

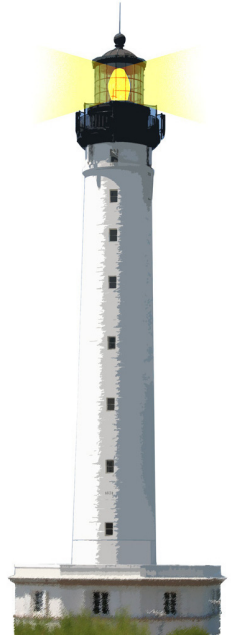
Seaside: Composing Components

Damien Cassou, Stéphane Ducasse and Luc Fabresse

W5S01



<http://www.pharo.org>



Remember: Seaside

- Think in terms of **reusable** and **stateful components**
- A web application = a root component
- Live Debugging
 - through the debugger, you can modify objects and proceed to generate the HTML response

Roadmap

3 mechanisms to reuse components:

- Components aggregation
- The call: / answer: mechanism
- Definition of workflows (Task)



Roadmap

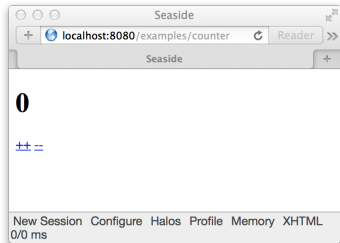
3 mechanisms to reuse components:

- **Components aggregation**
- The call: / answer: mechanism
- Definition of workflows (Task)



Remember: Counter / TwitterCounter

WACounter

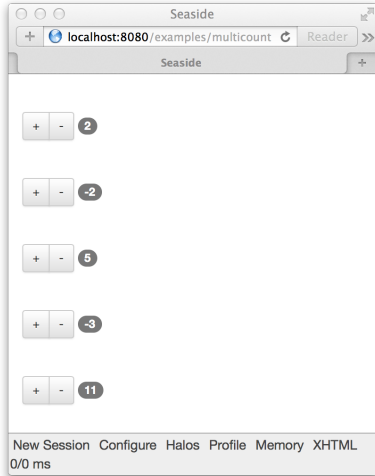


WATwitterCounter



How to Build a Multi-Counter Application?

WAMultiCounter



WAMultiCounter

```
WAMultiCounter subclass: #WAMultiCounter  
instanceVariableNames: 'counters'  
classVariableNames: ''  
package: 'Seaside-Examples-Misc'.
```

```
WAMultiCounter >> initialize  
super initialize.  
counters := (1 to: 5) collect: [ :each | WACounter new ].
```

```
WAMultiCounter >> children  
^ counters
```

```
WAMultiCounter >> renderContentOn: html  
counters do: [ :each | html render: each ]
```

How to Compose Components?

A composite:

- Stores subcomponents in an instance variable (e.g. `children`)
- Sends `render: to subcomponents` in its `renderContentOn: method`
- Redefines `children` to return the collection of its subcomponents



Aggregating Different Subcomponents

```
WExampleComponent subclass: #MyApp  
instanceVariableNames: 'children'  
classVariableNames: ''  
package: 'Seaside-Examples-Misc'.
```

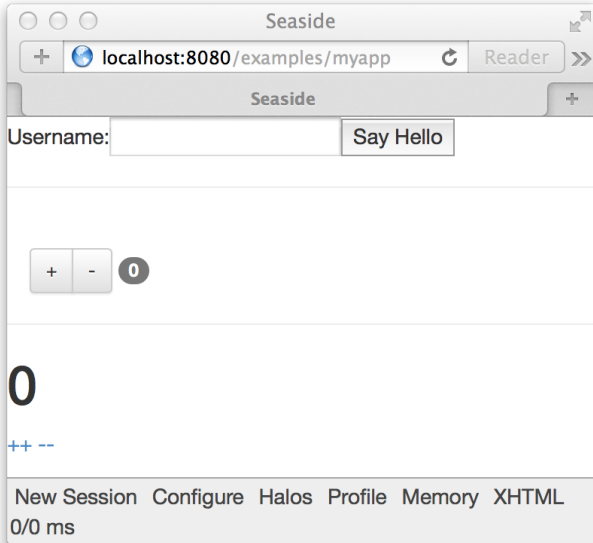
```
MyApp >> initialize  
super initialize.  
children := { Greeter new.  
              WATwitterCounter new. WACounter new }.
```

```
MyApp >> children  
^ children
```

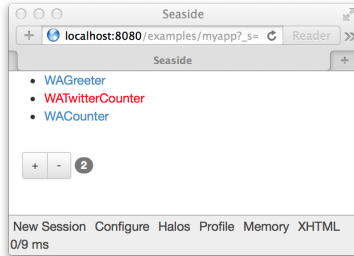
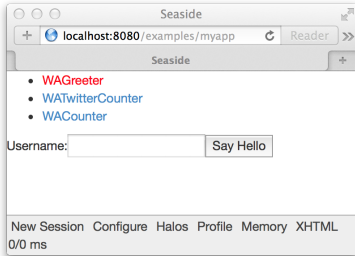
```
MyApp >> renderContentOn: html  
children do: [ :each | html render: each ]  
separatedBy: [ html line ]
```



Aggregating Different Subcomponents



Rendering Only One Subcomponent



Rendering Only One Subcomponent

```
MyApp >> initialize
```

```
"  "
```

```
selectedChild := children first
```

```
MyApp >> renderContentOn: html
```

```
self renderMenuOn: html.
```

```
html render: selectedChild
```

```
MyApp >> renderMenuOn: html
```

```
html unorderedList: [
```

```
self children doWithIndex: [ :child :i |
```

```
html listItem: [
```

```
html anchor
```

```
class: 'selected' if: child = currentComponent ;
```

```
callback: [ selectedChild := self children at: i ];
```

```
with: child className
```

```
]]]
```

Roadmap

3 mechanisms to reuse components:

- Components aggregation
- **The** call: / answer: **mechanism**
- Definition of workflows (Task)



call:

x := A call: B

A

call:

x := A call: B



answer:

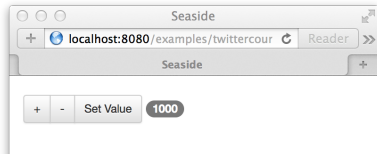
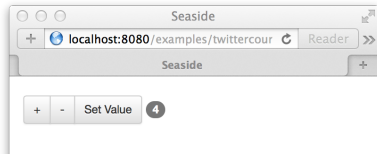


answer:

$x :=$ 

 A

call:/answer: Example



call:/answer: Example

```
WATwitterCounter >> renderContentOn: html
```

```
" ... "
```

```
html tbsButton beDefault;
```

```
  callback: [ self setCountToUserValue ]; with: 'Set Value'
```

```
" ... "
```

```
WATwitterCounter >> setCountToUserValue
```

```
| webDialog newCounterValueByUser |
```

```
webDialog := WAInputDialog new
```

```
  addMessage: 'Enter a new value for the counter:';
```

```
  default: '0';
```

```
  label: 'Ok';
```

```
  yourself.
```

```
newCounterValueByUser := self call: webDialog.
```

```
count := newCounterValueByUser asNumber
```



WInputDialog Internals

WInputDialog

- is a reusable component
- it answer: a result

```
WInputDialog >> renderContentOn: html  
html form  
  defaultAction: [ self answer: value ];  
  with: [  
    html div: [  
      html textInput on: #value of: self.  
      html space.  
      html submitButton  
        callback: [ self answer: value ];  
        text: self label ] ]
```



Roadmap

3 mechanisms to reuse components:

- Components aggregation
- The call: / answer: mechanism
- **Definition of workflows (Task)**



Encapsulating Workflows

Tasks are simple components:

- No UI part (i.e. no `renderContentOn:`)
- Orchestrating several components
 - use `call:/answer:` behind the scene



A Simple Web Task

```
WATask subclass: #Adder  
instanceVariableNames: ""  
classVariableNames: ""  
package: 'SeaExample'
```

```
Adder >> go  
| value1 value2 |  
value1 := self request: 'first number'.  
value2 := self request: 'second number'.  
self inform: value1 asNumber + value2 asNumber
```

```
WAAdmin register: self asApplicationAt: 'Adder'.
```

How request: is Implemented?

request: uses call: / answer: on a WAInputDialog

```
WComponent >> request: aRequestString label: aLabelString  
default: aDefaultString onAnswer: aBlock
```

"Display an input dialog with the question aRequestString,
the button label aLabelString and the default string
aDefaultString. Passes the answer into aBlock."

self

```
call: (WAInputDialog new  
  addMessage: aRequestString;  
  default: aDefaultString;  
  label: aLabelString;  
  yourself)  
onAnswer: aBlock
```



How inform: is Implemented?

inform: uses call: / answer: on a WAFormDialog

WAComponent >> inform: aString onAnswer: aBlock

"Display a dialog with aString to the user until he clicks the ok button. Continue by evaluating aBlock."

self

```
call: (WAFormDialog new
      addMessage: aString;
      yourself)
onAnswer: aBlock
```



Stepping Back

- **No** manual request parsing
- **No** request routing
- **No** hardcoding of next page
- **No** XML configuration files



Conclusion

Seaside provides:

- Stateful components
- Components composition (children)
- Components scheduling using `call: / answer:`
- Tasks to encapsulate workflows



A course by



and



in collaboration with



Inria 2016

Except where otherwise noted, this work is licensed under CC BY-NC-ND 3.0 France

<https://creativecommons.org/licenses/by-nc-nd/3.0/fr/>