CS 683 Emerging Technologies
Fall Semester, 2004
Doc 11 Web App Intro & Some Seaside
Contents

Exercises for Seaside People

1. In Seaside WAHtmlRenderer and its parent class WAAbstractHtmlBuilder contain methods to generate html pages. Find the methods that generate the common html elements like p, h1, h2, table, div, form, input, button, textarea, ul and li.

2. In Seaside create a web app with the root component Seaside.WAAllTests. That application displays a number of tabs, each tab a different example. Look at the examples “Input”, “Html”, “Error”, “Exception” and “Callbacks”. Find the classes used by those examples and make sure you understand how they work.

References

Seaside – A Multiple Control Flow Web Application Framework, Ducasse, Lienhard, Renggli, ESUG, Sept 6-10, 2004

http://www.iam.unibe.ch/~scg/Archive/Papers/Duca04eSeaside.pdf

Seaside Source code

Basic HTTP (Web)

HTTP is Stateless

- Web browser connects to Web server with request
- Web Server handles request
  - Web Server connects (starts) to program (cgi, etc)
  - Program gets request
    - Program frequently connects to database
    - Program handles requests
    - Program returns response
- Web Server returns response
- All connections closed

All requests from Web Browser repeat this process

CGI, Server Pages & Servlets

Common ways to dynamically generate web pages

- CGI & Servlets

  Web request is passed to a program

```
VeryBasicServlet>>doGet: aRequest response: aResponse

  aResponse write: '<HTML><BODY>
      Hello world</BODY></HTML>'.
```

- Server Pages

  Code is embedded in html pages

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
      "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<head>
  <meta http-equiv="content-type" content="text/html; charset=iso-
8859-1" />
  <title>Hi</title>
</head>
<body>
<% response write: 'Hello World'. %>
</body>
</html>
```

Multipage Transactions

Information (state) from one page must be saved for next page

Information can be stored in

- Hidden Fields
- Sessions
- Database

Hidden Fields

Data is stored by client

Server can forget about client

```
<form action="fooBar" method="post" name="Sample">
   <input type="hidden"  name="name" value="Whitney">
   <input type="hidden" name="cart" value="world peace">
   <input type="text" name="Credit Card Number" size="40">
   <input type="submit" name="submit">
</form>

<a href="bar/foo/index.html" title="Help information">Help</a>
```

Problems

Must insure all paths retain data

Pages are coupled

- Code depends on order of pages
- Reduces code reuse

Name clashes

- Field names must be different all other pages in transaction

Security Issues

Presentation & domain logic mixed

```
<form action="fooBar" method="post" name="Sample">
  <% response
        write: 'input type="hidden"  name="name" value="'.
     userName ifNotNil: [ response write: userName printString].
     response
        write: '">';
        write: '<input type="hidden" name="cart" value="';
     cartItems
        do: [:each | response write: each printString]
        separatedBy: [response write: ', '].
     response
        write: '">'.
  %>
etc.
</form>
```

Session & Database Storage of Transaction state

Require session key stored on client side

Store session key in

- Hidden field
- Cookie
- Encode in url

Amazon Example

http://www.amazon.com/exec/obidos/subst/home/home.html/103-0893119-0972637

Go to http://www.amazon.com/ and notice that the url changes and ends in a long number. Now look at all the links on the page and notice that they all contain the same long number.

Problems

No overview of Control Flow

Mixing of application & component logic

Difficulty in composing control flows

Seaside Examples and Concepts
Callback & State

```
Smalltalk.Seaside defineClass: #WACounter
  superclass: #{Seaside.WAComponent}
  instanceVariableNames: 'count '
```

Instance Methods

```
count
  ^ count

decrease
  count := count - 1

increase
  count := count + 1

initialize
  self session registerObjectForBacktracking: self.
  count := 0

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

http://bismarck.sdsu.edu/cs683/seaside/go/counter

Translation to Pseudo-Java

```
public class WACounter extends WAComponent {

int count;

increment() { count = count + 1; }

decrement() { count = count - 1; }

renderContentOn(WAHtmlRenderer html) {html.title( "Counter
Example");
   html.heading( count);
   html.anchorWithAction(increment, "++");
   html.space();
   html.anchorWithAction(decrement, "--");
}

initialize() {
   WASession session = session();
   session.registerObjectForBacktracking(this);
   count = 0;
}
```

Combining Components

```
Smalltalk.Seaside defineClass: #WAMultiCounter
   superclass: #{Seaside.WAComponent}
   indexedType: #none
   instanceVariableNames: 'counters '
```

Instance Methods

```
children
   ^counters

initialize
   counters := (1 to: 5) collect: [:i | WACounter new]

renderContentOn: html
   counters
      do: [:ea | html render: ea]
      separatedBy: [html horizontalRule]
```

http://bismarck.sdsu.edu/seaside/go/multi