

# Título

Eficiência Energética e “Standby” como Princípios de uma Linguagem de Programação para a IoT

## Resumo

Estima-se que o consumo anual de energia de dispositivos conectados em todo o mundo seja superior ao consumo total de energia da Alemanha. No entanto, a maior parte dessa energia é gasta em modo “standby” (modo de espera), ou seja, quando o dispositivo não está desempenhando sua finalidade. A Agência Internacional de Energia (IEA) prevê que o uso efetivo de standby terá papel fundamental na eficiência energética dos 25 bilhões de dispositivos da Internet das Coisas (IoT) esperados até 2025.

Este projeto de pesquisa visa endereçar os desafios de energia para o software de dispositivos conectados conforme determinados pela IEA: garantir que os dispositivos adotem níveis de standby profundos, e que permaneçam em standby o maior tempo possível.

Propomos investigar a eficiência energética de software como princípio da linguagem de programação Céu, de modo que todos os programas nela desenvolvidos se beneficiem de longos e profundos períodos de standby automaticamente, sem esforços extras de programação. O foco principal de Céu são aplicações reativas que interagem com o ambiente em ciclos contínuos de espera, leitura de sensores e atuação, típicos de sistemas embarcados e IoT. A linguagem é baseada no modelo de concorrência síncrono, que troca poder por confiabilidade e possui um modelo de tempo mais robusto e restrito mas que cobre os requisitos principais de aplicações reativas. Nesse modelo, todas as reações ao mundo externo são computadas em tempo finito, garantindo que as aplicações sempre cheguem a um estado ocioso suscetível ao modo standby.

## Estado da Arte

Apesar dos avanços de pesquisa em linguagens de programação, sistemas embarcados ainda são escritos praticamente somente em C [3]. A predominância de C está associada a sua portabilidade entre arquiteturas, sua eficiência em termos de uso de memória e CPU, e também ao seu legado de código e programadores. Sendo assim, o desenvolvimento completo da pilha de IoT ainda depende muito de C, desde as aplicações de mais alto nível, passando por protocolos de redes, até sistemas operacionais, drivers e SoC firmwares [3]. No entanto, C oferece uma simples abstração de hardware (um assembly portátil) e nenhuma ciência sobre o ambiente externo sob o qual as aplicações executam. Como exemplo, C

não oferece um vocabulário dedicado a expressar conceitos que naturalmente aparecem em aplicações de IoT, tais como o tempo, comunicação com sensores, concorrência de eventos e ciência de energia [3]. Além disso, C também é conhecida como uma linguagem insegura sob o ponto de vista de acesso à memória, sendo uma fonte de bugs característicos, tais como vazamento de memória, estouro de buffer e ponteiros pendentes.

Existem diversas propostas de pesquisa para linguagens e sistemas operacionais conscientes de energia [2]. Algumas propostas ajustam a qualidade de serviço (QoS) para reduzir o consumo de energia (ex., reduzindo a acurácia de computações). Outras propostas oferecem mecanismos para ajustar o comportamento das aplicações para atender às demandas dos níveis de bateria (ex., desabilitando funcionalidades). Essas iniciativas não se concentram em tirar proveito do modo de standby, mas sim em adaptar ou eliminar computações no modo ativo. Também existem protocolos de rede especializados em deixar os dispositivos em standby por longos períodos. No entanto, iniciativas baseadas em protocolos se aplicam somente às partes de rede das aplicações que ainda assim devem ser programadas cuidadosamente para tirar proveito de standby.

O TinyOS [3] é um sistema operacional para redes de sensores sem fio que usa um dialeto de C e atende parcialmente aos nossos objetivos. Aplicações escritas nesse dialeto podem tirar proveito de standby automaticamente, mas ainda dependem de um controle manual do ciclo de vida dos drivers (ex., para ligar ou desligar dispositivos). Além disso, sendo um sistema operacional completo, a sua base de código é complexa e escrita inteiramente em C. Por fim, para lidar com concorrência entre eventos, o TinyOS impõe o paradigma de programação por callbacks que é conhecido como difícil e mais suscetível a erros. Considerando essas limitações, temos como objetivo prover suporte a economia de energia automática já no nível da linguagem e de maneira completa, além de oferecer um paradigma de programação estruturado mesmo na presença de concorrência entre eventos [3].

## Trabalhos Anteriores

Tenho trabalhado no projeto e implementação da linguagem de programação Céu pelos últimos 10 anos [5]. Céu é uma nova linguagem reativa que tem como foco principal sistemas embarcados restritos. A linguagem oferece suporte primitivo a concorrência determinística (garantido comportamento reproduzível) e ainda detecção de acessos simultâneos a variáveis. Céu é baseada no modelo de concorrência síncrono, que troca poder por confiabilidade e possui um modelo de tempo mais robusto e restrito mas que cobre os requisitos principais de aplicações IoT. Nesse modelo, todas as reações ao mundo externo são computadas em tempo finito, garantindo que as aplicações sempre cheguem a um estado ocioso suscetível ao modo standby.

Em trabalhos anteriores [5], adaptamos Céu para executar sobre o TinyOS no contexto de redes de sensores sem fio e desenvolvemos algumas aplicações, protocolos e device drivers. Nós avaliamos a expressividade de Céu em comparação com C e verificamos uma redução de código na ordem de 25%, com um pequeno aumento de memória em torno de 5%. Também avaliamos a responsividade da CPU sob tráfego de dados intenso e verificamos uma performance similar entre as duas linguagens. Em outro trabalho [5], desenvolvemos uma máquina virtual minúscula de Céu para dispositivos de baixo consumo energético que permite reprogramação remota. Por fim, adotamos Céu em outras classes de sistemas reativos com sucesso, tais como jogos e multimídia [5].

Em um trabalho mais recente [5], discutimos o design de Céu sob a perspectiva de semântica de linguagens de programação e em comparação com o trabalho seminal de Esterel, a primeira linguagem síncrona. A diferença mais fundamental entre as duas linguagens é que Céu usa uma noção de tempo baseada na ocorrência de eventos externos que definem instantes indivisíveis na linha de tempo lógica das aplicações. Como consequência, semanticamente o tempo não avança durante períodos inativos (quando não há ocorrência de eventos), fazendo com que a implementação de todas as aplicações seja suscetível ao modo standby, fato que iremos explorar neste projeto.

## Objetivos

Para confrontar o desafio de um uso efetivo e difundido de standby, novas soluções devem ser escaláveis para a massa software IoT que está por vir. Este projeto de pesquisa visa endereçar os desafios energéticos de software, conforme determinados pela IEA [1]:

- Garantir que os dispositivos atinjam níveis profundos de standby.
- Garantir que os dispositivos permaneçam longos períodos em standby.

Tendo em vista a escala projetada para a IoT e o papel do modo standby para a eficiência energética, este projeto de pesquisa tem os seguintes objetivos específicos:

1. Endereçar a eficiência energética com o uso criterioso de standby.
2. Focar em arquiteturas embarcadas restritas que formam a IoT.
3. Prover mecanismos de standby no nível da linguagem Céu possibilitando escalar para todas as aplicações.
4. Suportar mecanismos de standby transparentes e não intrusivos para reduzir as barreiras de adoção por programadores.

Essa proposta se situa na camada mais baixa de desenvolvimento de software — na camada de linguagens de programação. Assim, todas as aplicações escritas na linguagem Céu tirarão vantagem do modo standby automaticamente, sem esforços extras de programação.

Esperamos que ao reescrever aplicações existentes, estas poderão se beneficiar de economias da ordem de 50%, baseado em estimativas da IEA e também de trabalhos em ciência transparente de energia [2].

## Metodologia

### 24 Meses Iniciais

#### Infraestrutura de Hardware para IoT

Usaremos o Arduino como a principal plataforma de hardware para IoT [4]. A maioria dos modelos é baseada em microcontroladores de baixo consumo de energia, tais como o ATmega328p que suporta seis modos de standby. Dependendo das configurações (ex., frequência e voltagem da MCU), um Arduino pode drenar de 45mA em operação máxima até 5uA no nível mais profundo de standby. A literatura mostra que é possível fazer com que aplicações IoT operem com apenas 50% de “duty cycle” em média. Assim, considerando que o consumo em standby é desprezível, poderemos economizar até 50% de energia.

No ambiente acadêmico, o Arduino já é adotado em muitas pesquisas no contexto de IoT [4]. Portanto, a popularidade do Arduino fará com que a nossa pesquisa seja mais acessível e reproduzível para outros grupos. Em educação, muitos cursos em universidades usam o Arduino [4]. Nós também usamos o Arduino em cursos de graduação e pós-graduação nos últimos 5 anos, o que nos permite (e permitirá) avaliar o uso de Céu com programadores de sistemas embarcados menos experientes.

#### Infraestrutura de Software para IoT

Em sistemas Arduino (e embarcados em geral), a maneira mais comum de interagir com o mundo externo é através da técnica de “polling”, que faz amostragens periódicas em periféricos externos para detectar mudanças de estado. A técnica de polling gasta ciclos de CPU e previne que o dispositivo entre em modo standby. No Arduino, mesmo funcionalidades básicas, tais como temporizadores, conversores A/D, e SPI, usam ciclos de polling que gastam energia ininterruptamente em modo ativo.

De modo a prover standby automático, as aplicações devem ser inteiramente reativas a eventos e precisaremos reescrever toda a infraestrutura de software em Céu. Esse processo consiste primordialmente em reescrever “device drivers”, que são os pedaços de software que interagem diretamente com o hardware. Mais concretamente, no nível mais básico, a eficiência energética depende das rotinas de interrupção (ISRs), que acordam o microcontrolador do modo standby quando ocorrem eventos em periféricos externos.

Em um projeto anterior, demos os primeiros passos na direção de uma infraestrutura de software em Céu. Já adicionamos suporte a rotinas de interrupção (ISRs) como um conceito primitivo de Céu, o que permite reconstruir a infraestrutura com ciência ao modo standby desde sua base. Essa abordagem não irá afetar a maneira como as aplicações são escritas em níveis mais abstratos, que permanecerão similares às aplicações em Arduino. No entanto, em vez de gastar ciclos da CPU em espera ativa, as aplicações entrarão no modo de standby mais profundo possível enquanto estiverem ociosas.

O código a seguir é um esboço da abordagem que iremos adotar. A aplicação solicita, a cada hora, a leitura de um conversor analógico digital e aguarda o seu retorno para executar alguma ação:

```
output none ADC_REQUEST // evento de saída para iniciar a conversao
input  int  ADC_DONE    // evento de entrada que sinaliza o resultado

#include "adc.ceu"      // driver que implementa os eventos de E/S

every 1h do             // a cada hora
    emit ADC_REQUEST    // solicita uma conversao
    var v = await ADC_DONE // aguarda o seu termino e le o resultado
    <...>                 // executa alguma acao
end
```

É possível notar que o código é escrito de maneira estruturada (sem callbacks) e sem qualquer referência explícita a modos de standby. As aplicações usam nomes para abstrair os eventos de entrada e saída que são implementados em drivers. O comando de espera “await” indica os pontos em que a aplicação poderá entrar em standby e é a única maneira de interagir com os sensores. A maior parte do trabalho fica a cargo do driver, que será escrito apenas uma vez e poderá ser reusado em todas as aplicações:

```
output none ADC_REQUEST do // implementacao do evento de saida
    <configura-ADC>
    <habilita-interruptoes-ADC>
    <informa-modo-standby>
end

input int ADC_DONE do      // implementacao do evento de entrada
    <desabilita-interruptoes-ADC>
    return <le-dados-ADC>
end
```

O evento de saída (bloco “output”) configura o periférico para fazer a requisição, habilita as interrupções correspondentes e informa à linguagem qual é o seu modo de standby mais profundo mas que ainda permita que o periférico acorde o microcontrolador. A linguagem será responsável por interagir com os drivers e identificar o maior denominador comum de standby entre todos os dispositivos

em uso. A cada momento ocioso, a linguagem acionará o modo de standby ótimo identificado, desligando temporariamente a CPU. O evento de entrada (bloco “input”) é acionado pelo dispositivo correspondente através de uma interrupção, acordando a CPU e retomando a execução da aplicação automaticamente. O código desabilita futuras interrupções do periférico e retorna a leitura requisitada para a aplicação final.

Com essa abordagem, o código da aplicação permanecerá similar aos seus equivalentes em Arduino. No entanto, em vez de gastar ciclos da CPU com polling, as aplicações irão entrar em standby sempre que estiverem ociosas (aguardando em um “await”). Essa abordagem já foi validada em aplicações e drivers muito simples, mas ainda não realizamos estudos completos por se tratar de projeto em estágio preliminar.

## Aplicações IoT

De modo a avaliar os ganhos de energia com a infraestrutura proposta, precisaremos medir o consumo de aplicações realísticas. A comunidade do Arduino tem grande disponibilidade de projetos open-source que poderão ser reescritos em Céu para tirar proveito do modo de standby automaticamente. Então poderemos comparar o consumo de energia entre as versões originais e reescritas para tirar conclusões sobre a efetividade do standby transparente. Os cenários mais realísticos de IoT usam comunicação por rádio extensivamente. Nesse contexto, iremos avaliar desde protocolos ad-hoc simples até protocolos mais complexos com ciência energética para ver até que extensão a nossa proposta contribuirá efetivamente para economias de energia.

Os códigos a seguir ilustram o processo de reescrever os programas entre as duas linguagens. Os dois códigos fazem a mesma coisa: piscam um LED com uma frequência de 1 segundo:

```
// Em C/Arduino
pinMode(13, OUTPUT);
while (1) {
    digitalWrite(13, HIGH);
    delay(1000);
    digitalWrite(13, LOW);
    delay(1000);
}

// Em Céu
output int PIN_13;
loop do
    emit PIN_13(high);
    await 1s;
    emit PIN_13(low);
    await 1s;
```

end

Note que as chamadas às funções em C, que não carregam nenhuma semântica de eventos, são substituídas em Céu por um vocabulário próprio que permite que a linguagem entenda os pontos de espera (comandos “await”) e possa colocar o microcontrolador em standby. Em testes preliminares como esse, conseguimos economias de ordem significativa (acima de 50%), mas ainda é preciso avaliar aplicações complexas onde há concorrência e uso de múltiplos sensores e atuadores.

No longo prazo, esperamos mostrar para desenvolvedores as vantagens de escreverem novas aplicações em Céu para tirarem proveito dos modos de standby automaticamente. Nessa direção, avaliaremos o tempo necessário para escrever (ou reescrever) as aplicações e os ganhos reais de eficiência energética.

## 12 Meses Finais e Trabalhos Futuros

### Arquiteturas e Aplicações de IoT Complexas

O nicho de sistemas embarcados restritos, que inclui o Arduino, cobre a parte substancial (e crescente) de aplicações IoT. Tipicamente, essas arquiteturas não requerem muitos recursos computacionais mas são sensíveis às dimensões físicas e consumo de energia. No entanto, a IoT também consiste de dispositivos conectados tradicionais, tais como roteadores, servidores e smartphones. Até 2016 existiam 3.9 bilhões de assinaturas de smartphones no mundo e esse número deve alcançar 6.8 bilhões até 2022 [1].

Smartphones usam arquiteturas muito mais complexas do que microcontroladores embarcados. Tipicamente essas arquiteturas dependem de um sistema operacional, uma pilha de TCP/IP completa, e podem executar múltiplas aplicações simultaneamente. Além de aplicações reativas, típicas de IoT, smartphones também executam computações puramente ativas, tais como processamento de imagens e funções criptográficas. Mesmo assim, smartphones são uma peça importante na IoT, servindo como uma interface comum aos humanos para processar, visualizar e atuar na rede.

Smartphones têm restrições similares de consumo de bateria e também podem tirar proveito das técnicas que propomos para sistemas embarcados restritos. De modo a transpor a barreira de dispositivos IoT restritos para também incluir os smartphones, iremos adotar uma abordagem análoga aos dois primeiros anos:

- **Infraestrutura de Hardware:** Usaremos o BeagleBone Black, que compartilha objetivos similares ao do Arduino, provendo uma plataforma barata e aberta mas que é adequada a aplicações mais ricas, tais como interfaces gráficas, multimídia, e jogos.
- **Infraestrutura de Software:** De modo a garantir standby automático para aplicações, toda infraestrutura de software, principalmente device drivers,

também terá que ser recriada usando ISRs em Céu.

- Aplicações: Além de aplicações IoT, as aplicações típicas de smartphones, tais como mensagens instantâneas e navegação Web, também poderão melhorar a sua eficiência energética através do modo de standby.

## Resultados Esperados

### 24 Meses Iniciais

O vocabulário de Céu dedicado à interação com o ambiente aumentará o nível de abstração dos programas para um nível mais próximo do domínio de IoT, provendo mais segurança e expressividade para programadores. Esse vocabulário se estenderá até o nível mais básico de rotinas de interrupção em uma abordagem inovadora.

Nossa proposta visa fazer com que todas as novas aplicações estejam sujeitas a modos de standby transparentemente. Sendo parte da infraestrutura de software, somente device drivers necessitarão de gerenciamento explícito de energia, e todas as aplicações construídas sobre eles se beneficiarão de eficiência energética automaticamente.

Céu é um projeto de 10 anos e tem uma implementação open-source madura que está disponível publicamente para downloads. Com a proposta de adaptação ao contexto de eficiência energética para a IoT, Céu poderá se tornar uma alternativa prática ao Arduino no curto prazo.

### 12 Meses Finais e Trabalhos Futuros

A transição de telefones simples para smartphones resultou na degradação do tempo de vida das baterias. No entanto, a maior parte do tempo, os smartphones estão ociosos nos nossos bolsos mas gastando energia. Esperamos aumentar consideravelmente a autonomia das baterias mantendo toda a funcionalidade de um smartphone moderno.

## Relevância e Impacto

De acordo com a Agência Internacional de Energia (IEA), existiam em 2013 em torno de 14 bilhões de dispositivos conectados tradicionais (ex., telefones TVs inteligentes). Esse número deve crescer para 25 bilhões até 2025 com a proliferação de dispositivos IoT (ex., lâmpadas inteligentes e tecnologia vestível). Dispositivos tradicionais e de IoT já superam o número de pessoas no planeta por um fator de dois e o tráfego de dados resultante deve crescer a uma taxa



exponencial nos próximos anos. No entanto, a maior parte da energia desses aparelhos é consumida quando eles estão em “standby” (modo em espera), ou seja, quando os dispositivos não estão desempenhando suas finalidades principais. A emissão anual de CO<sub>2</sub> relacionada somente a standby é equivalente a de 1 milhão de carros. A projeção de crescimento de IoT, juntamente com o efeito surpreendente do consumo de standby, fizeram com que a eficiência de standby para dispositivos conectados seja um dos seis pilares do “Plano de Ação para Eficiência Energética” do G20 [1].

Outras organizações também reportaram sobre a importância da economia de energia em dispositivos conectados [1]. Para o Internet Engineering Task Force (IETF), “o gerenciamento energético está se tornando um requisito adicional para redes devido a diversos fatores que incluem o aumento dos custos de energia, o impacto ecológico para a operação das redes, e a regulação de energia”. Para o American Council for an Energy-Efficient Economy (ACEEE), “o potencial para a nova eficiência energética permanece enorme, (...) devemos considerar uma abordagem sistêmica para escalar a eficiência energética. (...) a eficiência inteligente é adaptativa, antecipatória, e conectada”. Considerando iniciativas concretas, o grupo de trabalho “Electronic Devices and Networks” da IEA foca especificamente na questão do standby em dispositivos conectados [1]. A iniciativa da ACEEE em “Intelligent Efficiency” promove uma abordagem sistemática para otimizar o comportamento cooperativo de dispositivos de modo a buscar ganhos de energia como um todo. Ambas as abordagens (por dispositivo e sistêmica) envolvem soluções de software, dado que a economia de energia é uma política dinâmica que depende das demandas das aplicações e níveis de baterias em determinados momentos.

No que diz respeito à infraestrutura de telecomunicações de IoT, já existem padrões de baixo consumo de energia com diferentes demandas de alcance, velocidade e distribuição física. Como exemplo, o Bluetooth Low-Energy (BLE) é um substituto para o padrão Bluetooth clássico e é projetado para baixas velocidades em redes pessoais (PANs). O 6LoWPAN adapta o padrão IPv6 para baixo consumo e em dispositivos de processamento limitado. Essas tecnologias permitem transmissões mais eficientes, suportam topologias flexíveis e reduzem o tráfego consideravelmente. Elas também possibilitam o uso de modos de standby mínimos. No entanto, essas tecnologias ainda exigem o uso de software para controlar os modos ativos e de standby de maneira a construir uma IoT eficiente em termos de energia.

## Atividades de Pesquisa Relacionadas

### Projetos

Este projeto sendo agora submetido ao CNPq é baseado em um projeto de 5 anos aprovado pelo Instituto Serrapilheira mas que acabou sendo financiado somente

por um ano (apenas 12 dos 70 projetos inicialmente aprovados continuaram sendo financiados):

- Título: “Energy Efficiency for IoT Software in the Large”
  - Número do Processo: Serra-1708-15612
  - Agência de Fomento: Instituto Serrapilheira
  - Vigência: entre Março de 2018 e Fevereiro de 2019
  - Coordenador: Francisco Figueiredo Goytacaz Sant’Anna

## Trajectoria de Publicações

Como já destacado, tenho trabalhado no projeto e implementação da linguagem de programação Céu pelos últimos 10 anos. A linha de tempo a seguir enumera as publicações relacionadas a Céu nas áreas de linguagens de programação, sistemas embarcados, redes de sensores sem fio, jogos, e sistemas multimídia. O objetivo é sinalizar que esse caminho de pesquisa é contínuo, amadurecido e que ainda permanecerá como meu foco principal por muitos anos. Também destaco o Workshop REBLS voltado especificamente para linguagens reativas e que já está indo para a 6 edição anual dentro do SPLASH. Participei nele como autor, revisor, chair, e hoje faço parte do comitê permanente (<https://2019.splashcon.org/home/rebls-2019>).

- 2011
  - SenSys (PhD Colloquium) - The ACM Conference on Embedded Networked Sensor Systems
    - \* “Céu: A Reactive Language for Wireless Sensor Networks”
      - **primeira exposição de Céu em um colóquio para doutorandos**
- 2013
  - SenSys - The ACM Conference on Embedded Networked Sensor Systems
    - \* “Safe Concurrent Abstractions for Wireless Sensor Networks”
      - **paper principal sobre a parte estática de Céu: projeto e avaliação da linguagem**
  - REM - Workshop on Reactivity, Events and Modularity
    - \* “Advanced Control Reactivity for Embedded Systems”
      - **mecanismos de controle sobre Céu: tratamento de exceções, programação dataflow, paralelismo hierárquico**
  - Tese de Doutorado
    - \* “Safe System-level Concurrency on Resource-Constrained Nodes with Céu”
      - **descrição completa de Céu: projeto, avaliação, semântica, implementação**
- 2014

- REBLS - Workshop on Reactive and Event-based Languages & Systems
    - \* “Structured Reactive Programming with Céu”
      - **paper sobre a parte dinâmica de Céu: conceito de organismos, ponteiros e referências**
- 2015
  - TOSN - ACM Transactions on Sensor Networks
    - \* “Terra: Flexibility and Safety in Wireless Sensor Networks”
      - **máquina virtual de Céu com reprogramação remota**
  - REBLS - Workshop on Reactive and Event-based Languages & Systems
    - \* “Reactive Traversal of Recursive Data Types”
      - **tratamento de dados em Céu: uniões discriminadas, mecanismo de percorrimento, interpretador embutido**
  - Modularity - AOSD Conference On Modularity
    - \* “Structured Synchronous Reactive Programming with Céu”
      - **paper principal sobre a parte dinâmica de Céu: conceito de organismos, ponteiros e referências**
- 2016
  - WebMedia - Simpósio Brasileiro de Sistemas Multimídia e Web
    - \* “Céu-Media: Local Inter-Media Synchronization Using Céu”
      - **desenvolvimento de aplicações multimídia em Céu**
- 2017
  - TECS - ACM Transactions on Embedded Computing Systems
    - \* “The Design and Implementation of the Synchronous Language Céu”
      - **projeto e implementação de Céu em comparação com Esterel**
- 2018
  - SBGames - Simpósio Brasileiro de Games e Entretenimento Digital
    - \* “Structured Synchronous Reactive Programming for Game Development”
      - **desenvolvimento de jogos em Céu, padrões de programação de jogos**
  - LCTES - Languages, Compilers, Tools and Theory of Embedded Systems
    - \* “A Memory-Bounded, Deterministic and Terminating Semantics for the Synchronous Programming Language Céu”
      - **semântica formal de Céu, provas de terminação, determinismo e limites de memória**
- 2019
  - JSA - Journal of Systems Architecture
    - \* “A memory-bounded, deterministic and terminating semantics for the synchronous programming language Céu”
      - (versão estendida) **semântica formal de Céu, provas de**

## terminação, determinismo e limites de memória

### Software

Céu é uma linguagem com a implementação aberta e disponível para ser usada em diversos ambientes computacionais, tais como Arduino, SDL e LibUV. Cada ambiente computacional expõe um conjunto de eventos de entrada e saída que podem ser manipulados pelos programas escritos em Céu.

Estimamos que já foram escritas em torno de 100.000 linhas de código considerando todos os nossos repositórios envolvendo a linguagem:

- Céu: compilador e runtime da linguagem
  - <https://github.com/ceu-lang/ceu>
- Céu-Arduino: ambiente para usar Céu com o Arduino
  - <https://github.com/ceu-arduino/>
- Céu-SDL: ambiente para jogos e aplicações gráficas
  - <https://github.com/ceu-lang/ceu-sdl>
  - Pingus: jogo completo escrito em Céu
    - \* <https://github.com/fsantanna/pingus>
- Céu-libuv: ambiente para comunicação assíncrona em rede
  - <https://github.com/ceu-lang/ceu-libuv>
- Céu-Media: ambiente para aplicações multimídia
  - <https://github.com/rodrimc/ceu-media>

Os repositórios contém a implementação de Céu (em C, Haskell e Lua), testes (em Céu), bindings e bibliotecas para os ambientes (em C e Céu), drivers (em C e Céu) e aplicações (em Céu).

### Bibliografia

- [1] Estudos e Surveys sobre Dispositivos Conectados e Eficiência Energética:
  - OECD/IEA. More data less energy—Making network standby more efficient in billions of connected devices. Technical report, International Energy Agency, 2014.
  - G20's Energy Efficiency Action Plan: <https://www.iea-4e.org/projects/g20>, 2019.
  - EDNA initiative: <https://edna.iea-4e.org/>, 2019.
  - E. Tychon et al. Energy management (EMAN) applicability statement. Technical report, IETF, 2011.
  - R. N. Elliott, M. Molina, and D. Trombley. A defining framework for intelligent efficiency. Technical Report E125, American Council for an Energy-Efficient Economy (ACEEE), 2012.

- R. Brown, C. Webber, and J. G. Koomey. Status and future directions of the ENERGY STAR program. *Energy*, 27(5):505–520, 2002.
- G. Reiter. Wireless connectivity for the internet of things. Technical report, Texas Instruments, 2014.
- E. A. Rogers et al. Intelligent efficiency: opportunities, barriers, and solutions. Technical Report E13J, American Council for an Energy-Efficient Economy (ACEEE), 2013.
- N. Heuvelodp. Ericsson mobility report. Technical report, Ericsson, AB, 2017.
- [2] Trabalhos Relacionados em Ciência de Energia:
  - J. Flinn and M. Satyanarayanan. Managing battery lifetime with energy-aware adaptation. *ACM Trans. Comput. Syst.*, 22(2):137–179, May 2004.
  - W. Baek and T. M. Chilimbi. Green: A framework for supporting energy-conscious programming using controlled approximation. In *Proceedings of the 31st ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI ’10*, pages 198–209, New York, NY, USA, 2010. ACM.
  - A. Canino. Gradual mode types for energy-aware programming. In *Proceedings of SPLASH Companion 2015*, pages 79–80, New York, NY, USA, 2015. ACM.
  - A. Canino and Y. D. Liu. Proactive and adaptive energy-aware programming with mixed typechecking. In *Proceedings of PLDI 2017*, pages 217–232, New York, NY, USA, 2017. ACM.
  - M. Cohen et al. Energy types. In *Proceedings of OOPSLA ’12*, pages 831–850, New York, NY, USA, 2012. ACM.
  - H. Hoffmann. Jouleguard: Energy guarantees for approximate applications. In *Proceedings of the 25th Symposium on Operating Systems Principles, SOSP ’15*, pages 198–214, New York, NY, USA, 2015. ACM.
  - A. Kansal et al. The latency, accuracy, and battery (lab) abstraction: Programmer productivity and energy efficiency for continuous mobile context sensing. In *Proceedings of OOPSLA ’13*, pages 661–676, New York, NY, USA, 2013. ACM.
  - J. Park et al. Flexjava: Language support for safe and modular approximate programming. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015*, pages 745–757, New York, NY, USA, 2015. ACM.
  - A. Sampson et al. Enerj: Approximate data types for safe and general low-power computation. *SIGPLAN Not.*, 46(6):164–174, June 2011.
  - J. Sorber et al. Eon: A language and runtime system for perpetual systems. In *Proceedings of SenSys ’07*, pages 161–174, New York, NY, USA, 2007. ACM.
  - H. Zeng et al. Ecosystem: Managing energy as a first class operating system resource. *SIGARCH Comput. Archit. News*, 30(5):123–132,

- Oct. 2002.
- Y. Zhu and V. J. Reddi. Greenweb: Language extensions for energy-efficient mobile web computing. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '16*, pages 145–160, New York, NY, USA, 2016. ACM.
  - [3] *Linguagem C/Derivadas e Técnicas de Programação em Sistemas Embarcados*:
    - E. Baccelli et al. Riot os: Towards an os for the internet of things. In *Computer Communications Workshops (INFOCOM WKSHPS), 2013 IEEE Conference on*, pages 79–80. IEEE, 2013.
    - M. Barr. Real men program in C. *Embedded Systems Design*, 22(7):3, 2009.
    - M. de Icaza. Callbacks as our generations' go to statement. <https://tirania.org/blog/archive/2013/Aug-15.html> (accessed in Jul-2019), 2013.
    - Dunkels et al. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *Proceedings of LCN'04*, pages 455–462, Washington, DC, USA, 2004. IEEE Computer Society.
    - A. Dunkels, O. Schmidt, T. Voigt, and M. Ali. Protothreads: simplifying event-driven programming of memory- constrained embedded systems. In *Proceedings of SenSys'06*, pages 29–42. ACM, 2006.
    - D. Gay et al.. The nesC language: A holistic approach to networked embedded systems. In *Proceedings of PLDI'03*, pages 1–11, 2003.
    - J. Hill et al. System architecture directions for networked sensors. *SIGPLAN Notices*, 35:93–104, November 2000.
    - P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, et al. Tinyos: An operating system for sensor networks. *Ambient intelligence*, 35:115–148, 2005.
    - I. Maier, T. Rompf, and M. Odersky. Deprecating the observer pattern. Technical report, 2010.
    - E. Meijer. Reactive extensions (rx): curing your asynchronous programming blues. In *ACM SIGPLAN Commercial Users of Functional Programming, CUFPP '10*, pages 11:1–11:1, New York, NY, USA, 2010. ACM.
  - [4] *Arduino e Sistemas Embarcados*:
    - Atmel. ATmega328P Datasheet, 2011.
    - J. D. Brock, R. F. Bruce, and S. L. Reiser. Using arduino for introductory programming courses. *J. Comput. Sci. Coll.*, 25(2):129–130, Dec. 2009.
    - L. Buechley, M. Eisenberg, J. Catchen, and A. Crockett. The lilypad arduino: Using computational textiles to investigate engagement, aesthetics, and diversity in computer science education. In *Proceedings of SIGCHI '08*, pages 423–432, New York, NY, USA, 2008. ACM.

- C. Doukas and I. Maglogiannis. Bringing iot and cloud computing towards pervasive healthcare. In IMIS, 2012 Sixth International Conference on, pages 922– 926. IEEE, 2012.
  - V. Georgitzikis, O. Akribopoulos, and I. Chatzigiannakis. Controlling physical objects via the internet using the arduino platform over 802.15. 4 networks. IEEE Latin America Transactions, 10(3):1686–1689, 2012.
  - K. Gomez et al. Energino: A hardware and software solution for energy consumption monitoring. In WiOpt, 2012 10th International Symposium on, pages 311–317. IEEE, 2012.
  - P. Jamieson. Arduino for teaching embedded systems. are computer scientists and engineering educators missing the boat? Proc. FECS, 289294, 2010.
  - D. Kushner. The making of arduino. IEEE Spectrum, 26, 2011.
  - K. Mandula et al. Mobile based home automation using internet of things (iot). In ICCICT, 2015 International Conference on, pages 340–343. IEEE, 2015.
  - J. Sarik and I. Kymissis. Lab kits using the arduino prototyping platform. In Frontiers in Education Conference (FIE), 2010 IEEE, pages T3C–1. IEEE, 2010.
- [5] Céu:
    - Site de Céu: <http://www.ceu-lang.org/>
    - F. Sant’Anna. Céu: A reactive language for wireless sensor networks. <http://www.cse.ust.hk/~lingu/SenSys11DC/>, 2011.
    - F. Sant’Anna, N. Rodriguez, and R. Ierusalimschy. Céu: Embedded, Safe, and Reactive Programming. Technical Report 12/12, PUC-Rio, 2012.
    - F. Sant’Anna. Safe System-level Concurrency on Resource-Constrained Nodes with Céu. PhD thesis, PUC-Rio, 2013.
    - F. Sant’Anna, N. Rodriguez, and R. Ierusalimschy. Advanced control reactivity for embedded systems. Workshop on Reactivity, Events and Modularity (REM’13), 2013.
    - F. Sant’Anna, N. Rodriguez, R. Ierusalimschy, O. Landsiedel, and P. Tsigas. Safe System-level Concurrency on Resource-Constrained Nodes. In Proceedings of SenSys’13. ACM, 2013.
    - F. Sant’Anna et al. Structured reactive programming with céu. Workshop on Reactive and Event-based Languages & Systems (REBLS’14), 2014.
    - F. Sant’Anna et al. Reactive traversal of recursive data types. Workshop on Reactive and Event-based Languages & Systems (REBLS’15), 2015.
    - F. Sant’Anna, N. Rodriguez, and R. Ierusalimschy. Structured Synchronous Reactive Programming with Céu. In Proceedings of Modularity’15, 2015.
    - R. Santos, G. Lima, F. Sant’Anna, and N. Rodriguez. Céu-Media:

- Local Inter-Media Synchronization Using Céu. In Proceedings of WebMedia'16, pages 143–150, New York, NY, USA, 2016. ACM.
- F. Sant’anna, R. Ierusalimschy, N. Rodriguez, S. Rossetto, and A. Branco. The design and implementation of the synchronous language Céu. ACM TECS, 16(4):98:1–98:26, July 2017.
  - Santos, Rodrigo C. M. ; Lima, Guilherme F. ; Sant’Anna, Francisco ; Ierusalimschy, Roberto ; Haeusler, Edward H. . A memory-bounded, deterministic and terminating semantics for the synchronous programming language Céu. In: the 19th ACM SIGPLAN/SIGBED International Conference, 2018, Philadelphia. LCTES 2018. New York: ACM Press, 2018. p. 1.
  - Lima, Guilherme F. ; Santos, Rodrigo C.M. ; Ierusalimschy, Roberto ; Haeusler, Edward H. ; Sant’Anna, Francisco . A memory-bounded, deterministic and terminating semantics for the synchronous programming language Céu. Journal of Systems Architecture, 2019.