

Re: Why the dataflow paradigm?

Source: <http://coding.derkeiler.com/Archive/Lisp/comp.lang.lisp/2004-11/1713.html>

From: Kenny Tilton (*ktilton_at_nyc.rr.com*)

Date: 11/22/04

Date: Mon, 22 Nov 2004 20:50:41 GMT

Kalle Olavi Niemitalo wrote:

> *I have a feeling I'll regret posting this.*
>
> *Kenneth Tilton <ktilton@nyc.rr.com> writes:*
>
>
>>*One thing you are missing is that, for a display attribute such as*
>>*highlightedp, someone is always asking. The user. What good is it to*
>>*know that your functional highlightedp would return the right answer if*
>>*nothing is done to force a redraw of the given widget? This is a*
>>*dataflow problem, including flowing all the way out to the user.*
>
>
> *I wonder if a hybrid approach is possible.*

Why bother with the added complexity? And the slower performance? Which is what lazy does for you. The propagation stops if any dependent cell gets recalculated and decides not to change. But if I am not going to propagate immediately, I have to mark as invalid not just my immediate dependents but the entire downstream (if you will) dependency graph. I have lost the optimization of stopping propagation when possible. Every cell downstream is now flagged as invalid and all these rules will get kicked off when they get asked.

> *The nodes (do you*
> *call them cells?) that directly affect the user interface would*
> *be tagged as requiring immediate updates, and they would*
> *propagate this tag to their data providers. Then, when a data*
> *source changes, only the tagged direct descendants would be*
> *updated immediately; the rest would just be marked as stale and*
> *updated on demand.*

Preemy op. Have you seen a dataflow set-up where eager prop was both slow and unnecessary? Were synapses unable to help?

>

comp.lang.lisp: Re: Why the dataflow paradigm?

- > *On the other hand, even for UI objects, it would make sense to*
- > *delay updating them (and sending OpenGL display lists over the*
- > *network) until the incoming event has been fully processed. For*
- > *example, in a text editor, replacing the selection with a paste*
- > *should not require first deleting the selection from the screen*
- > *and scrolling the following text up, and then scrolling it back*
- > *down in order to insert the paste.*

That is not how GUIs work. All that happens is that each GUI element tells the OS "my turf needs redrawing", or a display list gets rebuilt. Each event propagates fully before polling for the next event (which would be an update event on Mac OS 9 since those were artificial events given top priority. So one event would have all the consequences you describe in the text-editing model, and all the visual elements involved would say "update my turf" (nothing happens except that the OS records the region and combines it with any others, saving them up for one update event), and then the event handling is over. We get the next (update) event and redraw those widgets overlapping the update region (if we take the trouble to check, which most of us do).

- >
- > *For extra points, stop updating the UI objects when they are*
- > *obscured by another window. Then the nodes on which they depend*
- > *might not have to be updated either. Though if you don't use*
- > *many intermediate nodes that are expensive to compute, perhaps*
- > *this isn't worth the trouble.*

Well you do not know about other windows in the OSes I have used. But the main point is that there is simply no need for all this, except in the imagination not just of you but anyone I have ever met who hears "eager propagation". :) I think I could clear a room faster with that phrase than "fire!". :)

- >
- > *If you make node A depend on node B (so that A is updated*
- > *whenever B changes), and then forget about A, will there still*
- > *be a reference from B that prevents collecting A as garbage?*
- > *Do you use weak references of some kind, or do you rely on*
- > *UNWIND-PROTECT to detach all nodes that are no longer needed?*

There goes the free lunch. :) I do have to have instances which are leaving the application model for good bow out gracefully. They also must come into existence gracefully, hence the two functions to-be and not-to-be. The good news is that the populations of my app models are also mediated by cells. The KIDS slot of a parent has an output callback which takes care of calling to-be or not-be as necessary. so to-be gets called once at app start-up on a root model instance, and that is that.

kt

--

Re: Why the dataflow paradigm?

comp.lang.lisp: Re: Why the dataflow paradigm?

Cells? Cello? Celtik?: <http://www.common-lisp.net/project/cells/>
Why Lisp? <http://alu.cliki.net/RtL%20Highlight%20Film>