Introduction to Seaside

Randal L. Schwartz, merlyn@stonehenge.com

Version 2.01 on 20 July 2009

Goals

- Components
- Callbacks
- HTML Generation (including forms)
- Persistence (including ORM)
- Deploying your application
- AJAX Integration (if time permits)

Getting Seaside

- Squeak - one click image or packages
- VisualWorks - Store or WebVelocity
- GNU Smalltalk - ?
- GemStone/S - GLASS

The Counter Example

- Launch one-click image
  - Ignore GUI for a moment
- Navigate to URL
- Count!
- All that excitement (yawn)

The Smallalk GUI

- Workspace - run snippets of code
- Browser - edit and view code
- Debugger - “d” bugs
- Inspector - look at complex values
- Monticello - share and load code

The Workspace

- Evaluate code snippets
- “do it”: just run the code
- “print it”: run the code, show the result
- “debug it”: run the code in the debugger
- Operates on selection
  - If nothing selected, line cursor is on

Smalltalk in a Hurry

- Just enough to understand Seaside

Objects and classes

- Everything’s an object
- An object belongs to a class
- An object has methods
    - The class (also an object) has methods
- A class inherits from a single superclass
    - Class-side and instance-side separately

Variables

- Alphanumerics
- Camelcased with initial lowercase:
  - rate, accelerationRate
- Value belongs to a class, variables don’t care
- Assign to get a value:
  - rate := 30
- Special vars:
  - self, true, false, nil, super, thisContext

Methods

- Unary: single name, follows variable:
  - rate squared
- Binary: 1-2 punctuation chars:
  - rate * time
- Keyword: names and colons:
  - rate raisedTo: 2.5
  - rate between: 5 and: 10
- Simple precedence!

Literal data

- Strings: ‘hello world’
- Numeric data: 3 2.5 1.23e45 -2e-5
- Symbols: #size #foo:bar: #+

Classes

- Alphanumeric, initial uppercase
- Class methods are often constructors:
  - rates := Set new.
- But could also have other uses:
  - superclassOfSet := Set superclass.

Method syntax

- Signature (like message send without self):
  - squared
  - * aNumber
  - raisedTo: aNumber
  - between: lowNumber and: highNumber
- Temporaries: | aDog aCat |
- Statements separated by periods
- Last statement can have ^ (“answer this”)
- Comments are in “double quotes”

Control structures

- Conditionals:
  - aBoolExpr ifTrue: [some. code. here].
  - aBoolExpr ifFalse: [some. other. code].
  - #ifTrue:ifFalse:, #ifFalse:ifTrue:
- Loops:
  - [code. code. aBoolExpr] whileTrue.
  - [code. aBoolExpr] whileTrue: [code].
  - #whileFalse, #whileFalse:

The Code Browser

- Packages - groups of classes
- Classes
- Class/instance/comment toggle
- Method categories (including "all")
- Method names
- Lower pane views/edits selection
  - Sometimes preloaded with a template
- Lots of coding help available in menus

The Debugger

- Debug notifier: proceed/cancel/full
- Full debugger:
  - Stack
  - Code pane (current line highlighted)
  - Instance vars
  - Temps and arguments
- Everything is live, editable, resumable
- Action buttons to step in, over, through

Hello World

- Create class for top-level component
  - Should inherit from WAComponent
- class #canBeRoot for GUI access
  - Or register during class #initialize
- Components implement:
  renderContentOn: html
- In our case:

  html text: ‘hello world’.

Configure the app

- Visit configuration screen (/seaside/config)
- Create a new URL path (below /seaside)
- Select our class as the root component
- Visit the URL!

When the Web Breaks

- Add time display to output
- Refactor it to use concatenation
- Boom! (Needs #asString)
  - Walkback in browser
  - Select debug to use Smalltalk GUI
  - “Proceed”, and browser refreshed

Halos

- Inspect components
- Edit CSS (for prototyping)
- View pretty-printed HTML source
- Edit source code (proof of concept)

Web Velocity

- Stay in web browser for:
  - Code browser
  - Debugger
  - Inspector
  - Source code management
- Everything!
- Lots of scaffolding for database views

Configuring brushes

- Each step separate:
  brush := html heading.
  brush level: 3.
  brush with: ‘my third level heading’.
- Combining first and second steps:
  brush := (html heading) level: 3.
  brush with: ‘...’.
- Any number of configurations
- But #with: has to be last!

Cascades

- Cascade omits common object:
  batallion selectTank target: enemy; fire.
- Same as (without the variable):
  aTank := batallion selectTank.
  aTank target: enemy.
  aTank fire.
- So our heading looks like:
  html heading level: 3; with: ‘my head’.

Fancier Blocks

- Like methods in square brackets
- Argument list (if any)
  :arg1 :arg2 :arg3 |
- Temporaries (if any)
  | temp1 temp2 |
- Statements
  arg1 dothis. arg2 dothat. arg3 + arg1.
- ^ exits enclosing method, not block!

Example

- add3ToDoubleOf := [:x | x * 2 + 3].
- a := add3ToDoubleOf.
- b := a value: 17.
- c := a value: (17 / 2).

Callbacks

- html anchor with: ‘text’ - not useful
- html anchor

```
callback: [self increase];
with: ‘text’.
```

Components

- Reusable chunks of HTML
- Individual instance vars for state
  - Example: current counter value
- State can be rewound if a URL is reused
  - Or not, depending on coder’s choice
- Components can be nested
  - Need to be declared with #children

Back to the Counter

- Simple code
- Instance var holds the counter
- Main view shows increase/decrease buttons
- Actions linked via callbacks

Collection classes

- OrderedCollection
- Array (fixed size OrderedCollection)
- Set
- Bag (counted items of Set)
- SortedCollection (order from chaos)
- Dictionary (key/value mappings)
- Interval (5 to: 100 by: 3)

Collection protocols

- collect:
  - fractions := (1 to: 10) collect: [:n | 1 / n]
- do:
  - fractions do: [:each | html div: each]
- select:
  - overQuarter :=
    fractions select: [:f | f > 0.25]

Multicounter

- Just a bunch of counters

The Back Button

- Callback URLs modify instance vars
- What if we reinvoke the same URL?
- Might want original action on old value
  - Or maybe on new current value
- To act on old value, use #states
  - Values are associated with URLs
  - Frozen and thawed as necessary
- Otherwise, acts on current value

Styles

- Add class to any relevant brush:
  html div class: ‘entry’; with: ‘some text’.
- Style with CSS
  - External files can be edited by designer
- Simple styles for testing defined inline
  - String returned by #style on component

Forms

- Input fields painted with brushes
- Callbacks executed on form submission
- No need to name anything:
  ```
  html text: 'name:'.
  html textInput callback: [:v | self name: v].
  ```
- Default values can be provided
- With right accessors, code is simple:
  ```
  html textInput on: #name.
  ```

LCM two numbers

- Build code to do this

Persistence Solutions

- Saving the image
- Writing objects
- Object prevayler (Sandstone, Prevayler)
- Object database (GemStone/S, Magma)
- Object/Relational Mapper (GLORP)

GLASS

- GemStone/S object engine
- Linux, Apache, Smalltalk, Seaside
- All wrapped up in a VMWare appliance
- Free to use for small applications
  - Even commercial applications!
- Not open source though. :(
- As hits increase, scale up for modest fee

Ajax

- JavaScript library integration
  - Scriptaculous (now)
  - jQuery (soon)
  - MooTools
- No need to write JavaScript
  - Everything is coded from Smalltalk!
- [demo]

Testing

- Component level testing
  - Using standard Smalltalk Unit Tests
- HTTP level testing
  - SeasideTesting package
  - Albatross (like Selenium)

More info

- http://seaside.st/
- http://MethodsAndMessages.vox.com/