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

Example: JavaScript code with DOM manipulation

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

Example: JavaScript code with DOM manipulation.

```javascript
var i = 0;
function inc() { ++i; }
function render() {
    return (<div>
        Count: {i}
        <button onclick={() => inc()}>
            Inc!
        </button>
    </div>);
}
```

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

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

Navigating History of Execution and Code Versions

Future Work

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

Related Work / Inspirations

Burckhardt et. al. It's Alive! Continuous Feedback... (PLDI'13)
Squeak/Smalltalk
Elm
React/Redux

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

Replay recorded events
Continue with previous state
State+code always consistent
Potentially inconsistent
Slow
Retains state and context

Live Programming aims to provide quick and continuous 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Output out of date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
Separate rendering from event handling
Restrict application state
Closures/function values not allowed in application state
Programmer may need to restart

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