in-progress papers on this approach, which we presented in June at LCTES'18 (Languages, Compilers, and Tools for Embedded Systems) and in November at REBLS'18 (Reactive and Event-Based Languages and Systems).

In the first paper, which targets embedded system researchers, we describe the design at a higher level, first discussing why current languages cannot provide automatic standby for applications, and then why and how we can extend Céu to understand which level of standby to adopt after each reaction and how to awake only from hardware interrupts.

In the second paper, which targets reactive language designers, we described the technical details of interrupt service routines in Céu. For instance, we detailed how programmers can map an input from the environment (e.g., a button press) to a reaction in the program. We also discussed the safety threads due to concurrency and how Céu can detect race conditions at compile time in order to prevent them.

The papers are attached as the files "lctes-wip-18.pdf" and "rebls-wip-18.pdf".

Goal 2.2 - Experimentation and Evaluation

Goal 2.3 - Formal Proofs

- Anny
  - education
- Joao
  - games
- Anna
  - IoT / energy

not originally

rewrite from scratch use automatic proofs - more reliable - extensible

The main challenge of our project is to design a programming language that uses standby effectively for all programs.

- prove?

- Provide standby mechanisms at the programming language level that scale to all applications.
- Support transparent/non-intrusive standby software mechanisms that reduce barriers of development.

  CEU's dedicated vocabulary to deal with events raises the programming abstraction to a level closer to the domain of IoT, providing more safety and expressiveness to programmers. These results have already been partially published in the past~[**?**], when we implemented a radio device driver on top of TinyOS. Nevertheless, the novel support for ISRs would complete the whole development cycle, from the application down to the hardware, without operating system support. \item **An energy-aware system language:** The method described in this section makes all applications subject to standby modes transparently. Only device drivers, which are typically provided by vendors, will require explicit energy management. In our case, we will need to rewrite each driver only once, as part of the software infrastructure, and all applications built on top of them would benefit from energy efficient automatically. %\item[TODO: education] \end{enumerate}

Specific Goals - Actual Plan

energy-efficient.

For instance, we could reach

We proposed three fronts

- Research progress report (short paragraph for each aim)
- Original aims
- Current aims
- no trained
- more clear what is expected

Risks (very low: 1/5)

Expected Contributions

IoT Software Infrastructure

Team and Partners

- What individuals have worked on the project and what other organizations have been involved as partners

Carrier Impact

- Carrier Impact/Comments/Concerns (short paragraph)
- Life-time project
- PEL

Next Steps

- The next steps for the following three years in your project
- syntax and vocabulary
- semantics
- only myself

  Nevertheless, the novel support for ISRs would complete the whole development cycle, from the application down to the hardware, without operating system support.
- All software uses progr. languages
- Turing awards
- Big enterprises (Images)
  - Microsoft: VB, C#, ...
  - Google: Go
  - Apple: Swift
  - Mozilla: Rust
  - Sun/Oracle: Java
  - none of them are looking at energy