Programming with Seaside

Alexandre Bergel
Alexandre.Bergel@cs.tcd.ie

LERO & DSG
Trinity College Dublin, Ireland

Part I:
Seaside in a Nutshell

Outline

1. What is Seaside?
2. Starting Seaside
3. Create new Seaside Component
4. Creating GUI
5. Using CSS
6. Interaction Between Components

Introduction to Seaside

- Application server Framework
- Useful for generating dynamic web pages

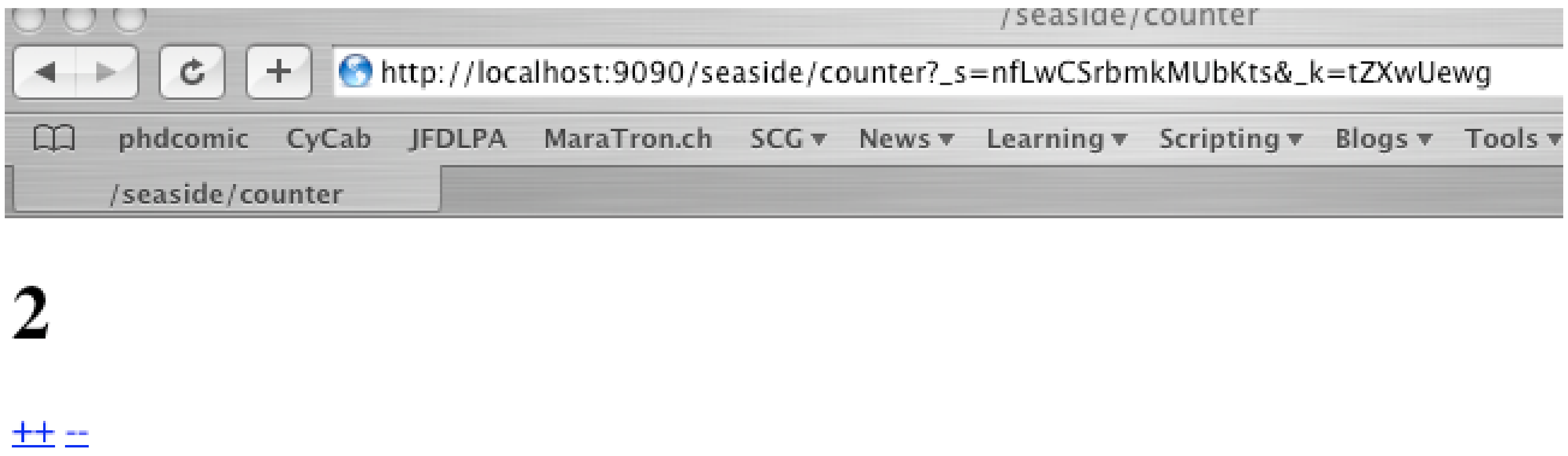
- Web server application for Squeak (used in this presentation) and VisualWorks.
- Works on the top of a webserver (Comanche, Swazoo).
- Provides high-level API to handle navigation between pages (links) and GUI.

Some of the Seaside Features

- Sessions as continuous piece of code
- XHTML/CSS building
- Callback based event-model
- Composition and Reuse
- Development tools
- Interactive debugging
- Multiple control flow

Starting Seaside

- Start the server with:
  WAKom startOn: 9090
- Go to to access the counter component:
  http://localhost:9090/seaside/counter



2

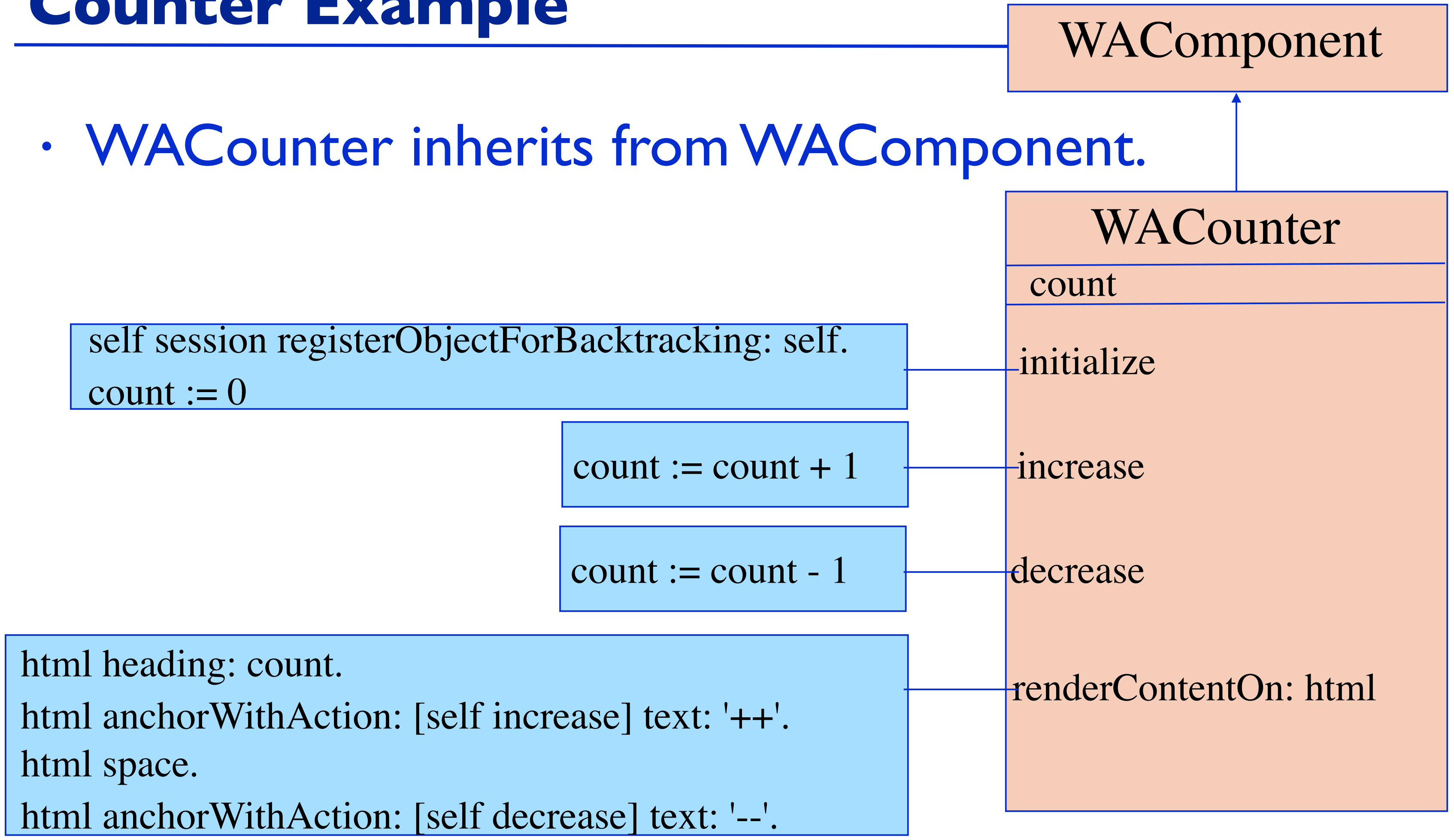
++ --

Component Responsibilities

- It is a subclass of WAComponent
- It contains a State modeled as instance variables
- The flow is defined by methods
- Rendering (high-level API that generate XHTML)
- Style (CSS)

Counter Example

- WACounter inherits from WAComponent.



```
self session registerObjectForBacktracking: self.
count := 0
```

```
count := count + 1
```

```
count := count - 1
```

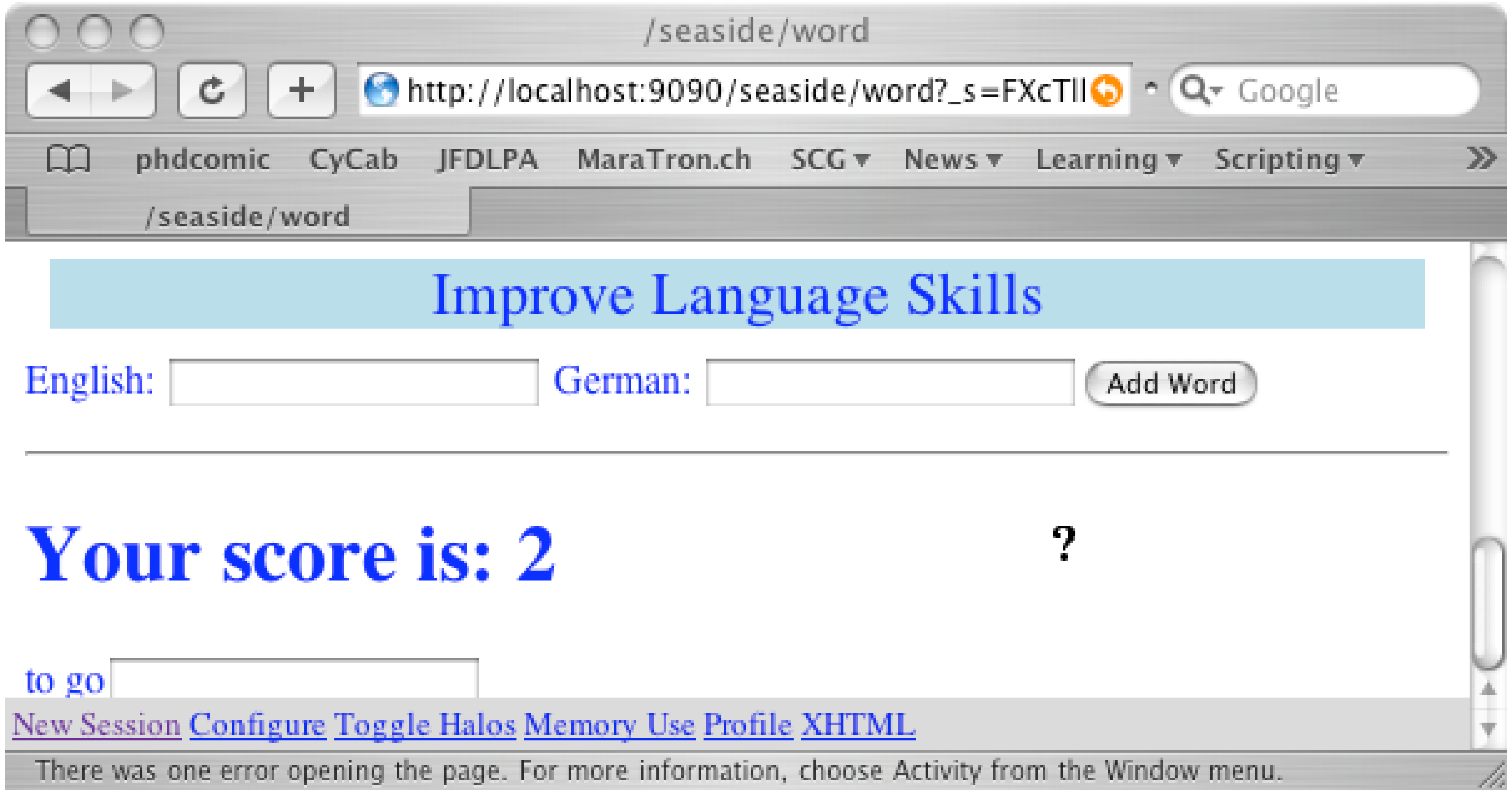
```
html heading: count.
html anchorWithAction: [self increase] text: '++'.
html space.
html anchorWithAction: [self decrease] text: '--'.
```

```
WACounter class>>initialize
        self registerAsApplication: ‘counter’
```

Creating new Component

- Designing a small application to memorize words in a foreign language.
- Display a score to show the progress.
- 2 ways of using:
  - Adding a new word in the database
  - Entering a translation

Creating new Component



Improve Language Skills

English: German: Add Word

Your score is: 2 ?

to go

New Session Configure Toggle Halos Memory Use Profile XHTML

There was one error opening the page. For more information, choose Activity from the Window menu.

Component Definition

- Definition of the main class:

```
WAComponent subclass: #Learner
  instanceVariableNames: 'words germanWord englishWord
score'
  classVariableNames: ''
  poolDictionaries: ''
  category: 'WordLearning'
```

Variables Initialization

- List of entered words:

```
Learner>>words
  words ifNil: [words := OrderedCollection new].
  ^ words
```

- Score (increased when an entered word is correct):

```
Learner>>score
  score ifNil: [score := 0].
  ^ score
```

- Choose a word:

```
Learner>>chooseEntry
   ^ self words atRandom
```

Helper Methods

- Could we ask for a word?

```
Learner>>readyToGuessWord
  ^ self words notEmpty
```

- Increasing the score:

```
Learner>> increaseScore
  score := self score + 1
```

Managing the Back Button

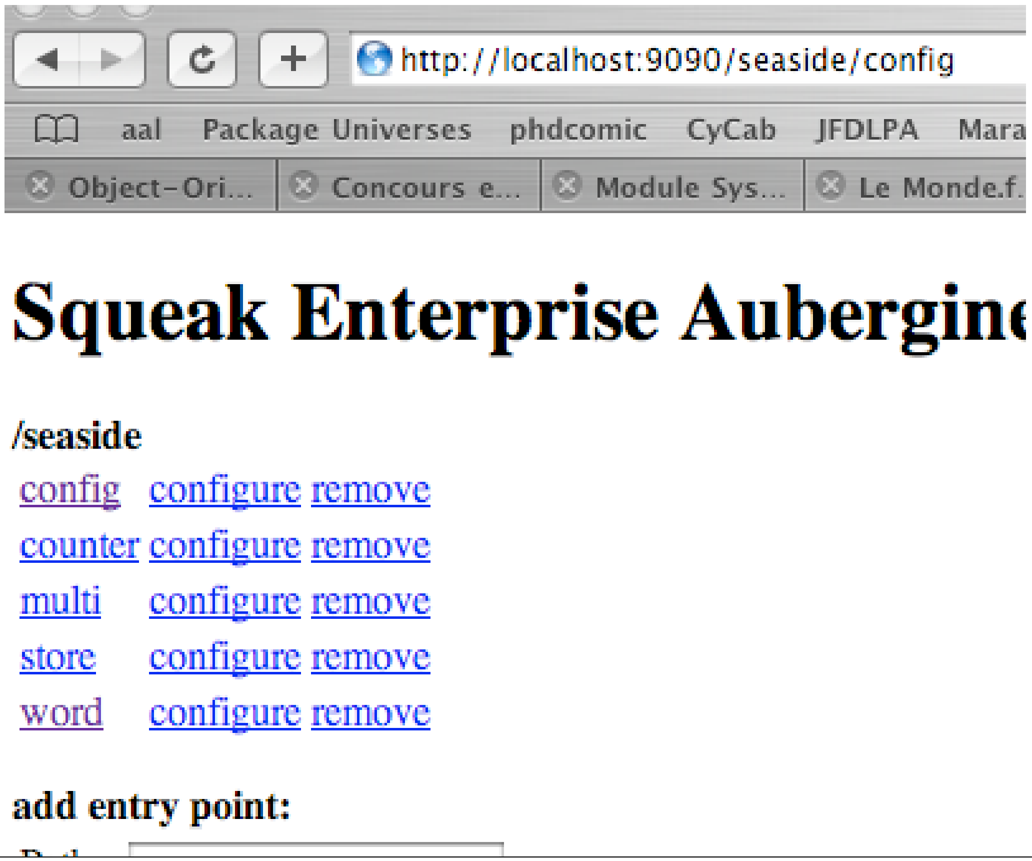
- Need to keep the history of the objects, in case of pressing the back button on the web browser

```
Learner>>initialize
  super initialize.
  self session registerObjectForBacktracking: self.
```

- A trace of the lifetime is kept. When the back button is pressed, state previously recorded is restored.

Registration of the Application

- Application registration:
```
Learner class>>initialize
  self registerAsApplication: 'word'
```



Squeak Enterprise Aubergin

/seaside

config	configure	remove
counter	configure	remove
multi	configure	remove
store	configure	remove
word	configure	remove

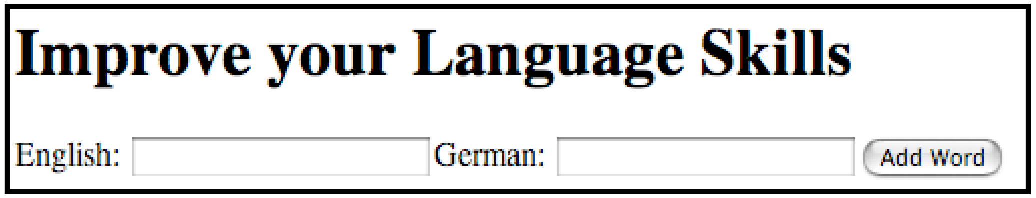
add entry point:

Alexandre Bergel

Rendering (1/2)

- Learner>>renderContentOn: html

```
html heading: 'Improve your Language Skills'.
html form:  [
 html text: 'English: '.
 html textInputWithCallback: [:w| englishWord := w].
 html text: ' German: '.
 html textInputWithCallback: [:w| germanWord := w].
 html submitButtonWithAction:
    [self words add: (Array with: englishWord with: germanWord)]
        text: 'Add Word'.
].
...
```

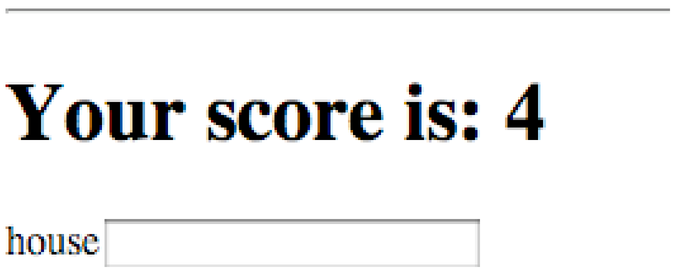


Improve your Language Skills

English: German:

Rendering (2/2)

- ...
```
html horizontalRule.
self readyToChooseWord ifTrue: [
 html heading: 'Your score is: ', self score asString.
 html form: [ |chosenWord|
   chosenWord := self chooseEntry.
   html text: (chosenWord first).
   html textInputWithCallback:
      [:w| (w = chosenWord second) ifTrue:
            [self increaseScore]]]]
```


Your score is: 4

house

Creating GUI (1/2)

- Displaying simple text:
  html text: 'My Text'
- Using different size:
  html heading: aBlockOrText level: level
  html heading: aBlockOrString
- Link with action:
  html anchorWithAction: aBlock text: aString
- TextField without any button:
  html form: [... html textInputWithCallback: aBlock ...]

Creating GUI (2/2)

- Using a form:

```
html form: [
    html textInputWithCallback: aBlock.
    ...
    html submitButtonWithAction: aBlock text: aString]
```

- Look at the class WAHtmlRenderer and WAAbstractHtmlBuilder

CSS: to give a better look

- Use divNamed: aString with: aBlockOrObject

```
html divNamed: 'title' with: [
    html text: 'Improve Language Skills'
].
```

- Or

```
html divNamed: 'title' with: 'Improve Language Skills'
```

CSS: defining the style

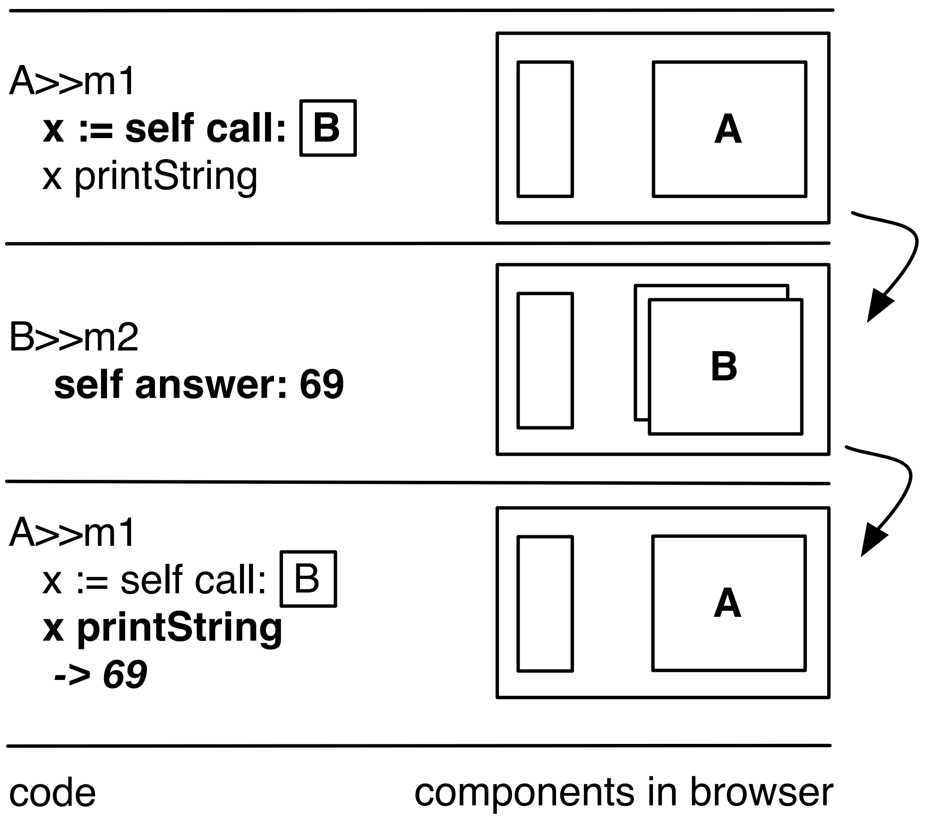
- Define a method named **style** on the seaside component:

```
WordLearningComponent>>style
 ^ '#title {
 background-color: lightblue;
 margin: 10px;
 text-align: center;
 color: blue;
 font-size: 18pt;
 margin-top: 400px}
body {
 background-image: url("http://www.iam.unibe.ch/~bergel/
catsEye_hst_full.jpg");
 background-repeat: no-repeat;
 background-position: top center;
 color: blue;}'
```

CSS: more info

- Supported by many web browsers
- Where to get more information:
  http://www.w3schools.com/css
- ZenGarden:
  http://www.csszengarden.com/

call: / answer:



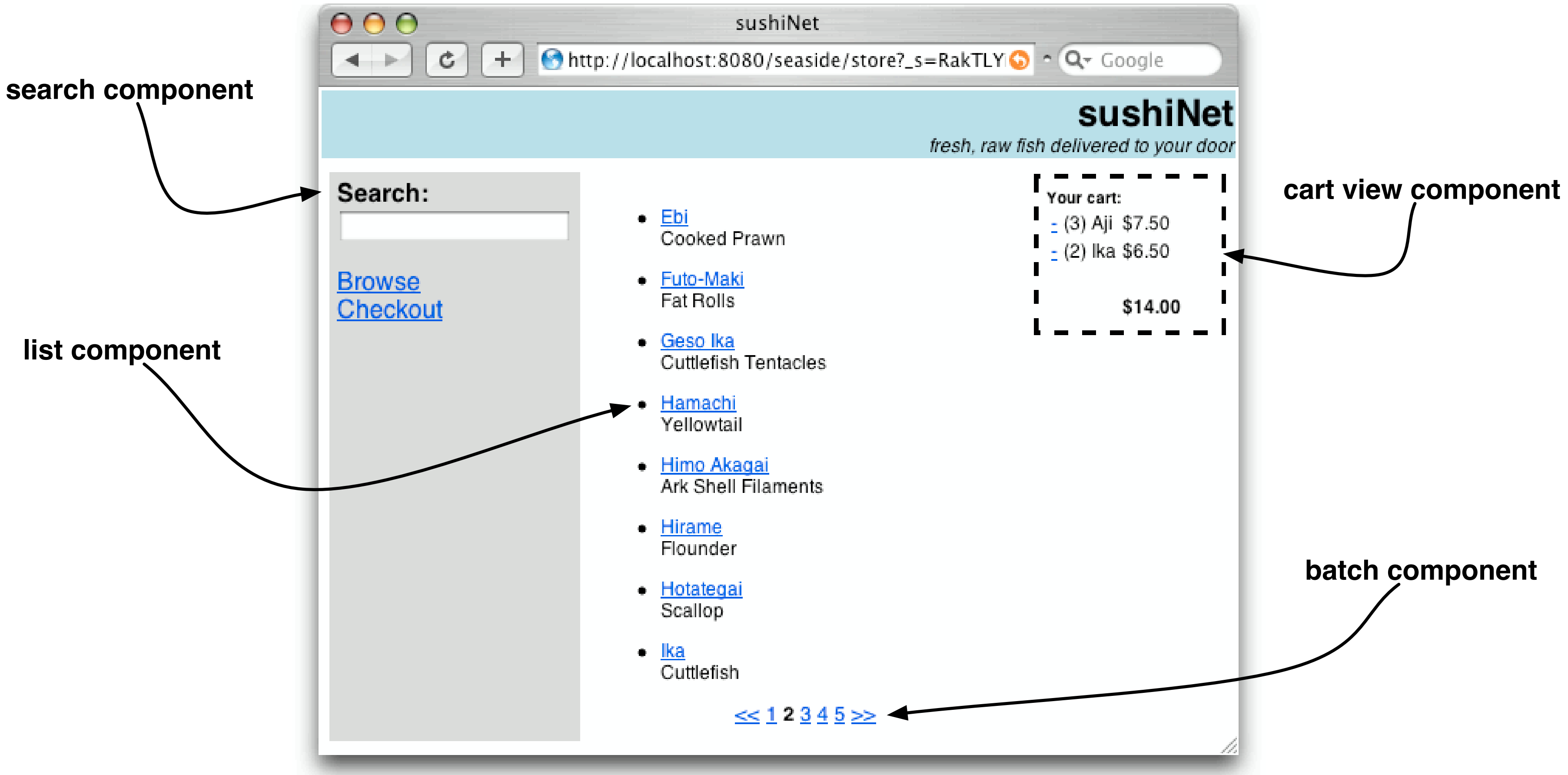
The framed B in the method m1 is a graphical object displayed as the window B in the web browser. m2 is a method that is invoked in a callback i.e., when an action on the component B is invoked such as a button pressed or a link clicked.

call: / answer:

- To transfer control to another component, WAComponent provides the special method #call:. This method takes a component as a parameter, and will immediately begin that component's response loop, displaying it to the user.

- If a called component provides an argument to #answer:, that argument will be returned from #call:. In other words, calling a component can yield a result.

Example: Sushi Shop Online



sushiNet
fresh, raw fish delivered to your door

Search:

Browse
Checkout

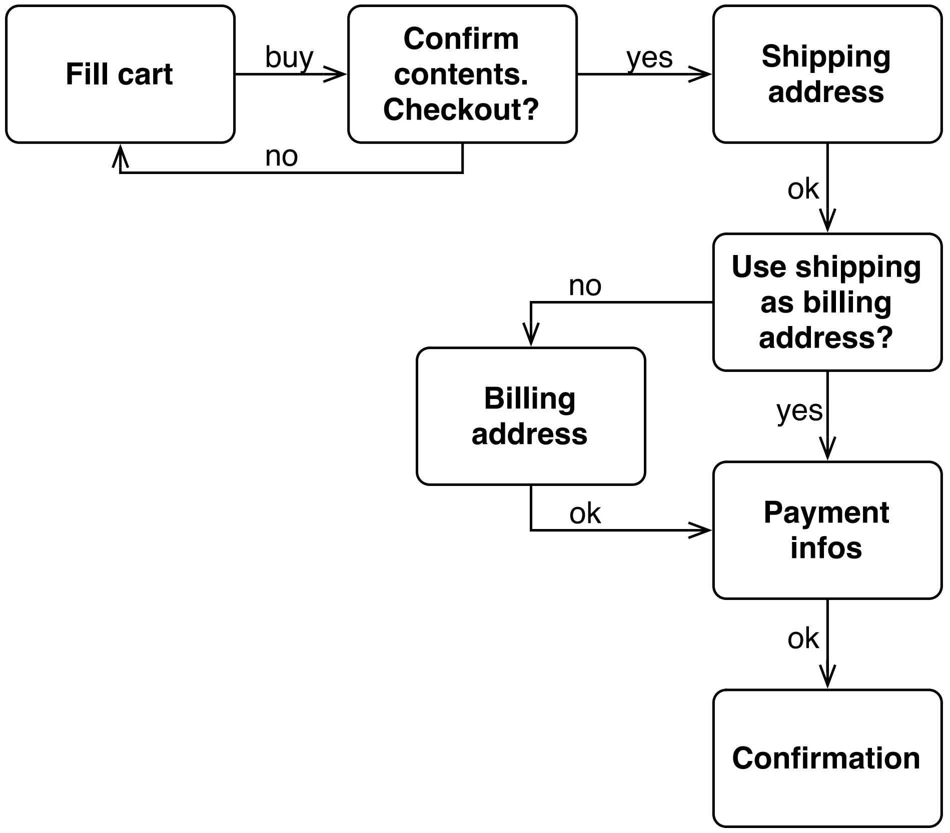
- Ebi
  Cooked Prawn
- Futo-Maki
  Fat Rolls
- Geso Ika
  Cuttlefish Tentacles
- Hamachi
  Yellowtail
- Himo Akagai
  Ark Shell Filaments
- Hirame
  Flounder
- Hotategai
  Scallop
- Ika
  Cuttlefish

<< 1 2 3 4 5 >>

Your cart:
- (3) Aji $7.50
- (2) Ika $6.50

$14.00

Logical Flow



XHTML generation

- XHTML code is generated programmatically:

```
Store>>renderContentOn: html
 html cssId: 'banner'.
 html table: [
         html tableRowWith: [
                 html divNamed: 'title' with: self title.
                 html divNamed: 'subtitle' with: self subtitle.
         ]
 ].
 html divNamed: 'body' with: task
```

Control Flow

```
WAStoreTask>>go
 | shipping billing creditCard |
 cart := WAStoreCart new.
 self isolate:
    [[self fillCart. self confirmContentsOfCart] whileFalse].
 self isolate:
    [shipping := self getShippingAddress.
     billing := (self useAsBillingAddress: shipping)
                                        ifFalse: [self getBillingAddress]
                                        ifTrue: [shipping].
           creditCard := self getPaymentInfo.
           self shipTo: shipping billTo: billing payWith:
creditCard].
 self displayConfirmation.
```

Control Flow

- To fill in the cart:
```
WAStore>>fillCart
    self call: (WAStoreFillCart new cart: cart)
```
- To confirm contents of cart:
```
WAStoreTask>>confirmContentsOfCart
     ^ self call:
          ((WAStoreCartConfirmation new cart: cart)
                    addMessage: 'Please verify your order:')
```
- Payment:
```
WAStore>>getPaymentInfo
  ^ self call:
    ((WAStorePaymentEditor new
       validateWith: [:p | p validate])
  addMessage: 'Please enter your payment information:')
```

Control Flow

- answer returns the component itself

```
WAStoreFillCart>>checkout
    self answer
```

Some Guidelines

- Tasks are used to embed the logical flow of an application within the go method, whereas
- The rendering is in charge of components.
- Hence, the entry point of an application should be a task's go method

Seaside

- Used in industries
- More info on:
  http://www.beta4.com/seaside2
- Seaside's fathers: Avi Bryant and Julian Fitzell
- Mailing list:
  http://lists.squeakfoundation.org/listinfo/seaside

Part II: Developing Web-based Applications

Outline

1. What is a Web-based Application?
2. Issues when Directly Dealing with HTML
3. Example: Sushi Shop Online
4. Seaside Approach
5. Manipulating Non-Linear Control Flow
6. Development Tools

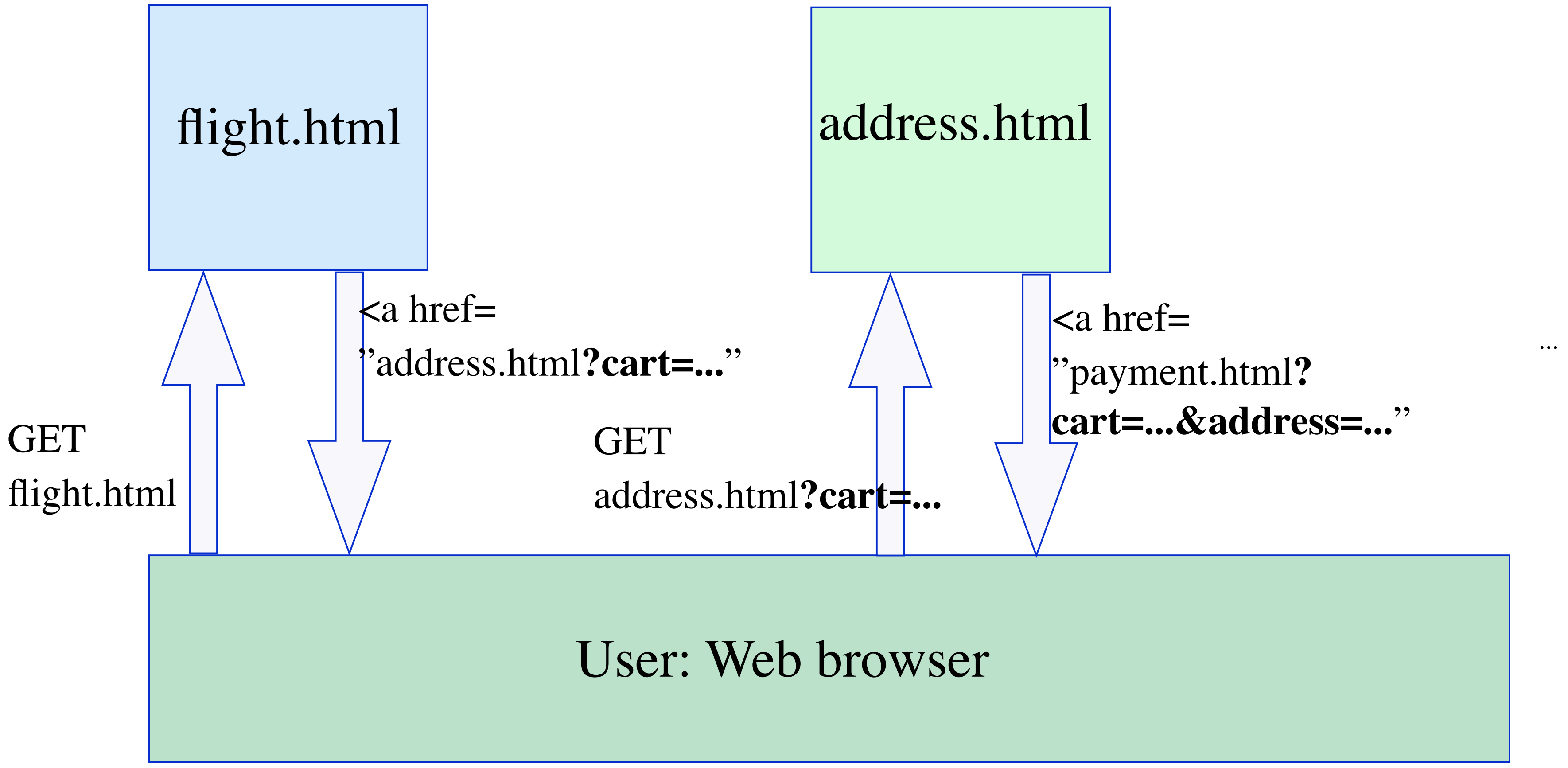
What is a Web-based Application?

- A collection of functions that take HTTP requests as input and produce HTTP responses as output.
- Logical part centralized

Directly Manipulating HTML

- Stateless connection toward the server. State has to be passed around for each connection.
- ASP, PHP

What is a Web-based Application?



Directly Manipulating HTML

- Applications are difficult to maintain:
  - Adding, renaming, removing some state is difficult
  - Flow execution scattered in several files
  - Poor management of the bandwidth: state has to be passed for each action!

Common Issues with Classical Framework

- Applications are often tedious to use:
  - Do not use the back button!
  - Do not duplicate the windows!
  - “Press OK only once!!!”
  - “Do you want to resend the form?”
  - Cookies manipulations

Seaside Approach

- Each session has one unique ID kept over its life time:
  - Users (web browsers windows) are distinguished
- Each action has one ID unique over the session:
  - In the lifetime of a session, an action is unique (”press OK only once”)

Non-Linear Control Flow

- The control flow of an application can always be modified by the user when pressing the back button or by opening a new browser on the same url.

Backtracking State

- With seaside, an object can be backtracked using the method:
  WASession>>registerObjectForBacktracking: anObject
- After each response sent to the client, Seaside snapshots the registered objects by creating a copy and putting them into a cache.
- Pressing the back button on the browser restores the state of the object which is in sync of the display.

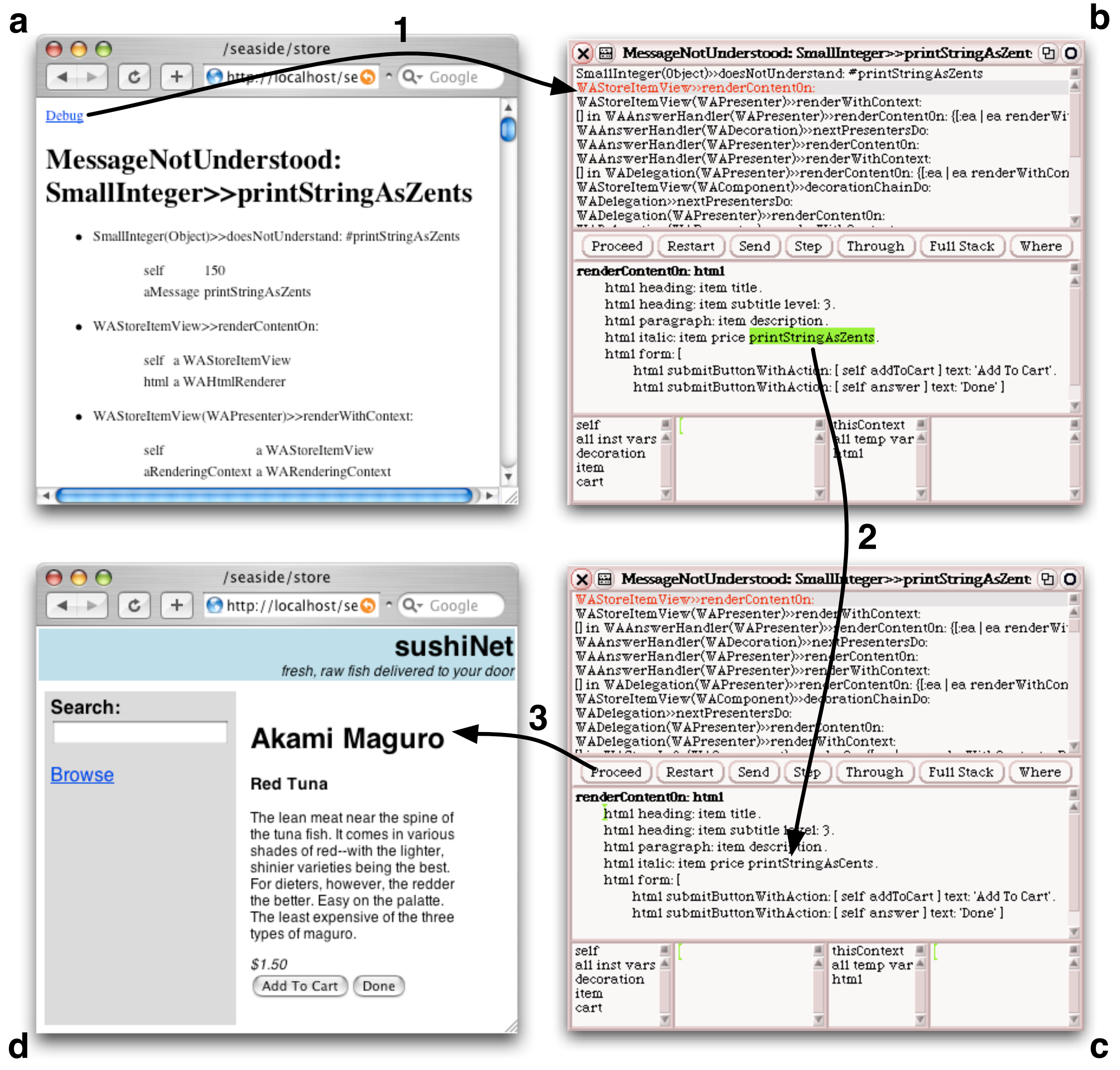
Transaction

- In complex applications it is often the case that we must ensure that the user is prevented from going back over a sequence of pages to make modifications.
- Controlling the control flow is implemented by the method:
  Component>>isolate: aBlock
- It treats the control flow defined in the block as a transaction. It makes sure that the user can move forward and backward within the transaction. Once completed, the user cannot go back anymore.

Debugging with Seaside

- When debugged, an application does not need to be restarted or manually recompiled

Debugging



Debug

MessageNotUnderstood: SmallInteger>>printStringAsZents

- SmallInteger(Object)>>doesNotUnderstand: #printStringAsZents

  self 150
  aMessage printStringAsZents

- WAStoreItemView>>renderContentOn:

  self a WAStoreItemView
  html a WAHtmlRenderer

- WAStoreItemView(WAPresenter)>>renderWithContext:

  self a WAStoreItemView
  aRenderingContext a WARenderingContext

MessageNotUnderstood: SmallInteger>>printStringAsZent

```
SmallInteger(Object)>>doesNotUnderstand: #printStringAsZents
WAStoreItemView>>renderContentOn:
WAStoreItemView(WAPresenter)>>renderWithContext:
[] in WAAnswerHandler(WAPresenter)>>renderContentOn: {[:ea | ea renderWi
WAAnswerHandler(WADecoration)>>nextPresentersDo:
WAAnswerHandler(WAPresenter)>>renderContentOn:
WAAnswerHandler(WAPresenter)>>renderWithContext:
[] in WADelegation(WAPresenter)>>renderContentOn: {[:ea | ea renderWithCon
WAStoreItemView(WAComponent)>>decorationChainDo:
WADelegation>>nextPresentersDo:
WADelegation(WAPresenter)>>renderContentOn:
```

Proceed | Restart | Send | Step | Through | Full Stack | Where

```
renderContentOn: html
    html heading: item title.
    html heading: item subtitle level: 3.
    html paragraph: item description.
    html italic: item price printStringAsZents.
    html form: [
        html submitButtonWithAction: [ self addToCart ] text: 'Add To Cart'.
        html submitButtonWithAction: [ self answer ] text: 'Done' ]
```

self | all inst vars | decoration | item | cart — thisContext | all temp var | html

MessageNotUnderstood: SmallInteger>>printStringAsZent

```
WAStoreItemView>>renderContentOn:
WAStoreItemView(WAPresenter)>>renderWithContext:
[] in WAAnswerHandler(WAPresenter)>>renderContentOn: {[:ea | ea renderWi
WAAnswerHandler(WADecoration)>>nextPresentersDo:
WAAnswerHandler(WAPresenter)>>renderContentOn:
WAAnswerHandler(WAPresenter)>>renderWithContext:
[] in WADelegation(WAPresenter)>>renderContentOn: {[:ea | ea renderWithCon
WAStoreItemView(WAComponent)>>decorationChainDo:
WADelegation>>nextPresentersDo:
WADelegation(WAPresenter)>>renderContentOn:
WADelegation(WAPresenter)>>renderWithContext:
```

Proceed | Restart | Send | Step | Through | Full Stack | Where

```
renderContentOn: html
    html heading: item title.
    html heading: item subtitle level: 3.
    html paragraph: item description.
    html italic: item price printStringAsCents.
    html form: [
        html submitButtonWithAction: [ self addToCart ] text: 'Add To Cart'.
        html submitButtonWithAction: [ self answer ] text: 'Done' ]
```

self | all inst vars | decoration | item | cart — thisContext | all temp var | html

/seaside/store — http://localhost/se

sushiNet
fresh, raw fish delivered to your door

Search:

Browse

Akami Maguro

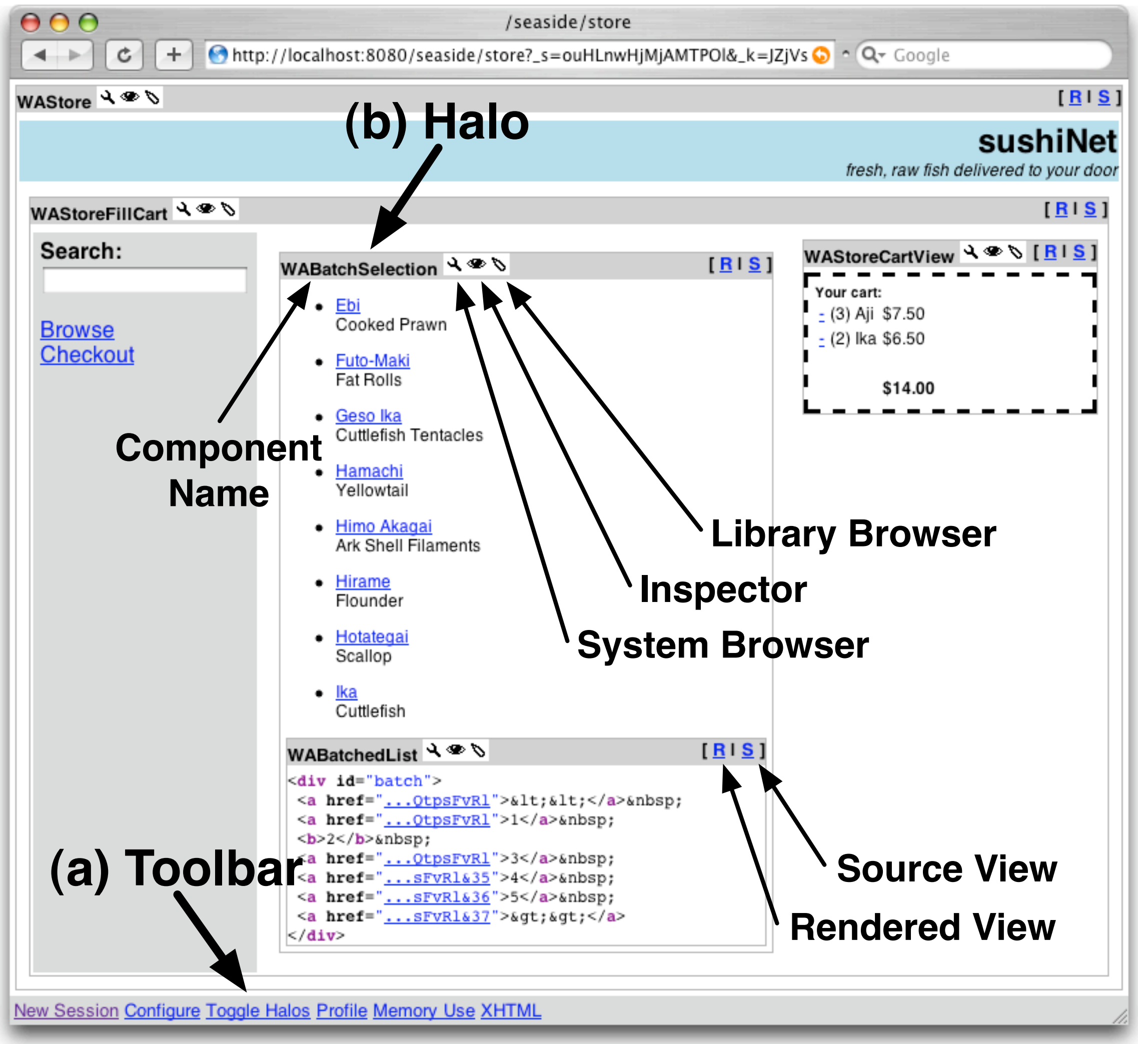
Red Tuna

The lean meat near the spine of the tuna fish. It comes in various shades of red--with the lighter, shinier varieties being the best. For dieters, however, the redder the better. Easy on the palatte. The least expensive of the three types of maguro.

$1.50

Add To Cart | Done

Toolbar



/seaside/store

http://localhost:8080/seaside/store?_s=ouHLnwHjMjAMTPOl&_k=JZjVs

Google

WAStore [R | S]

sushiNet

fresh, raw fish delivered to your door

WAStoreFillCart [R | S]

Search:

Browse
Checkout

WABatchSelection [R | S]

- Ebi
  Cooked Prawn
- Futo-Maki
  Fat Rolls
- Geso Ika
  Cuttlefish Tentacles
- Hamachi
  Yellowtail
- Himo Akagai
  Ark Shell Filaments
- Hirame
  Flounder
- Hotategai
  Scallop
- Ika
  Cuttlefish

WAStoreCartView [R | S]

Your cart:
- (3) Aji $7.50
- (2) Ika $6.50

$14.00

WABatchedList [R | S]

```
<div id="batch">
 <a href="...OtpsFvRl">&lt;&lt;</a>&nbsp;
 <a href="...OtpsFvRl">1</a>&nbsp;
 <b>2</b>&nbsp;
 <a href="...OtpsFvRl">3</a>&nbsp;
 <a href="...sFvRl&35">4</a>&nbsp;
 <a href="...sFvRl&36">5</a>&nbsp;
 <a href="...sFvRl&37">&gt;&gt;</a>
</div>
```

New Session Configure Toggle Halos Profile Memory Use XHTML

Toolbar

- A toolbar is shown at the bottom of the web-application during the development phase.
- It allows one to access some tools:
  - *New Session* restart the application
  - *Configure* opens a dialog letting the user configure some settings
  - *Toggle Halos* shows or hides the halos (explained later)
  - *Profile* shows a detailed report on the computation time used to render the page
  - *Memory Use* display a detailed report on the memory consumption
  - *XHTML* start an external XML validator on this page

Halos

- When enabling the halos, every component gets surronded by a thin grey line and a header giving the class name of the component and a set of buttons to run tools and to change the viewing mode.
  - *System Browser* opens an editor on the current component.
  - *Inspector* opens a view on the current component.
  - *Library Browser* opens an editor that lets a UI designer tweak the associated CSS-Stylesheets.
  - *Source View* provides a pretty-printed and syntax-highlighted XHTML view onto the source code .

Benefits with Seaside

- With PHP: Control flow scattered into files (flight.html, address.html, ...)
- With Seaside: Control flow = method calls (getFlight, getAddress, ...)
- Bandwidth saved: session state is only stored on the server side.
- It makes reuse easier!