Live Programming for Event-based Languages

Christopher Schuster, Cormac Flanagan
Software and Languages Research Group, University of California, Santa Cruz

Imperative updates to render output and register event handlers hinder live programming

Example: JavaScript code with DOM manipulation

Code for generating output is spread accros the program

Changes to code and state might cause the output to be outdated

Source

```
<div id="c">Count: 0</div>
<button id="b">Inc!</button>

<script>
  var i = 0;
  $("#b").on("click", function() {
    $("#c").html("Count: " + (++i));
  });
</script>
```

"Dead" View

Count: 2 Inc!

Restarting the program resets the state which might delay the programming/debugging process

Changing `"Count: "` to `"Clicks: "` should update the output immediately but ...

... rerunning this function increments `i`! ⚡

... reloading the application resets our state: `i` ⚡

Seperating rendering and event handling in an event-based language enables live programming, time travel and continous feedback

The global application state gets initialized on program start

Event handlers can change the state but not the output

The output is always up-to-date with the current state and code

Try it out yourself! levjj.github.io/rde

Source

```
var i = 0;

function inc() { ++i; }

function render() {
  return (
    <div>Count: {i}
      <button onclick={inc}>Inc!</button>
    </div>);
}
```

Live View

Count: 2 Inc!

Time Control

Version 8

State 3

The `render` function computes the output but cannot change the state (Here, we use JSX but other ways of generating output would also work)

Tracking changes to the code and the state enables time traveling in two dimensions

No general Solution For Dynamic Software Updating/Hot swapping - Design Space of Alternatives

Replay recorded events	Continue with previous state	Specify migrations manually	"Best Effort" data migration
state+code always consistent	potentially inconsistent	useful for production systems	works well for small changes
slow	retains state and context		

Live Programming aims to provide quick and continous feedback, so needs to minimize the impact on development/debugging

Challenges

- Updating functions in the active call stack
- Output out of date
- Updating function values/closures in the state
- New code might not be compatible with old state

Proposed Solutions

- ✓ Use single event loop to update code
  Only update when no event handler is running
- ✓ Seperate rendering from event handling
  `render` cannot change state, `handle` cannot change output
- ! Restrict application state
  Closures/function values not allowed in application state
- ✗ Programmer may need to restart

Semantics for Live Programming and Time Travel in Event-based Languages

$$e ::= \lambda x.\ e \mid e(e) \mid x \mid \{x : e, ...\} \mid \ldots \text{ (Expressions)}$$
$$v ::= \lambda x.\ e \mid \{x : v, ...\} \mid \ldots \text{ (Values)}$$
$$p ::= \{\text{init} : e,\ \text{handle} : e,\ \text{render} : e\} \text{ (Programs)}$$
$$q ::= [\text{event } v] \mid [\text{swap } p] \mid [\text{reset}] \mid [\text{time } n] \text{ (Events)}$$
$$e \downarrow v \text{ (Evaluation)} \qquad \langle q, p, \vec{S}\rangle \Downarrow \langle p', \vec{S}', O\rangle \text{ (Updates)}$$

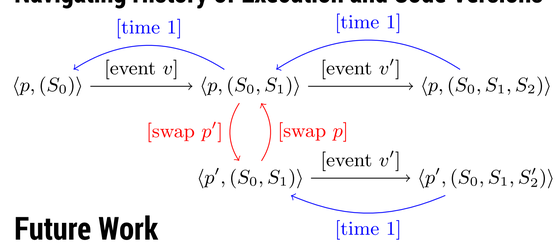
$$\frac{p.\text{handle}(v, S_j) \downarrow S_{j+1} \qquad p.\text{render}(S_{j+1}) \downarrow O}{\langle [\text{event } v], p, (..., S_j)\rangle \ \Downarrow\ \langle p, (..., S_j, S_{j+1}), O\rangle}\ \text{E-EVENT}$$

$$\frac{p'.\text{render}(S_j) \downarrow O}{\langle [\text{swap } p'], p, (..., S_j)\rangle \ \Downarrow\ \langle p', (..., S_j), O\rangle}\ \text{E-SWAP}$$

$$\frac{p.\text{init} \downarrow S_0' \qquad p.\text{render}(S_0') \downarrow O}{\langle [\text{reset}], p, \vec{S}\rangle \ \Downarrow\ \langle p, (S_0'), O\rangle}\ \text{E-RESET}$$

$$\frac{p.\text{render}(S_n) \downarrow O}{\langle [\text{time } n], p, (..., S_n, ...)\rangle \ \Downarrow\ \langle p, (..., S_n), O\rangle}\ \text{E-TIME}$$

Navigating History of Execution and Code Versions



Future Work

Enforce constraints (static analysis, contracts, ...)
Optimize performance (*incr. computation* for re-rendering)
Improved time travel (*copy-on-write* to keep old state)
Programming by Example (direct manipulation, synthesis,...)

Related Work / Inspirations

Bret Victor: *Inventing on Principle* (2012)
Burckhardt et. al. *It's Alive! Continuous Feedback...* (PLDI '13)

Squeak/Smalltalk Elm React/Redux

Short paper presented at REBLS'15 workshop. Online Live Demo: http://levjj.github.io/rde/ Source Code: http://github.com/levjj/rde/ Contact: {cschuste,cormac}@ucsc.edu