5. Seaside

Birds-eye view

Model your domain with objects — model domain components as objects. Compose objects, not text. Strive for fluent interfaces. Build applications by *scripting components*.

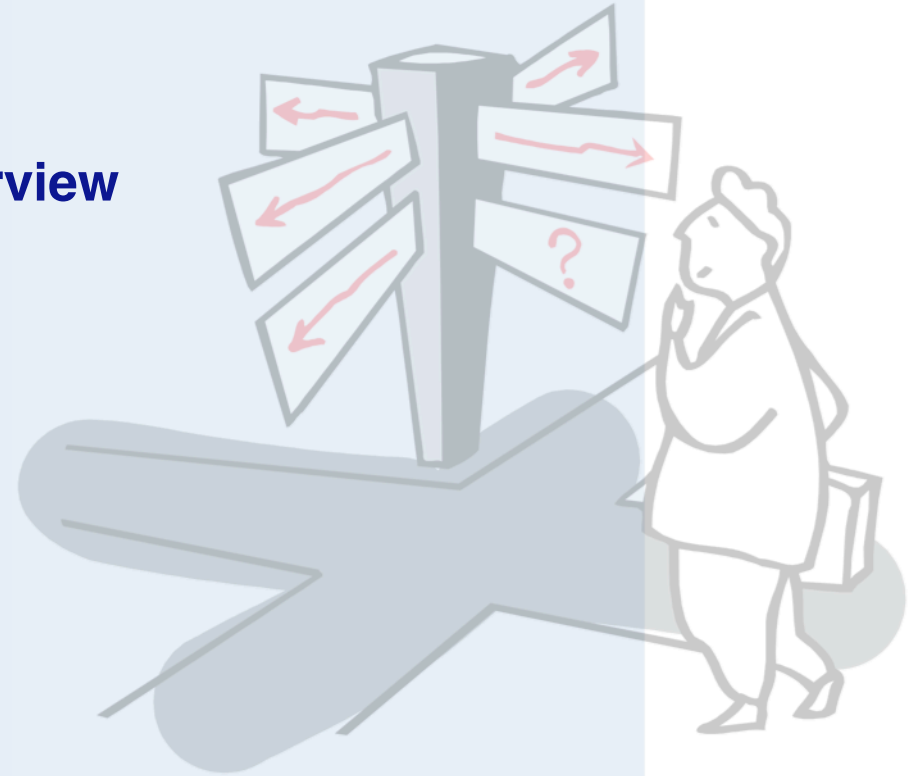
Roadmap

- Introduction
  - Web applications / Overview
  - Installation
- Control Flow
- Components
- Composition

Original lecture notes by Lukas Renggli

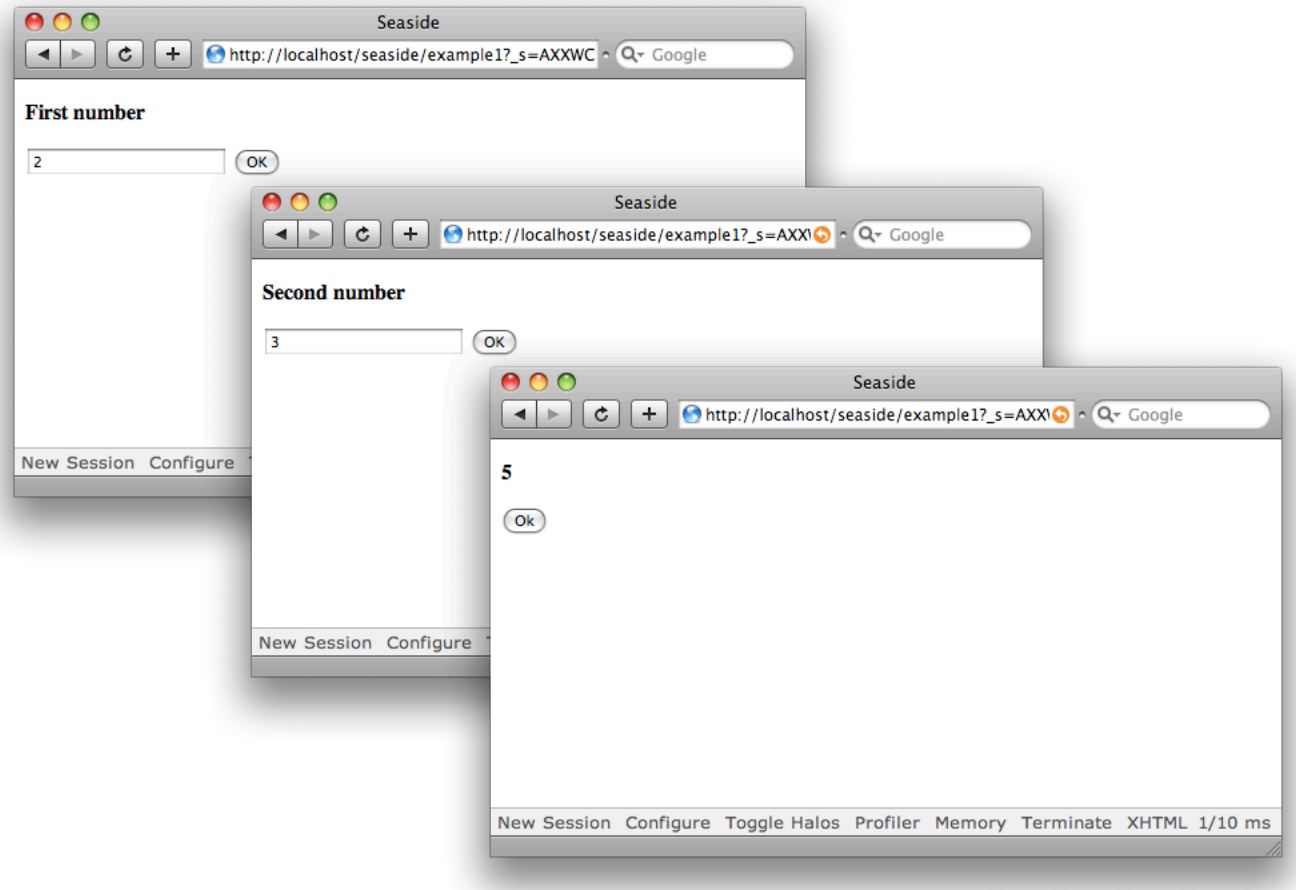
Roadmap

- Introduction
  - **Web applications / Overview**
  - Installation
- Control Flow
- Components
- Composition



Introduction: Web Applications

Example: Adding two numbers



What is going on?

first.html

```
<form action="second.html">
   <input type="text" name="value1">
   <input type="submit" name="OK">
</form>
```

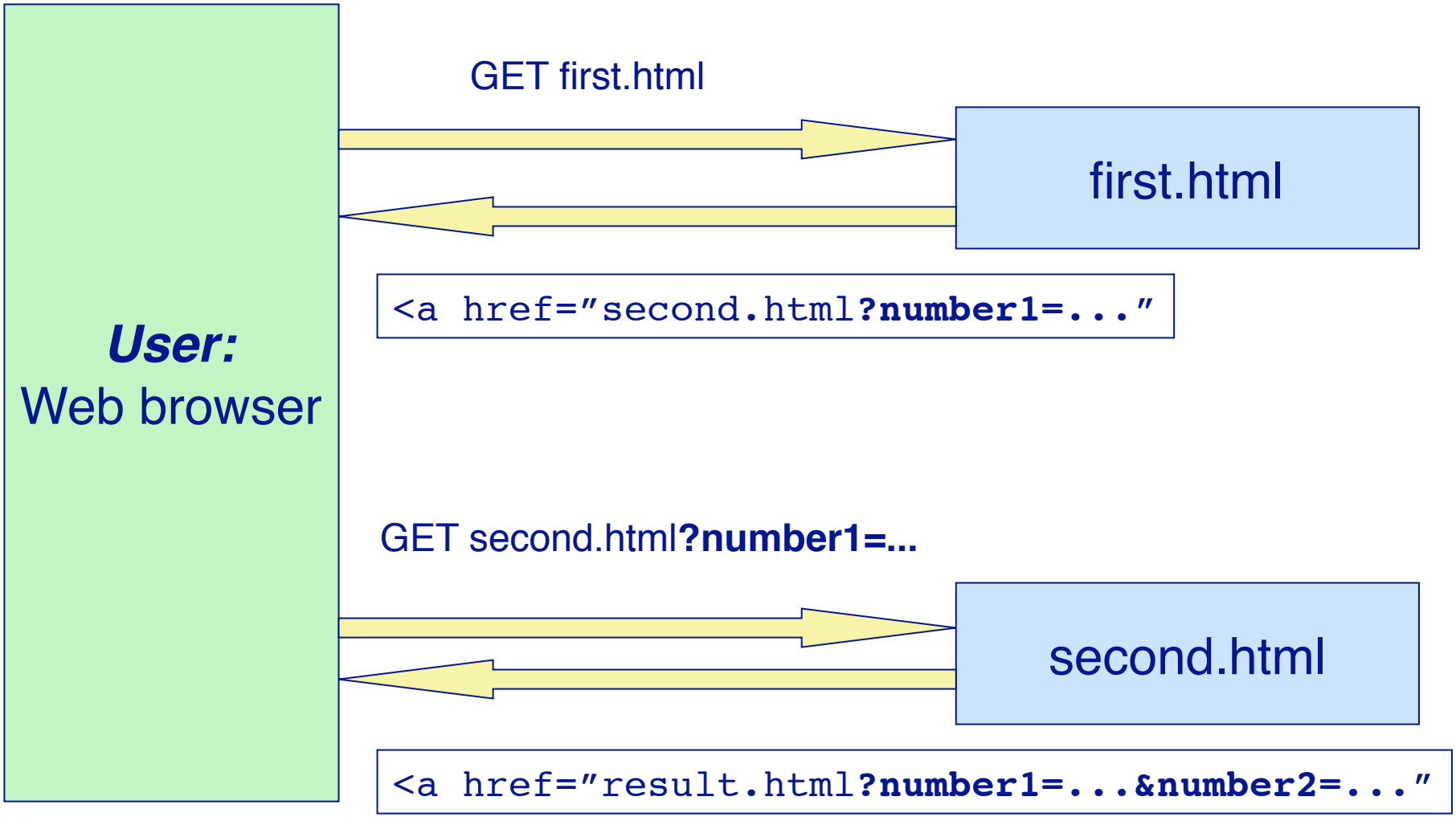
second.html

```
<form action="result.html">
   <input type="hidden"
                  name="value1" value="<% value1 %>">
</form>
```

result.html

```
<p>
   <% value1 + value2 %>
</p>
```

Control Flow: HTTP request-response



Something is wrong…

- Control-flow quite arcane
  - Remember GOTO?
  - We do not care about HTTP!
- How to debug that?
- And what about
  - Back button?
  - Copy of URL (second browser)?

What we want

- Why not this?

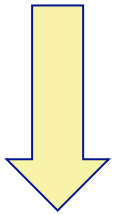
```
go
   |number1 number2 |

   number1 := self request: 'First Number'.
   number2 := self request: 'Second Number'.

   self inform: 'The result is ',
                (number1 + number2) asString
```

Seaside: Features

- Sessions as continuous piece of code
- XHTML/CSS building
- Callback based event model
- Composition and reuse
- Debugging and Development tools

XHTML Building

```
html div id: 'title'; with: 'Title'
html div id: 'list'; with: [
    html span class: 'item'; with: 'Item 1'.
    html span class: 'item'; with: 'Item 2'.
]
```



```
<div id="title">Title</div>
<div if="list">
    <span class="item">Item 1</span>
    <span class="item">Item 2</span>
</div>
```

CSS

- CSS Zengarden: http://csszengarden.com

Callback Event Model

```
Example3>>renderContentOn: html
          html form: [
          html submitButton
              callback: [ self inform: 'Hello' ];
              text: 'Say Hello' ]
```

```
....
<form action="/seaside/example2" method="post">
<input type="hidden" name="_s" value="JBbTXBnPaTLOjcjI" class="hidden"/>
<input type="hidden" name="_k" value="FFQrpnBg" class="hidden" />
<input type="submit" name="1" value="Say Hello" class="submit" />
</form>
....
```

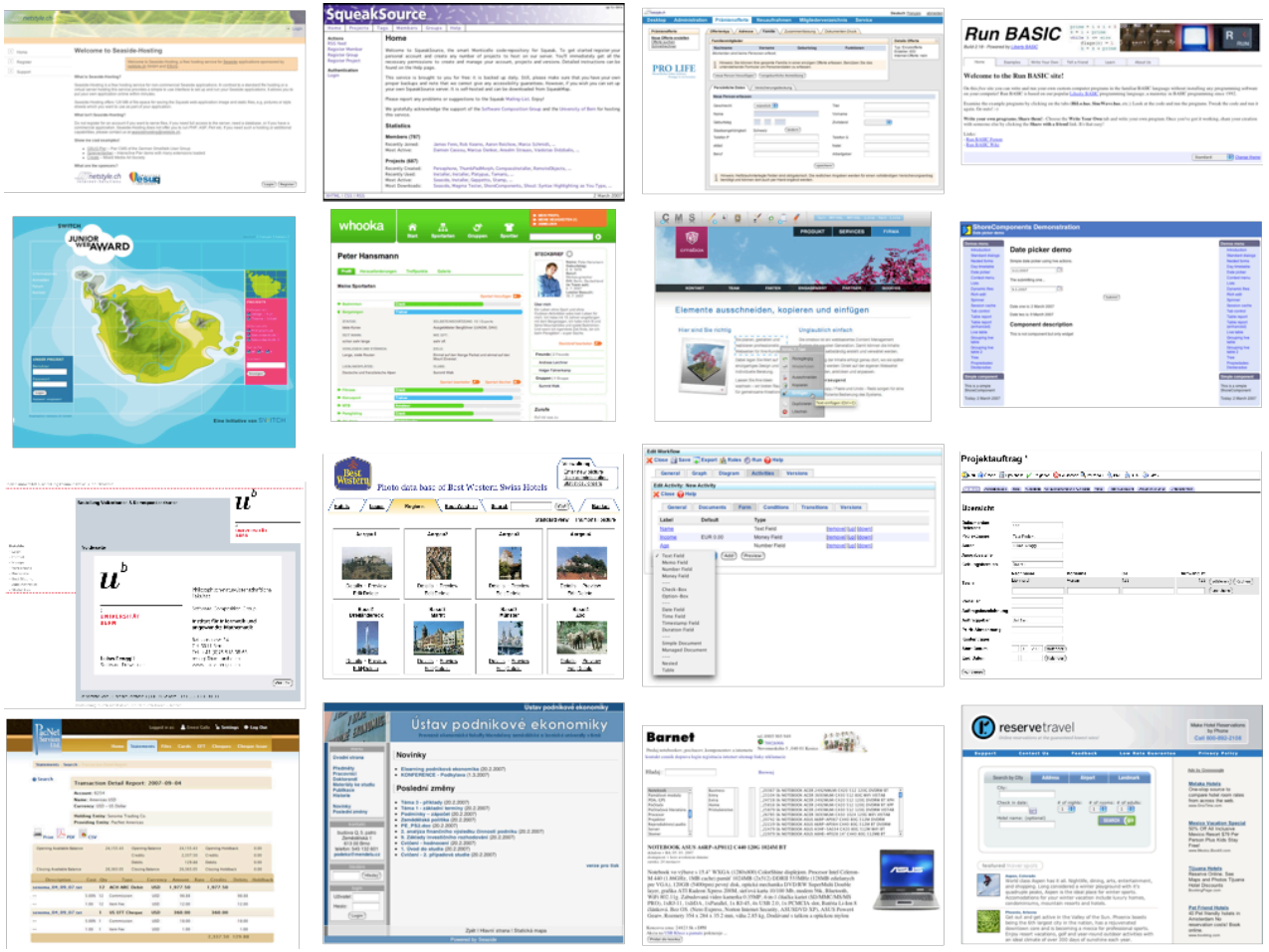
Composition + Reuse

Example:

Multicounter

More later!

Examples


Welcome to Seaside-Hosting
SqueakSource
Run BASIC
Welcome to the Run BASIC site!
whooka
Peter Hansmann
Elemente ausschneiden, kopieren und einfügen
Date picker demo
Ústav podnikové ekonomiky
Barnet
reservetravel

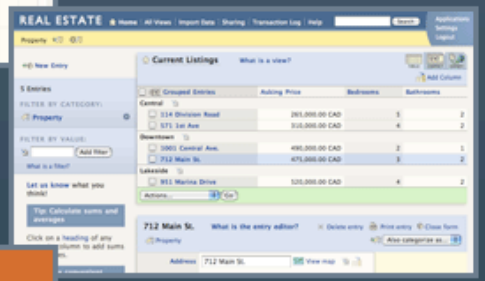
Example: dabbledb.com

Sales inquiries? PHONE: +1 866 779 DATA CHAT: Live chat room EMAIL: info@dabbledb.com

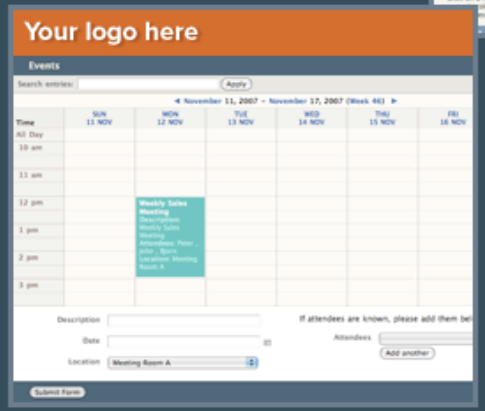
dabble db

Enter your data **once**.
Share it with **a dozen people**.
View it from **a hundred angles**.
The **power** of a database with point-and-click **simplicity**.

Dabble DB helps you create an online database on the web. It's easy to use yet extremely flexible and powerful.



Save multiple views of data from any angle and **re-organize** on a whim. **Share everything with colleagues**.



Collect data easily by creating **online forms** using a **drag-and-drop interface**.

Sign up now

Or find out more…
How you create a database
Free Commons account
Our blog
Contact us

Existing user? Click here to login.

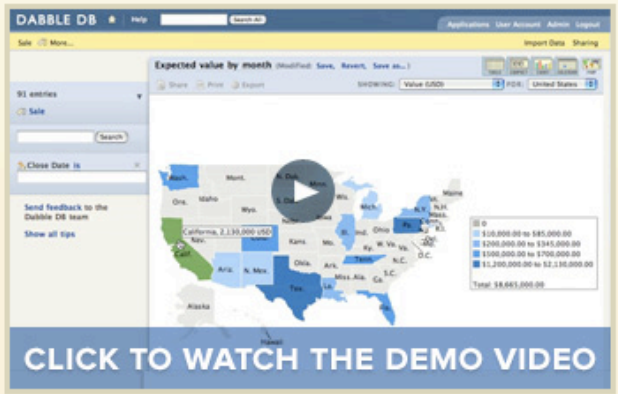
Try Dabble DB right now
Free for 1 month

Pricing starts at $10 per month for private plans.
Free for public data!

Sign up: 1 Month Free Private Trial

No credit card needed. Upgrade to a **paid plan** or to **Dabble DB Commons** at any time.

Not sure? **Watch our 8-minute demo** to see Dabble DB in action.



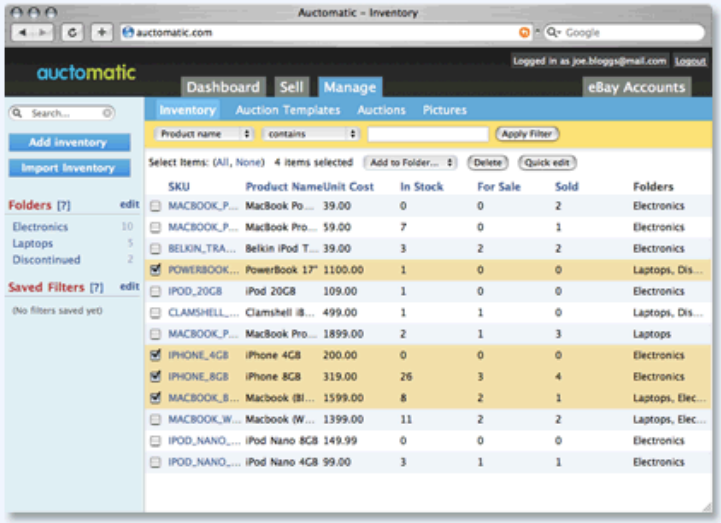
Example: auctomatic.com

auctomatic

Home Signup Blog About Features Support | Login

Selling made easy

Wish you could **sell more** on eBay?
Then start using **auctomatic**.



"Essential to anyone handling high volumes of listings"
— Pete Cashmore, Mashable

Inventory Management
Stay on top of your stock

Auctions
Create great-looking auctions quickly

Pictures
Store and edit your pictures

Take the tour

or

Sign up now

What is Auctomatic?

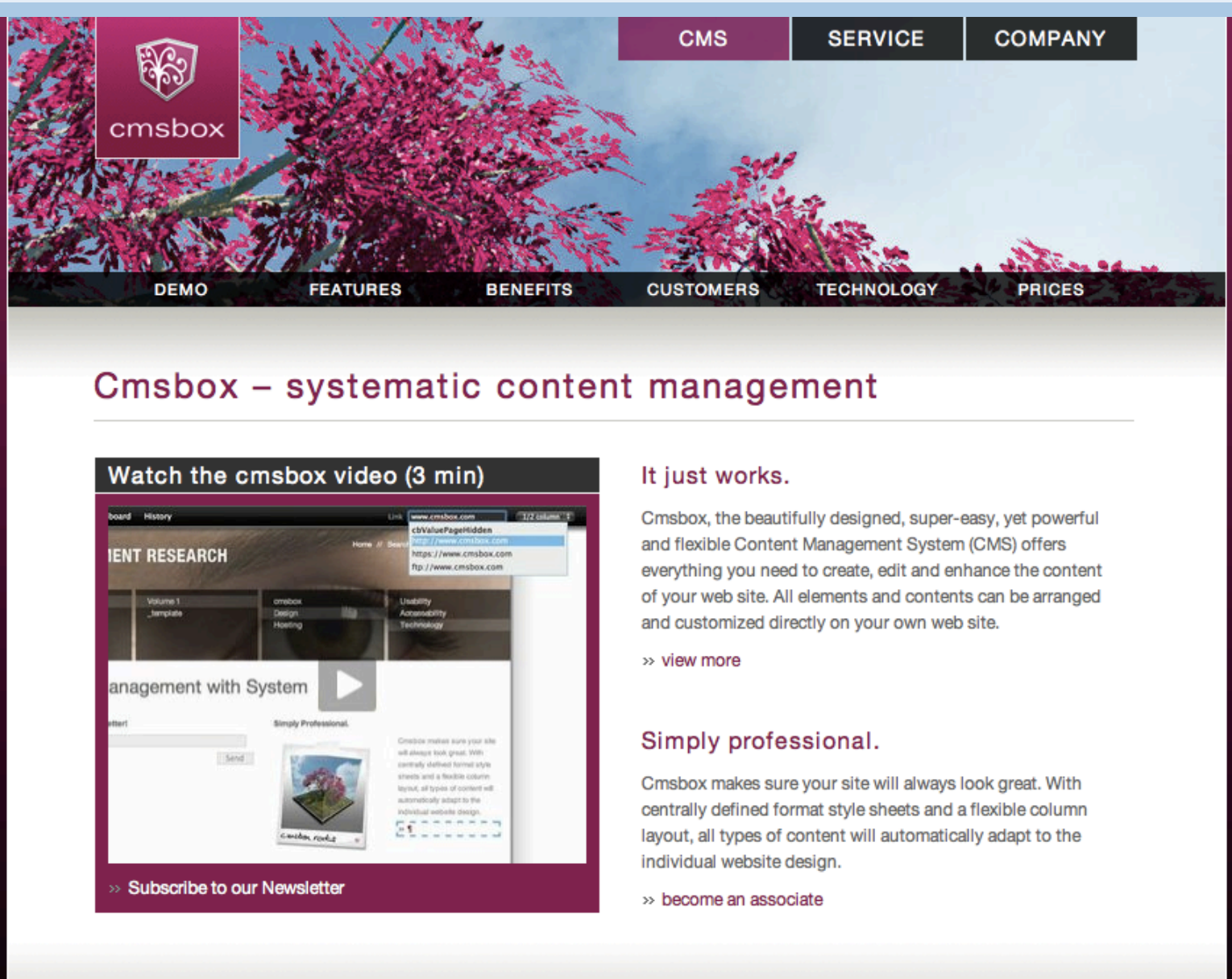
Auctomatic is a tool for managing your eBay business. Use Auctomatic to track your inventory, pictures, and auction templates, and track the traffic on your auctions so you can optimize your listing strategy.

Talk to us...

Submit the features you want us to build for you here.

Give us general feedback by emailing feedback@auctomatic.com.

Example: cmsbox.com



Roadmap

- Introduction
  - Web applications / Overview
  - **Installation**
- Control Flow
- Components
- Composition

Installing Seaside

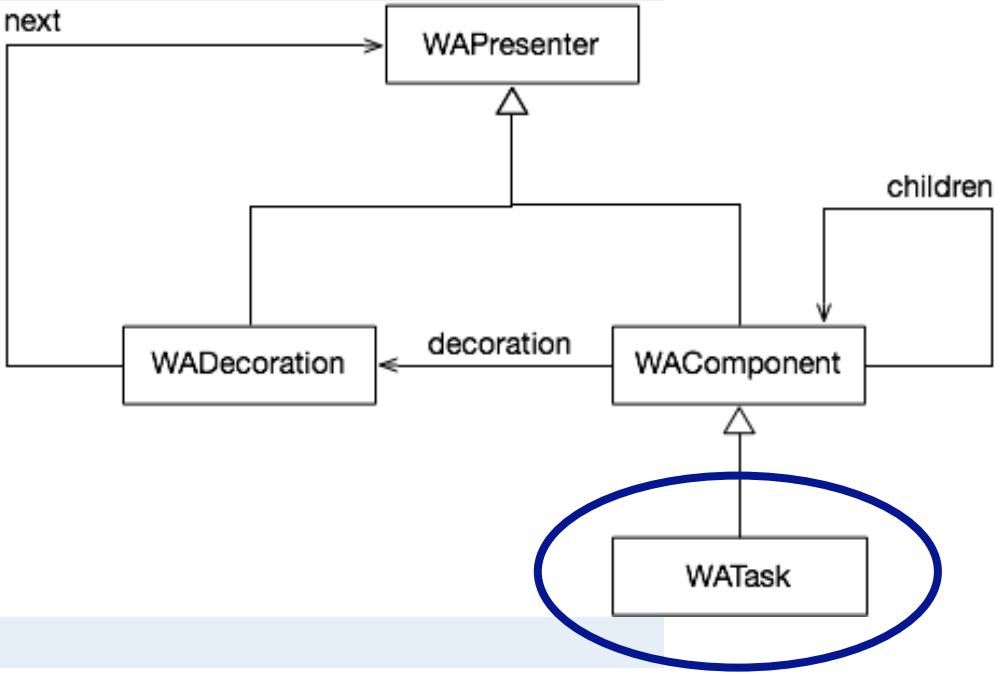
Download the one-click image from http://seaside.st/

Seaside

http://localhost:8080/seaside

Roadmap

- Introduction
  - Web applications / Overview
  - Installation
- **Control Flow**
- Components
- Composition



2. Control Flow

- Defining control flow
- Convenience methods
- Call / Answer
- Transactions

Defining Flow

- Create a subclass of `WATask`
  - Implement the method `#go`
  - Split the method `#go` into smaller parts to ensure readability
- Tasks are a special kind of component
  - No visual representation
  - Define a logical flow (`#go`)
  - Call other components for output

Convenience Methods

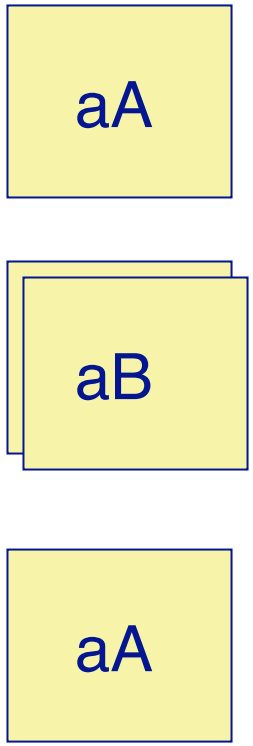
- #inform: aString
- #confirm: aString
- #request: aString
- #request:label:default:
- #chooseFrom:caption:

Call and Answer

- #call: aComponent
  - Transfer control to aComponent
- #answer: anObject
  - anObject will be returned from #call:
  - Receiving component will be removed

Call and Answer

Client



Server

```
A>>go
    x := self call: B new.
    x asString.


B>>go
    …
    self answer: 77.


A>>go
    x := self call: B new.
    x astring.
              -> 77
```

Transactions

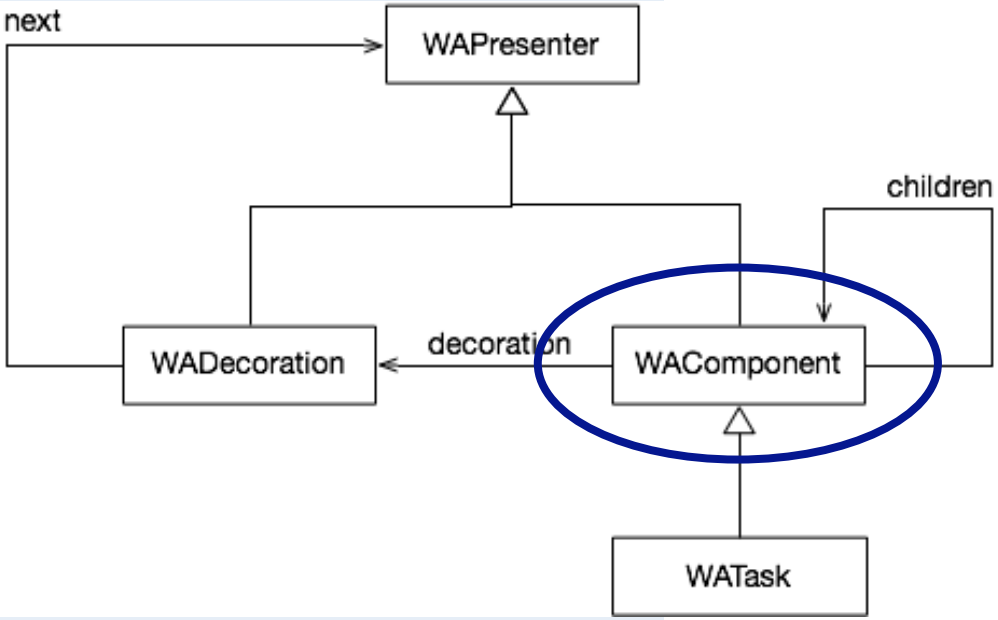
- Sometimes it is required to prevent the user from going back within a flow
- Calling #isolate: treats the flow defined in the block as a transaction
- Users are able to move back and forth within the transaction, but once completed, they cannot go back anymore

Example for #isolate:

```
self isolate: [
   self doShopping.
   self collectPaymentInfo].
Self showConfirmation.
```

Roadmap

- Introduction
  - Web applications / Overview
  - Installation
- Control Flow
- **Components**
- Composition



3. Components

- Rendering
  - XHTML
  - CSS
- Callbacks
  - Anchor
  - Form
- Customization

Components

- Components are the *Views* (and *Controllers*) of Seaside applications.
- Components keep their state (model and state of user-interface) in *instance-variables.*
- Components define the *visual appearance* and handle *user interactions*

Building Components

- Components are created by subclassing `WAComponent`
- Add instance-variables to hold your model and user-interface state
- Put view related methods in a category called *rendering*
- Put controller related methods into method categories named *accessing, actions, private*

Rendering

- XHTML is built programmatically.
- This process is called rendering.

- Create a method called #renderContentOn:

```
SomeComponent>>renderContentOn: html
   html text: 'Hello World'
```

Text Rendering

- Render a string:

```
html text: 'My Text'
```

- Render an un-escaped string:

```
html html: '<foo>Zork</foo>'
```

- Render any object (using double dispatch):

```
html render: 1
```

Canvas and Brushes

- `html` parameter is instance of `WARenderingCanvas`
  - Basic html output
  - Render logic
- Canvas provides brushes
  - For rendering html tags

Basic Brushes

- Render a new line `<br />`:

```
html break.
```

- Render a horizontal Rule `<hr />`:

```
html horizontalRule.
```

- Render a non-breaking space ` `:

```
html space.
```

Using Brushes

1. Ask the canvas for a div brush

```
html div.
```

2. Configure the brush, e.g. set attributes

```
html div class: 'beautiful'.
```

3. Render the contents of the tag-brush:

```
html div
  class: 'beautiful';
  with: 'Hello World'.
```

Painting with Brushes

Seaside

```
html div


html div
  class: 'beautiful'


html div
  class: 'beautiful';
  with: 'Hello World'.
```

XHTML

```
 <div></div>


<div class="beautiful">
</div>


<div class="beautiful">
Hello World
</div>
```

Nesting Brushes

- Render a text in **bold**:

```
html strong with: 'My Text'.
```

- Render a text in *italic*

```
html emphasis with: 'My Text'.
```

- Render a text in **bold** and *italic*:

```
html strong with: [
  html emphasis with: 'My Text'].
```

Nesting Brushes

- To nest brushes use the message `#with:.`
- Always send `#with:` as the *last message* in the configuration cascade.
- The argument of `#with:` is rendered using double-dispatch, therefore any object can be passed as an argument.
- To nest tags, pass a block that renders the elements to nest.

Nesting Brushes

- Render nested divs:

```
html div id: 'frame'; with: [
   html div id: 'contents'; with: …
   html div id: 'sidebar'; with: … ].
```

- Render a list:

```
html orderedList with: [
   html listItem with: …
   html listItem with: … ].
```

Rendering Pitfalls I

- Don’t change the state of the application while rendering, unless you have a really good reason to do so.
- Rendering is a *read-only* phase.
- Don’t put all your rendering code into a single method. Split it into small parts and choose a method name following the pattern `#render*On:`

Rendering Pitfalls II

- Rendering is a *read-ony* phase.
  - Don't send `#renderContentOn:` from your own code, use `#render:` instead.
  - Don't send `#call:` and `#answer:` while rendering
- Always use `#with:` as the last message in the configuration cascade of your brush

Anchor Callback

- Ask the rendering canvas for an anchor and configure it with a callback-block:

```
html anchor
   callback: [self someAction];
   with: 'Some Action'].
```

- The callback-block is cached and will be executed later.

Anchor Example

```
WACounter>>renderContentOn: html
   html heading
        level: 1;
        with: self count.
   html anchor
        callback: [self increase];
        with: '++'.
   html space.
   html anchor
        callback: [self decrease];
        with: '--'.
```

Forms

- Render a form around your form elements:

```
html form: [ … ]
```

- Put the Form elements inside the form:

```
html form: [
   html textInput
      value: text;
      callback: [:value | text := value].
   html submitButton ].
```

More Brushes with Callbacks..

- Text Input / Text Area
- Submit Button
- Check-Box
- Radio Group
- Select List
- File-Upload

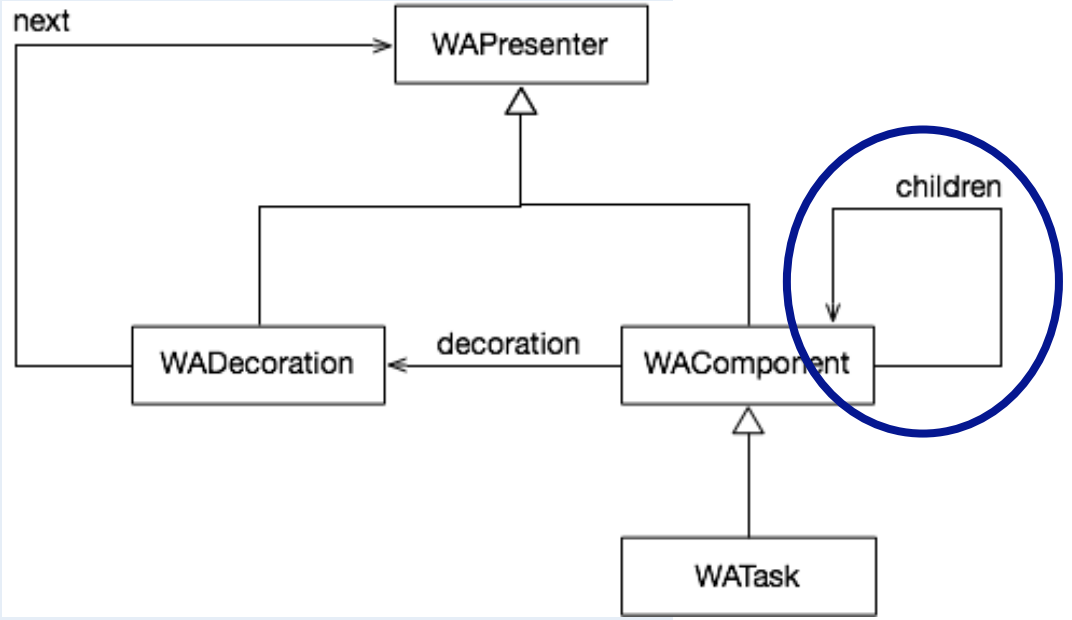
Have a look at the functional tests!

Register new Component / Task

- Create method `#canBeRoot` returning true on class side
- Register using Seaside configuration interface.
- Or call `#registerAsApplication:` in the class-side `#initialize`

Roadmap

- Introduction
  - Web applications / Overview
  - Installation
- Control Flow
- Components
- **Composition**



4. Compositon

- Backtracking
- Subcomponents
- Widgets

Backtracking State

- Seaside does not backtrack state by default.
- Often it is not obvious whether an object should be backtracked or not. Mostly this has to be decided by the developer on a per-object basis.
- Any object can be declared to be backtracked.

Shopping Cart Problem

- Online Bookstore (without backtracking)
  - When using the back-button, usually the items should not be removed from the cart; just resume browsing from the old location.
- Flight Reservation System (with backtracking)
  - When using the back-button, usually you want to check other flights, this means the selected flight should be removed.

Register Object

- Implement method `#states` that returns an Array that contains your object.
- This will backtrack the *instance-variables* of the objects, not the objects themselves

```
SomeComponent>>#states
    ^ Array with: model.
```

Subcomponents

- It is common for a component to display instances of other components.
- Components can be nested into each other using the composite pattern.
- A subcomponent is displayed using the method `#render:` on the canvas.

Initialize Children

- Subcomponents are usually stored within instance variables of the parent component.
- Subcomponents are commonly created lazily or as part of the components `#initialize` method.

```
SomeComponent>>initialize
   super initialize.
   counter := WACounter new.
```

Enable Children

- Parent Components *must* implement a `#children` method returning a collection of subcomponents that they *might* display.
- If you fail to specify `#children` correctly, Seaside will raise an exception.

```
SomeComponent>>children
   ^ Array with: counter
```

Render Children

- Children are rendered by sending the message `#render:` to the rendering canvas.
- Never directly send `#renderContentOn:` to the subcomponent.

```
SomeComponent>>renderContentOn: html
   html heading level: 1; with: 'My Counter'.
   html render: counter.
```

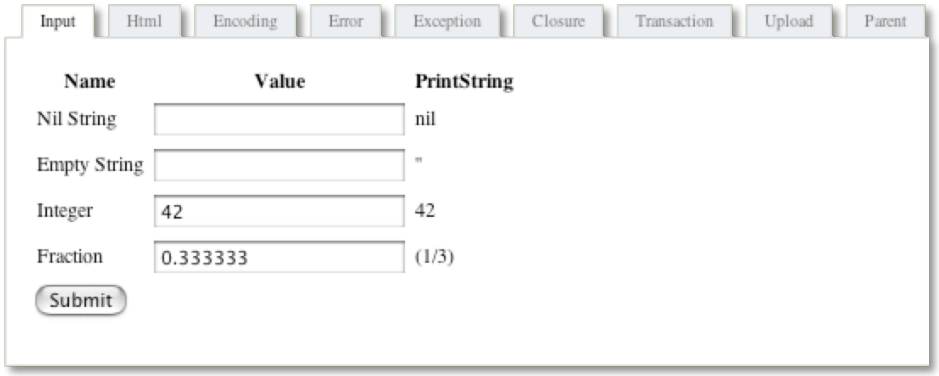
Widgets

- Components can be reused in different contexts within different applications.
- Seaside is shipped with a small collection of widgets ready to use.
- Load and use widgets that have been developed by the Seaside community.
- Write your own widgets that exactly fit your needs

Widgets: Examples

Batched List

Tab Panel



Calendar

Have a look at the classes in Seaside-Components-Widgets

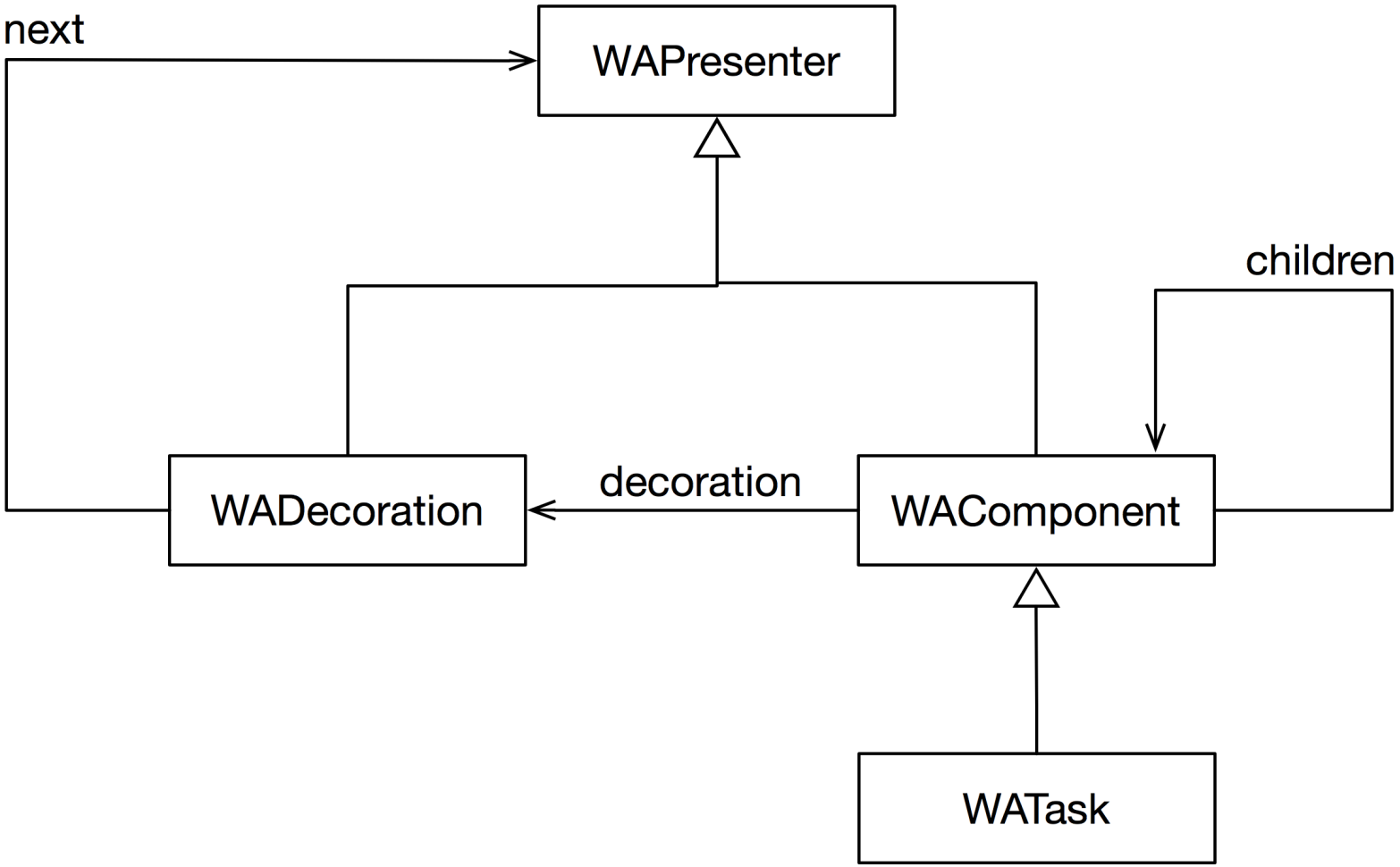
Custom Widgets

- Create a new component.
- Add methods to specify domain-model, subcomponents, properties…
- Assign CSS names/classes to make it skinnable with css style-sheet.
  - Implement method `#style` to return CSS for component
- Write tests and small example applications.

There is more..

- Development Tools
  - Demo in the Exercise Session (Halo, Configuration…)
  - Debugging: Next Lecture
- Javascript: Prototype, script.aculo.us, jQuery, jQueryUI
- Persistency (Databases)

Summary


next
WAPresenter
children
WADecoration
decoration
WAComponent
WATask

Literature

- Dynamic Web Development with Seaside
  - http://book.seaside.st/book
- HPI Seaside Tutorial:
  - http://www.swa.hpi.uni-potsdam.de/seaside/tutorial
- Articles:
  - “Seaside — a Multiple Control Flow Web Application Framework.”
  - “Seaside: A Flexible Environment for Building Dynamic Web Applications”
  - http://scg.unibe.ch/scgbib?query=seaside-article
- …. more at http://seaside.st

License

http://creativecommons.org/licenses/by-sa/3.0/