cljs + gui

simplicity yields possibilities

slides and code online (at meetup)

guis are nice, programming them not so

introducing:
the horror called state

```
+------------------------+
|                        |
|  Services & Factories  <------------------+
|                        |                  |
+------------^-----------+                  |
             |                              |
       +-----v----+                 +-------+-------+
       |          |                 |               |
       |  Models  <----------------->  Controllers  |
       |          |                 |               |
       +-----+----+                 +---^---^-------+
             |                          |   |
             |                          |   |
             |  +-----------------------+   |
             |  |                           |
             |  |                           |
     +-------v--v--+                +-------v------+
     |             |                |              |
     |  Templates  <---------------->  Directives  |
     |             |                |              |
     +-------------+                +--------------+
```

wouldn't it be great if...

```
+--------------------+        +--------------------+
|                    |        |                    |
|                    |        |                    |
|       state        <-------->       render       |
|                    |        |                    |
|                    |        |                    |
+--------------------+        +--------------------+
```

React

- Declarative view layer, yes!
- Unidirectional flow (No two way data binding!)
- State > view problem resolved, right!?

Reagent (& Om)

- Clojurescript wrappers for React.
- Truly reusable components (through cursors or lenses)
- Immutable data structures out of the box!

```
(defn comment-box []
    [div.commentBox
        [h1 “Comments”]
        [comment-list {:data comments}]
        [comment-form]])
```

“Clojure's sweet spot is any application that has state.”

— Stuart Halloway

Data manipulation made simple

```
(def state (atom {:deeply
                   {:nested
                    {:data
                     {:structure true}}}}))

(swap! state assoc-in [:deeply :nested
                       :data :structure] false)
```

free goodies

- hotswap code
- time travelling
- introspection
- portable state
- cross session persistant state

time travelling

being able to traverse back and forth in time aka. undo/redo.

demo: om todomvc by David Nolen[1]
demo: goya by Jack Schaedler[2]

[1] Om TodoMVC App Undo src

[2] Goya App Repo

```clojure
(def app-history (atom [@app-state]))

(add-watch app-state :history
  (fn [_ _ _ n]
    (when-not (= (last @app-history) n)
      (swap! app-history conj n))
    (set! (.-innerHTML (.getElementById js/document "message"))
      (let [c (count @app-history)]
        (str c " Saved " (pluralize c "State"))))))

(aset js/window "undo"
  (fn [e]
    (when (> (count @app-history) 1)
      (swap! app-history pop)
      (reset! app-state (last @app-history)))))
```

source: http://swannodette.github.io/2013/12/31/time-travel/

portable state

being able to copy paste state.

demo: spagetthi by me[3]

demo: circleci by Circle CI[4]

[3] spaggethi src

[4] circle ci video

```clojure
(defn focus-input [e app owner]
  (if (and (or (.-ctrlKey e) (.-metaKey e)) (not= (.-keyCode e) 86))
    (let [node (om/get-node owner)]
      (set! (.-value node) (t/write w @app))
      (.select node))))

(defn paste-state [app owner]
  (let [input-data (.-value (om/get-node owner))]
     (om/update! app (t/read r input-data))))

(defcomponent clipboard [app owner]
  (did-mount [_]
    (.listen goog/events js/document "keydown" #(focus-input % app owner)))
  (render [_]
          (html [:input {:type "textarea" :style {:opacity 0}
                         :onPaste (fn [] (js/setTimeout #(paste-state app owner) 30))}])))
```

source: https://github.com/jellea/spaghetti/blob/master/src/cljs/spaghetti/core.cljs#L27-L42

other advantages

- **compiler** - fetches most of the syntax errors!
- **syntax** - structural editing
- **core.async** - concurrency management
- **all inc** - no grulpack needed (yay!)
- **hiccup** - template syntax
- **static typing** - optional!
- **sharing code** with backend

“but I like javascript…”

- **mori/immutable.js** - clj data structures in js[6]
- **vec** - showcase of immutablejs[7]
- **react-hot-loader** - webpack plugin[8]
- **js-csp** - core.async[9]

[6] mori immutablejs

[7] vec

[8] hot-loader

[9] js-csp

Worth a look

- **Re-frame** - a Reagent “framework”[10]
- **Datascript** - client side database[11]
- **Elm** - signals[12]
- **Zelkova** - Elm FRP signals[13]

[10] re-frame

[11] datascript

[12] elm

[13] zelkova

questions? thanks.

twitter + github: @jellea

mail: m@jelle.io