Imperative Reactive Programming with Céu

May 2, 2014

1 Abstract

The origins of reactive programming date back to the early 80's with the co-development of two styles of synchronous languages: The imperative style of Esterel organizes programs with control flow primitives, such as sequences, repetitions, and also parallelism. The dataflow style of Lustre represents programs as graphs of values, in which a change to a node updates its dependencies automatically.

In recent years, Functional Reactive Programming modernized the dataflow style and became mainstream, producing a number of languages and libraries, such as Flapjax, Rx (from Microsoft), React (from Facebook), and Elm. In contrast, the imperative style did not follow this trend and is now confined to the domain of real-time embedded control systems.

We present the programming language Céu, a contemporary outlook of imperative reactivity that aims to expand the application domain of this family with new abstraction mechanisms. Céu has its roots on Esterel and relies on a similar synchronous and deterministic execution model that simplifies the reasoning about concurrency aspects. The example below in Céu prints the "Hello World!" message every second, terminating with a key press:

```
par/or do
   every 1s do
      print("Hello World!");
   end
with
   await KEY;
end
```

The use of control primitives, such as `await`, `every`, and `par/or`, hides most complexity related to flows of execution and their life cycles (e.g. starting, aborting, and resuming them). In particular, the `await` statement is the most representative of Céu, capturing the imperative and reactive nature of the language at the same time. Awaiting multiple events in parallel releases programmers from the infamous *callback hell*, which is often a concern in general purpose imperative languages.

The new abstraction mechanisms of Céu to be presented expands classical object orientation with instances that are reactive to the environment. They make imperative reactivity suitable for applications outside the embedded domain. As an example, we implemented in Céu a game of moderate complexity for Android. The imperative reactive style of Céu is an effective alternative to Functional Reactive Programming that complements the realm of synchronous programming.

2 Comments

- Presentation Outline:
  - Reactive applications: characteristics, challenges
  - Reactive models: Declarative/Dataflow/FRP vs Control/Imperative
  - Overview of Céu
  - Reactive "Hello World!"
  - Reactive abstractions in Céu
  - Demos: aprox. 10 incremental live demonstrations
- Intended audience:
  - Programmers interested in real-time and reactive concurrency (embedded systems, games, and multimedia).
  - Requires knowledge of C or C++.
- Previous StrangeLoops?
  - No, it would my first time.
- Speaking Experience:
  - Talks related to Céu:
    - 2014 - Arduino Day - Presentation (`http://thesynchronousblog.wordpress.com/2014/03/31/ceu-on-arduino-day/`)
    - 2013 - SenSys Conference - Full Paper (`http://sensys.acm.org/2013/index.html`)
    - 2013 - REM Workshop (inside OOPSLA) - Full Paper (`http://soft.vub.ac.be/REM13/`)
    - 2011 - SenSys Conference - Doctoral Colloquium (`http://www.cse.ust.hk/~lingu/SenSys11DC/`)
  - I'm also a CS teacher and we've been using Céu in Distributed System classes.
- Maturity of Céu:
  - Developed over the past 6 years in a PhD thesis (mine). (`http://www.ceu-lang.org/`)
  - Open sourced last year. (`http://github.com/fsantanna/ceu/`)
  - Participating this year with one project in the "Google Summer of Code" (under the umbrella of the LabLua at PUC-Rio). (`http://www.lua.inf.puc-rio.br/gsoc/ideas2014.html`)
  - Evaluated in the context of Wireless Sensor Networks together with the DCS group at Chalmers University (Sweden). (`http://www.cse.chalmers.se/research/group/dcs/`)
  - Used by other graduate and undergraduate students.
  - No sensible uses outside of our domains (as far as we know).