



QL in Lisp (Clojure)

Rico Huijbers
Sioux Embedded Systems
rico.huijbers@sioux.eu

Why Clojure

- Perfect for (horizontal) DSLs
- I want a general purpose programming language that helps me raise the level of abstraction
- Selfish: learning a Lisp

QL

```
(defform box1-house-owning
  [boolean has-sold-house      "Did you sell a house in 2010?"]
  [boolean has-bought-house    "Did you buy a house in 2010?"]
  [boolean has-maint-loan      "Did you enter a loan for
                                maintenance/reconstruction?"]

  [group has-sold-house
    [currency selling-price     "Price the house was sold for"]
    [currency private-debt      "Private debts for the sold
                                house"]
    [calc value-residue (- selling-price private-debt)
      "Value residue"]])
```

QL = pattern matching

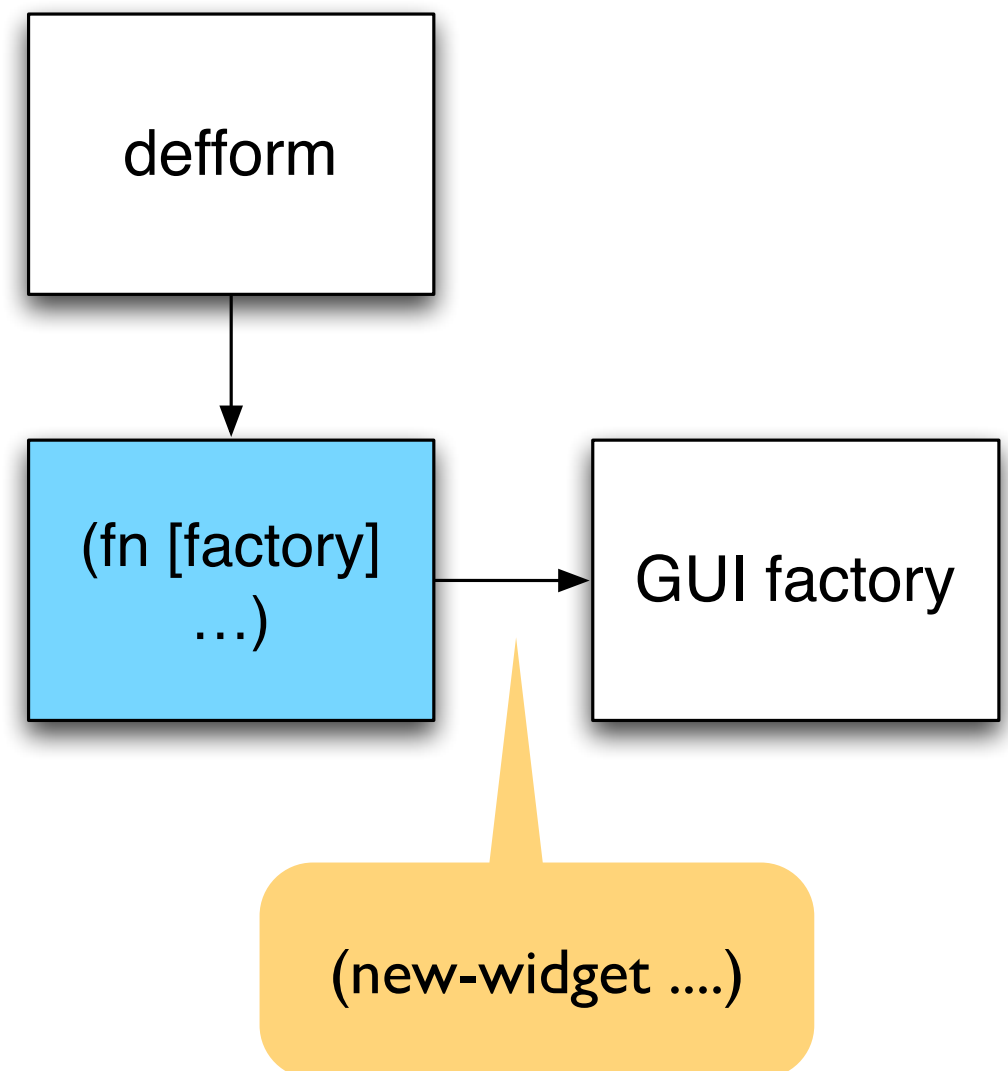
```
(defn create-elements [element gui-fact]

  (match [element]
    [['calc name expr caption]] `(output-element ...)
    [['group expr & subelements]] `(group-element ...)
    [['type name caption]]       `(input-element ...)
    :else ...))
```

(Using a DSL called `clojure.core.match`)

QL implementation

- Generate a function that takes a factory which creates the UI
- Using a Swing DSL for creating the GUI (seesaw)



Expressions (1)

- First idea: completely reuse Clojure expressions
- Only thing that needs to be replaced is value lookup:

```
(- price debt)
```



```
(fn [lookup] (- (lookup 'price)  
                (lookup 'debt)))
```

Expressions (2)

- We have the annoying “undefined” value, so we need to lift operators

```
(def operators
  {'+ (lift +)
   '- (lift -)
   ...
  })
```

```
(- price debt)
```



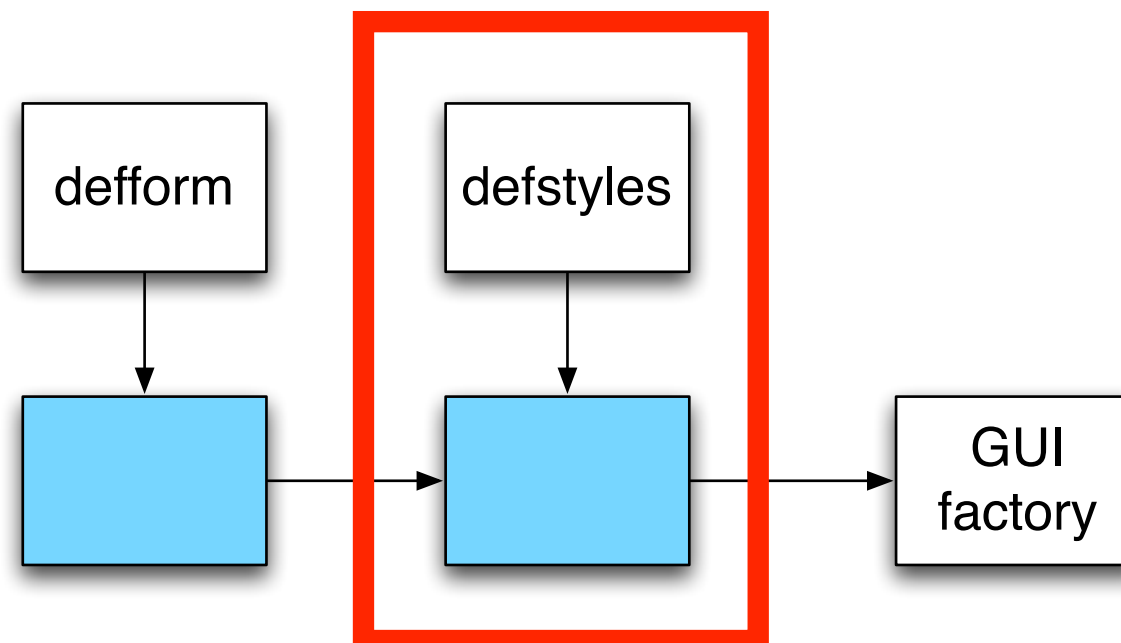
```
(fn [lookup] ((operators '-)
              (lookup 'price)
              (lookup 'debt)))
```

QLS Example

```
(defstyles some-style
  [has-sold-house {:type :radio-group}]
  [value-residue {:label {:foreground "blue"}}]
  [twice          {:widget {:font {:size 20}}}]])
```


QLS Design

- Values are arguments to Seesaw factory functions
- QLS macros are translated to GUI factory wrappers that add additional arguments



- Slight hack: requires passing the element name into the factory, (which it shouldn't need)

Validation

- Entirely “do it yourself!” ☺
- Currently:
 - Existence of variables in expressions
 - Circular dependencies
- Errors at compile time, not “live”

Clojure: The Good

- Defining DSLs is trivial: `defmacro`
 - Pattern matching
- Re-use is trivial: rewrite to other DSL
 - Seesaw
 - `core.match`
- Small: ~400 LoC

Clojure: The Bad

- Dynamically typed, no tool support
 - Should have used Scala ☺
- Quoting Hell
 - Clojure has `:keywords`, should have used those
- Mostly functional, not a great fit with a state-oriented systems such as GUIs

Clojure: The Ugly

```
clojure.lang.Compiler$CompilerException: java.lang.IllegalArgumentException: Don't
know how to create ISeq from: Symbol, compiling:(questionnaire.clj:5)
  at clojure.lang.Compiler.analyze (Compiler.java:6281)
    clojure.lang.Compiler.access$100 (Compiler.java:37)
    clojure.lang.Compiler$LetExpr$Parser.parse (Compiler.java:5883)
    clojure.lang.Compiler.analyzeSeq (Compiler.java:6455)
    clojure.lang.Compiler.analyze (Compiler.java:6262)
    clojure.lang.Compiler.analyzeSeq (Compiler.java:6443)
    clojure.lang.Compiler.analyze (Compiler.java:6262)
    clojure.lang.Compiler.analyze (Compiler.java:6223)
    clojure.lang.Compiler$BodyExpr$Parser.parse (Compiler.java:5618)
    clojure.lang.Compiler$FnMethod.parse (Compiler.java:5054)
    clojure.lang.Compiler$FnExpr.parse (Compiler.java:3674)
    clojure.lang.Compiler.analyzeSeq (Compiler.java:6453)
    clojure.lang.Compiler.analyze (Compiler.java:6262)
    clojure.lang.Compiler.analyzeSeq (Compiler.java:6443)
    clojure.lang.Compiler.analyze (Compiler.java:6262)
    clojure.lang.Compiler.access$100 (Compiler.java:37)
    clojure.lang.Compiler$DefExpr$Parser.parse (Compiler.java:518)
    clojure.lang.Compiler.analyzeSeq (Compiler.java:6455)
    clojure.lang.Compiler.analyze (Compiler.java:6262)
    clojure.lang.Compiler.analyzealyze (Compiler.java:6223)
    clojure.lang.Compiler.eval (Compiler.java:6515)
```



SOURCE OF YOUR TECHNOLOGY

<https://github.com/rix0rrr/lwc-2013-clojure>
rico.huijbers@sioux.eu
@rix0rrr