Where Do Events Come From?

Reactive Programming From The Ground Up

Anonymous Author(s)

Abstract

In reactive and event-based systems, execution is guided by an external environment which produces inputs and consumes output events to and from the applications. Reactive languages provide dedicated syntax and semantics to deal with events and greatly simplify the programming experience in this domain. Nevertheless, the environment is typically prefabricated in a host language and the very central concept of events is implemented externally to the reactive language. This separation exposes the environment as a rigid system component that evolves in separate from the application. In this work, we propose an interrupt handler primitive for a reactive language targeting embedded systems in order to take control of the whole event loop, from the hardware input source and back to the hardware output sink. We propose the new primitive in the context of the structured reactive language Céu and discuss how it synergizes with the structured paradigm to provide automatic standby for applications and hot swapping for drivers during runtime.

Keywords interrupt service routine, sleep mode, synchronous reactive programming

1 Introduction

Reactive applications interact continuously and in real time with the external world through input and output devices (e.g., buttons, displays, timers, etc.). These interactions are typically represented as input events flowing from the devices to the application and as output events flowing from the application to the devices. As illustrated in Figure 2, devices can be encapsulated as a single component, *the environment*, which controls the system execution in an event loop: the application sits idle waiting for an input; the environment awakes the application on the occurrence of an input; the application reacts to the input and generates back one or more outputs; the application becomes idle and the loop restarts.

The environment is typically implemented in a host language (e.g., C) and controls the main event loop, invoking entry points in the reactive language runtime on the occurrence of inputs and also receiving output calls, both through a documented API. As examples,

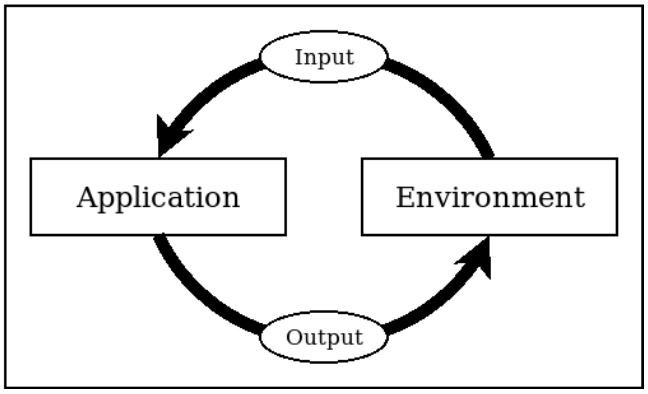

Figure 1. Event loop in reactive systems. The environment controls the application through input & output events.

Esterel [1] relies on C for passing events between the environment and the running program [4], while Elm [3] uses the concept of *ports*, which allows sending out values to JavaScript as commands and listening for values as subscriptions [2].

The event-based separation between the application and the environment is arguably inevitable and happens at the right level of abstraction, since it connects the application with the operating system resources through a non-invasive API. However, it involves two languages and the programmer may have to unnecessarily deal with multiple syntaxes, incompatible type systems, and different address spaces. Furthermore, in the context of embedded systems, a proper host OS may even be absent or lacking enough drivers, which requires more low-level intervention from the application.

In this work, we propose an interrupt handler primitive for a reactive language in order to take control of the whole event loop, from the hardware input source and back to the hardware output sink. We propose the new primitive in the context of the structured reactive language Cu and discuss how it synergizes with the structured paradigm to provide automatic standby for applications and hot swapping for drivers during runtime.

with different syntaxes and semantics. For instance, , type systems, and

abstraction - rigid - two languages - syntax and semantics - integrates with features such as lexical scope, type system - platforms w/o OS

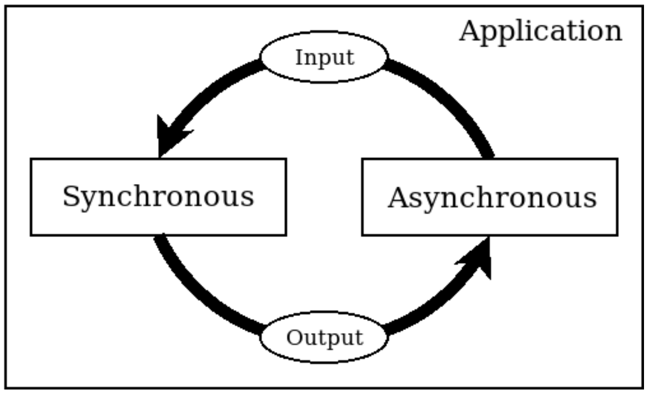

Figure 2. Event loop in reactive systems. The environment controls the application through input & output events.

what would be
output (int,int) O;
becomes
output (int x, int y) O do end
- excel define formulas for input but not the input itself
output -¿ input
the environment sits inverse reacts to output and generates inputs
remain idle and react to the occurrence of events

References

[1] Frédéric Boussinot and Robert De Simone. 1991. The Esterel language. *Proc. IEEE* 79, 9 (Sep 1991), 1293–1304.
[2] Evan Czaplicki. 2018. Elm and JavaScript Interop. https://guide.elm-lang.org/interop/javascript.html (accessed in Jul-2018).
[3] Evan Czaplicki and Stephen Chong. 2013. Asynchronous functional reactive programming for GUIs.. In *Proceedings of PLDI'13*. 411–422.
[4] Dumitru Potop-Butucaru, Stephen A Edwards, and Gerard Berry. 2007. *Compiling esterel*. Vol. 86. Springer Science & Business Media.

A Appendix

Text of appendix ...