Viewpoints Research Institute

# On Serializing and Deserializing FRP-style Interactive Programs

Yoshiki Ohshima

This material is based upon work supported in part by the National Science Foundation under Grant No. 0639876. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

VPRI Memo M-2013-001

# On Serializing and Deserializing FRP-style Interactive Programs

Yoshiki Ohshima

February 15, 2013

## 1 Introduction

Our group has been developing a GUI framework called KSWorld [7] that draws upon Functional Reactive Programming (FRP) [3]. The core idea is to describe the application logic in terms of the graph of data dependency in a time-aware manner. As in other GUI frameworks, there is a type of basic graphical object, which is called the "`Box`". A `Box` has fields that represent its characteristics such as its location, its graphical appearance (called a shape) etc., and also its dynamic actions. What separates the KSWorld from more conventional GUI framework is that these fields are nodes, or *streams*, in the dependency graph.

The KSWorld is designed to support end-user authoring in the spirit of Etoys [1]. The user can add and remove `Box`es and streams into his project at any time while the system/application is running. When the user is satisfied, a set of `Box`es that the user is interested in will be serialized to be stored or sent over the network, and deserialized at the other end of the pipe. The deserialized project may be merged into the environment that may be quite different from where it was created.

Upon serializing a project made in this manner, which elements in the project to export and how to import them is an interesting question. (But, here is the short answer: "behaviors" should be saved with their current values while the "events" should not be saved with their current values. The behaviors should be re-evaluated upon loading.)

In the following sections, we explain the idea more deeply.

## 2 Events and Behaviors

In FRP, there are two types of time-varying variables. One is called the "behavior", which represents a continuous value over time whose time domain stretches from $-\infty$ to $\infty$. In other words, a behavior has *always* a value at any given time, including at the "current time", which denotes the logical time where the system is. (The value of the behavior at the current time is "current value".) Another type is called the "event", which represents a discrete sequence of values that spread over time, but until the first value in the event arrives, the "current value" is undefined. (The terms "behaviors" and "events" can be confusing because these have other meanings.)

As often suggested in discussions [4] [5], the distinction is rather subtle in the implementation; a behavior can be implemented as an event with the latest value cached as the current value. They are connected in the dependency graph, and a change in one triggers the changes in its dependents. The events and behaviors in the KSWorld is implemented thusly; there is a single kind of type called `EventStream`.

## 3 What to Save?

As described above, the fields of a `Box` are all EventStreams. Upon saving a `Box` into the external form, some values, such as the location of the `Box`, the shape of the `Box`, etc. should be saved so that they can be reconstructed when loaded. On the other hand, events do not retain their current values, as the value of the event is useful only at the time when the event occurs.

## 4 Examples

Let us take an example. Imagine that there is a `Box` who is reacting to click events, and moving to right by 20 pixels for each click:

```
this.myMover = $$(@click.doE(() -> this.translateBy(P(20, 0))))
```

There are three streams involved; `click`, that is an event and receives a new value when the box is clicked. The `@myMover` stream describes a reaction to `click`; it uses the side-effecting method `translateBy`. In the `translateBy` method, there is a stream that represents the transformation (called `@transformation`) of the box from its container. The double-dollar quote ("$$") serves as a macro and the expression enclosed in it is expanded to an expression that creates an `EventStream`.

(As described in [7], one could write the same effect without using the side-effecting method but more directly, even the presence of self-referencing stream such as in this case. Explaining this is out of scope of this memo.)

The `Box` *always* has a valid value in `transformation`. When this `Box` is exported and later imported, the position of the `Box` should be as it was, and the next click should move the box by 20 pixels from that location.

But what if a behavior is dependent on another behavior? Let us imagine that a `Box` is monitoring the color of the extent of top-level `Box` (also known as the "window") and showing its textual representation:

```
this.extentPrinter =
  $$(this.textContents(@__topContainer__.extent.asString()))
```

The `__topContainer__.extent` notation retrieves the reference to the `extent` stream of the window. It is symbolically referenced so that exporting and importing it does not break the reference. The value of `extent` (in Point) is converted to a string by the `asString()` method and the `textContents()` method puts characters in itself. Again, `extent` is a behavior and something that always has a valid value.

When this `Box` is exported and imported to a new session with a new window, the `Box` should not wait until the next change to the `extent` of the new top-level window (which happens to be a relatively rare-event). Rather, all behaviors in the imported `Boxes` that depend on behaviors should be "synchronized" with the logical time of the current system *regardless of the original logical time when they were exported*. So, upon loading the `extentPrinter`, the stream needs to be evaluated and the user sees the extent of the window in the box as soon as it appears on the screen.

## 5 Implementation

To ensure this, an EventStream has a flag to denote whether it is a behavior or an event. Any stream that is created as a "value stream" (c.f. [7]) is a behavior, as well as a stream that has received `startsWith()` call.

An `EventStream` has a (non-stream) field called `currentValue` to hold the current value of the stream. It also has `lastUpdateTime`, which denotes the logical time when the stream acquired a new value. The value of `lastUpdateTime` is compared against the `lastUpdateTime` of the dependencies to determine whether it needs to be updated.

One might think that the `lastUpdateTime` field is not necessary in an implementation of FRP, especially for a push-based one. In the KSWorld implementation, this turned out to be useful. For instance, the field can have a special value (-1, which never be a valid logical time) that denotes that the stream has to be re-evaluated. The behavior going through serialization uses this to trigger the update.

All such data are stored in a restricted form of S-expressions.

## 6 Conclusions

The Frank document editor [2] is being written in KScript. The Frank editor is made to edit such end-user projects, but the editor itself is built in the KSWorld and uses the FRP-style code. Parts of Frank have been saved and loaded for making further revisions during the development, and the externalization scheme seems to be working well for our purpose.

## References

[1] Etoys. `http://squeakland.org`.

[2] D. Amelang, B. Freudenberg, T. Kaehler, A. Kay, S. Murrell, Y. Ohshima, I. Piumarta, K. Rose, S. Wallace, A. Warth, and T. Yamamiya. STEPS Toward Expressive Programming Systems, 2011 progress report. Technical report, Viewpoints Research Institute, 2011. Submitted to the National Science Foundation (NSF) October 2011.

[3] C. Elliott and P. Hudak. Functional reactive animation. In *International Conference on Functional Programming*, 1997.

[4] A. Guha. On the Flapjax[6] mailing list: `https://groups.google.com/forum/#!topic/flapjax/oKPYa68Dhas`.

[5] A. Kay. Personal communication on variables being spreadsheet cells with their current values cached.

[6] L. A. Meyerovich, A. Guha, J. Baskin, G. H. Cooper, M. Greenberg, A. Bromfield, and S. Krishnamurthi. Flapjax: a programming language for Ajax applications. *SIGPLAN Not.*, 44(10):1–20, Oct. 2009.

[7] Y. Ohshima, B. Freudenberg, A. Lunzer, and T. Kaehler. A Report on KScript and KSWorld. Technical report, Viewpoints Research Institute, 2012. VPRI Research Note RN-2012-001.