Francisco Sant'Anna

Universidade do Estado do Rio de Janeiro - UERJ

Energy Efficiency for IoT Software in the Large

Research Progress

Abstract

Effective use of low-power standby will play a fundamental role in the energy efficiency of the expected 50 billion IoT devices by 2020, as observed by the International Energy Agency. Given the projected scale of the IoT and the role of standby towards energy efficiency, this research project has the following goals:

- Addressing energy efficiency through extensive use of standby in software.
- Targeting constrained embedded microcontrollers that form the IoT.
- **Providing transparent standby mechanisms at the programming language level in order to scale to all applications.**

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

In the original proposal, we enumerated three specific goals with associated risks and expected contributions. In the mid-term report, we included two new goals. We now also include the estimated efforts (from the total workforce) and completeness of each goal, to give a perception of our priorities and overall progress.

Goal 1 - IoT Hardware Infrastructure

- **Risks:** very low (1/5)
- **Expected contribution:** A low-power IoT hardware platform.
- **Efforts:** 5%
- **Completeness:** 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 and potentially reach over 99% 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 did not receive the grant. The proposal is annexed as the file "cnpq-universal-18.pdf".

Goal 2 - IoT Software Infrastructure

- **Risks:** high (4/5)
- **Expected contribution:** An energy-aware programming language that integrates with the surrounding physical environment.
- **Efforts:** 45%
- **Completeness:** 80%

The main challenge we confronted this year was on extending the design of the programming language Céu to use 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 in the most efficient standby mode for as long as possible.

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 their 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 materialize this insight:

- Enforcing that the program reaches an idle state after each reaction.
- Putting the device in standby mode whenever the program is idle.
- Awaking the program only from hardware interrupts on external stimuli.

Céu provides dedicated vocabulary to deal with external events, e.g., a statement like "`await 1s`" makes the program to go idle for 1 second. Since the language compiler understands this vocabulary, it can now apply standby automatically. With the support of device drivers, which are also written in Céu, the runtime chooses the optimal standby mode based on the events the program is currently awaiting. For the "`await 1s`" example, the runtime will choose a standby mode in which the clock remains active, since the hardware still has to count time somehow. The language also guarantees that after it awakes, the program must reach another await statement in finite time in order to sleep again. It does so by refusing non-awaiting infinite loops at compile time. Céu also supports logical parallelism to handle multiple events concurrently. In this case, the runtime chooses the best standby mode that contemplates and awakes from all events.

We described our approach for automatic standby in two work-in-progress papers, 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). The papers are annexed as the files "lctes-wip-18.pdf" and "rebls-wip-18.pdf".

The first paper, which targets embedded system researchers, describes our 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.

The second paper, which targets reactive language designers, describes 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.

We consider that we reached a satisfactory design regarding energy efficiency, which was our main goal for the first year. However, we still need to polish the implementation of Céu to make it ready for a comprehensive quantitative evaluation to be described next.

Goal 3 - IoT Applications

- **Risks:** medium (3/5)
- **Expected contribution:** A realistic energy-efficient IoT programming alternative to Arduino.
- **Efforts:** 15%
- **Completeness:** 30%

In order to evaluate the gains in energy efficiency with the proposed software infrastructure, we need to evaluate the consumption of realistic applications.

In the first year, we focused on basic support for radio communication, which plays a fundamental role in the IoT: we developed SPI and RF drivers on top of our new infrastructure and also a basic ad-hoc communication protocol. Then, we measured the energy consumption of a number of applications to compare our system with the original Arduino:

App	Arduino	Céu	Description
Empty	3.7	0.002	empty program
Blink	6.0	3.1	periodic LED blinking
Sensor	11.4	7.7	sensors reads
Radio	19.5	15.8/3.0	periodic RF communication
Protocol	19.6	15.9	active mesh RF protocol

We did not advance much on these experiments after the mid-term report, so the numbers above are the same. The first case illustrates how an empty program in Arduino remains awake 100% of the time, while the equivalent in Céu sleeps in the most efficient standby mode. Even though an empty program is useless, it shows how far we can go with automatic standby, serving as a basis of comparison. The blinking example in Céu relies on timer standby, which is not the most efficient, but still consumes less energy than its Arduino counterpart. The radio example transmits a message periodically, and Céu is able to turn the radio off automatically, showing significant savings while in the idle state (3.0 mA). The protocol example illustrates a case in which the language cannot help: if the specification states that the radio must be on on all the time, then the language cannot apply standby automatically, since this would change the behavior of the application.

As we already mentioned on the mid-term report, the results are preliminary. We still did not evaluate complex applications, which will demand substantial work until we consider a full publication comprising our three original goals.

Goal 4 - Training and Education

- **Risks:** low (2/5)
- **Expected contribution:** A framework for early computer science education centered around the IoT.
- **Efforts:** 10%
- **Completeness:** 20%

One of the difficulties we encountered early in the project was to hire students with basic skills on embedded systems and IoT. Since we are proposing a new programming approach for the IoT, this difficulty is even more noticeable.

In May, one of our students got a 3-month scholarship from the "Google Summer of Code" initiative and started to develop the new tool "Céu-Maker" targeting students with no programming background. In October, we submitted a project to encourage more girls to pursue STEM careers (*CNPq - Girls on Sciences, Engineering, and Computing*). The project was recommended but we did not receive the grant. The proposal is annexed as the file "cnpq-meninas-18.pdf". As a follow up, we plan to offer a 3-week course open for early undergraduate students on each semester, starting this year.

In September, we introduced a new course on embedded systems in our University targeting graduate and late undergraduate students. The goal is to expose more technical aspects related to our research and instigate students seeking for dissertation topics.

Goal 5 - Formal Verification

- **Risks:** very high (5/5)
- **Expected contribution:** An automated formal verification that reactions in Céu are deterministic, memory bounded, and always terminate.
- **Efforts:** 25%
- **Completeness:** 30%

Céu guarantees by design that a program always reaches an idle state amenable to standby after each reaction. In order to verify that our design is correct, we formalized the language specification using a small-step operational semantics. We then formalized the notion of termination and proved, using standard induction techniques, that reaction times are indeed finite. We also proved that the language is deterministic and uses finite memory. These results were presented in June at LCTES'18 (Languages, Compilers, and Tools for Embedded Systems) and were expanded in a journal paper to be published soon on JSA (Journal of Systems Architecture). The papers are annexed as the files "lctes-full-18.pdf" and "jsa-19.pdf".

These are important but strictly theoretical results, since they apply to a mathematical model of Céu, but not to its current implementation. We are now working on a verifiable interpreter of Céu in Haskell and LiquidHaskell. LiquidHaskell uses an SMT (Satisfiability Modulo Theories) solver to automatically prove statements about programs. We already managed to prove that a simplified version of our interpreter behaves deterministically and always terminates.

Fully automated theorem proofs are more reliable and extensible in comparison to the "paper-and-pencil" approach we adopted earlier. However, they are also much more laborious and rely on unpredictable exponential-time decision algorithms.

Team

- Francisco Sant'Anna
  - PhD, professor at UERJ
  - 12 months (40h/week)
  - Goals 2, 3 & 5

Francisco is the Serrapilheira grantee responsible for the project. During the first semester, he focused on the core idea of the research (Goal 2), i.e., on the design and implementation of energy-aware mechanisms for Céu. He also wrote the first IoT applications to take advantage of automatic standby (Goal 3). During the second semester, he focused on the mathematical foundations of the language (Goal 5), specifying the formal semantics, sketching the proofs, and rewriting the language implementation from scratch towards automated proofs.

- Guilherme Lima
  - PhD, self employed
  - 6 months (20h/week)
  - Goal 5

Guilherme worked part time in the second semester on the foundations of Céu (Goal 5). He worked on the details of the proofs and also did the first explorations with LiquidHaskell. He co-authored two of the papers we published during the year.

- Anny Caroline Chagas
  - MSc student at UERJ
  - 8 months (20h/week)
  - Goals 3 & 4

At the end of the first semester, Anny received an external scholarship (from "Google Summer of Code") to work on "Céu-Maker", an educational tool for Céu (Goal 4). In the second semester, she joined our MSc program in Electronic Engineering at UERJ. She worked on the design of new a entry-level programming course focused on IoT and wrote an educational project for encouraging girls in STEM careers (Goal 4). She is now developing complex IoT applications in Céu to measure the energy efficiency of our proposed solution (Goal 3).

- Anna Leticia Alegria
  - Undergraduate student at PUC-Rio
  - 7 months (15h/week)
  - Goal 3

Anna worked on the initial experiments with IoT applications in Céu (Goal 3). She tested the early prototypes of the new version of Céu with support for automatic standby. She also investigated existing applications in C/Arduino and rewrote them in Céu. This allowed us to compare the energy consumption of the same applications written in two different languages.

- João Sampaio
    - Undergraduate student at UERJ
    - 6 months (20h/week)
    - Goal 3

João explored the development of video games in Céu, which are considerably more complex than typical IoT applications (Goal 3). Games are demanding in terms of computational resources and programming expressiveness, serving as a stress test for the robustness of Céu beyond the domain of IoT.

Career Impact

The Serrapilheira grant was a key condition for me to get accepted on the graduate school of my University last year. Being part of a graduate school opens new opportunities for collaborations with students and colleagues. In the second semester, I started to advise a master student which is already reaching a level of maturity to contribute effectively this year.

The grant also allowed me to take two international trips to attend the top conferences in my field of programming languages (PLDI & SPLASH). It is important to notice that for the international programming languages community, top conferences have much more prestige than journals, an aspect that is neglected by the Brazilian "one-size-fits-all" evaluation policy for researchers. In the two conferences, I was able to present three papers on satellite workshops. I was also the program chair of the workshop on "Reactive and Event-Based Languages and Systems", which resulted in an invitation to become part of its permanent steering committee.

Next Steps

The immediate next step is to complete the evaluation of energy consumption for realistic applications (Goal 3). We plan to take existing complex IoT applications and rewrite them in Céu to measure the difference in consumption between the two implementations. If our initial experiments are confirmed for realistic applications, then we will complete our three original goals: (i) a low-power hardware architecture, (ii) controlled by automatic standby, and (iii) validated in real-world scenarios. Most of the hard work can be done by students and we believe that by the end of the year we will have a handful amount of applications to produce solid results.

In the last semester, we shifted our focus towards more fundamental aspects of programming languages and started a new implementation of Céu from scratch with verification in mind (Goal 5). We will probably need this whole year to reach the same functionality of the current implementation (Goal 2) in order to start using it exclusively in new applications. The other front is to take the new implementation and make use of LiquidHaskell to automatically verify the properties of termination, determinism, and memory boundedness. This would be a result of a greater impact, since extensive automatic verification of properties that hold for all programs is a valuable asset. We already proved termination for a small subset of the language, which gave us confidence to continue on this direction. We will probably need over a year to cover a significant portion of the language, since it is estimated that for each line of the interpreter, we need two other lines of annotations for verification.

Besides constrained embedded systems, the IoT also consists of traditional networked devices, such as routers, servers, and smartphones. These devices are based on much more complex architectures (e.g., 64-bit CPUs with memory management unit) and rely on operating systems to orchestrate multiple applications. A long term goal is to investigate an alternative to the bloated and antiquated operating systems like Windows and UNIX, which were not designed with mobile and energy awareness in mind. The concept and vocabulary of events designed for Céu can be extended to deal with inter-process communication, serving as the foundations of a new operating system.