

Seaside - An Object-Oriented Web Application Development Framework

Seminar

**Weiterführende Themen zu Internet- und WWW-Technologien
Sommersemester 2016**

Lennard Wolf

Betreuer:

Matthias Bauer, Haojin Yang
Prof. Dr. Christoph Meinel

September 13, 2016

Contents

1	Introduction	3
2	History of Seaside	3
2.1	Historical Context	3
2.2	Motivation	4
2.3	Development	4
2.4	Current State and Key Features	5
3	Working with Seaside	6
3.1	Smalltalk	7
3.2	Installing Seaside	7
3.3	Setting Up a Website	8
3.4	Rendering and Subcomponents	9
3.5	Debugging	10
3.6	Managing Sessions	10
4	Evaluation	10
5	Conclusion	10

1 Introduction

Seaside is an open source object-oriented web application development framework for the Smalltalk programming language. The strict object-oriented approach as well as its concept of user sessions offers a unique perspective on programmers' workflow and logic behind web apps. It thus differs immensely from today's more common webframeworks in the field such as *AngularJS*, *Django*, or *Ruby on Rails*. This text aims at a basic understanding of its history and the context in which it was conceived, its concepts, how they differ from other frameworks of its kind, what working with it looks like, and what can be learned from it.

This work is structured as follows. Section 2 introduces the history of Seaside including its context, the motivations behind it, as well as its current state. In Section 3 we introduce the Smalltalk programming language, the core concepts of Seaside, and the workflow that results from them. Then, in Section 4, we take a critical look at Seaside, evaluate it in the context of current technologies, and summarize what Seaside can teach web developers of today. Section 5 concludes this work.

2 History of Seaside

To understand Seaside's origins, knowledge of the technologies currently popular at the beginnings of its development is required first. This Section will thus start out with such an overview, then present the original and current motivations behind Seaside, followed by a list of the people involved, a development timeline, and finally the current state of both the feature list and the development in general will be examined.

2.1 Historical Context

The viewable internet started out as a network of static pages usually written in HTML. With the progress of time came the want to not only make these sites more visually pleasing, but for the user to be able to interact with them. For this, web application development frameworks have been created for many different programming languages and with differing opinions of what the web of the future should look like. At the beginning of the new millenium, technologies such as *AJAX* (**A**synchronous **J**avaScript **A**nd **X**ML) did not exist, mostly because there had not been a broadly accepted answer to the question, whether web applications should run on the *server* or on the *client* side. Today, in 2016, most web apps run in the users' browsers. In the early 2000s however, many people were sceptical towards the idea of just letting any foreign script run freely on their machine, leading them to block JavaScript applications by default. This state of affairs led many developers to build their interactive web applications for server side execution.

During this time, questions like this one, or for example whether URLs have to be human readable, were still up for debate. This is why technologies like Seaside present their own take on them which might appear confusing and non-sensical to some web developers of today.

2.2 Motivation

Over the last few years, some best practices have come to be widely accepted in the web development world. Share as little state as possible. Use clean, carefully chosen, and meaningful URLs. Use templates to separate your model from your presentation.

Seaside is a web application framework for Smalltalk that breaks all of these rules and then some. Think of it as an experiment in tradeoffs: if you reject the conventional wisdoms of web development, what benefits can you get in return? Quite a lot, it turns out, and this "experiment" has gained a large open source following, seen years of production use, and been heralded by some as the future of web applications.

— Avi Bryant, *Web Heresies: The Seaside Framework* [2]

Avi Bryant and Julian Fitzell started working on a web application framework for Smalltalk in 2001 within the context of their consulting endeavours, as well as their current project of building a theatre boxoffice sales system, which was supposed to be using it. Bryant had built such a framework before in Ruby, called *Iowa*, which was heavily influenced by Apple's *WebObjects*¹ for Java. The philosophies shared by these two frameworks were *object-orientation*, *database connectivity*, and *fast prototyping*². These then, as well as the concept of *continuations* through *user sessions*, built the foundations for the new project, which aimed to bring a unique framework to Smalltalk environments.

2.3 Development

The first version, Seaside 0.9, was announced in February 2002 by Bryant and Fitzell to the *squeak-dev* mailing list³. It supported action callbacks, a session state system, and a component system, of which the latter two are the core of the Seaside programming principles.

From the beginning it was open source under the MIT License and because of the big amount of feedback from the Smalltalk community, the development could happen

¹The still active community of volunteer developers can be found at <https://wiki.wocommunity.org/display/WEB/Home>, accessed: 2016-09-13

²This feature is best shown by example, as Ramon Leon created a fully functional skeleton for a blog within 15 minutes with Seaside, back in 2006: <http://onsmalltalk.com/screencast-how-to-build-a-blog-in-15-minutes-with-seaside>, accessed: 2016-09-13

³The community is still active today and can be found at <http://lists.squeakfoundation.org/mailman/listinfo/squeak-dev>, accessed: 2016-09-12

quite fast. Thus, version 2.0 came out in October 2003 and was a complete rewrite of the system. It included many new features such as HTML generation, in-browser development tools, and a *Views* layer to name a few. [4]

2.4 Current State and Key Features

Since Seaside is still in use in some places, it still has contributors⁴ that fix bugs, work on the support of other technologies and their updates, improve memory efficiency as well as rendering speed, and maintain the overall code quality. The latest stable version, Seaside 3.2⁵, was released on May 5, 2016 and mainly focuses on performance and scalability. Improvements in these areas over time are presented on the <http://www.seaside.st> website by comparison of unit test results with older version, as shown in Figure 1.

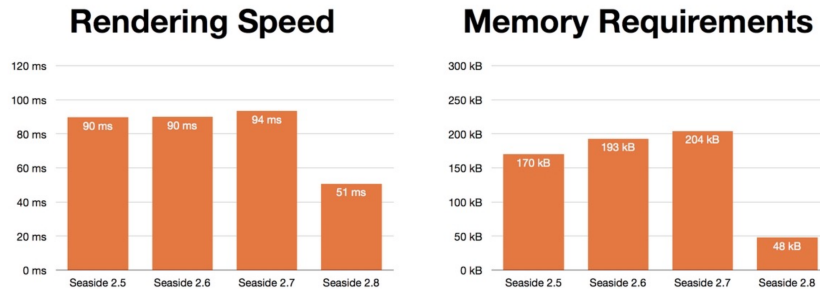


Figure 1: The substantial performance improvements in version 2.8

As mentioned above, the key ideas underlying the design of Seaside are *object-orientation*, *continuations*, as well as *fast prototyping and development*. The object-orientated approach brought about the so called *components*, which are the stateful and reusable building blocks of a website. Their state is saved in *user session objects*, or *continuations*, on the server and each of which is linked to a certain user via **TODO: XXXX**, so that when one returns to the website, its state is the same as when one has left it earlier. Figure 2 shows what a website subdivided into components with parent-child-relationships can look like.

The stateful components based site creation makes Seaside’s workflow comparable to that of working with the *React* JavaScript library⁶. As it has become common, Seaside too works over the *Model View Controller* architecture pattern, which had actually originally been conceived by Trygve Reenskaug whilst working on Smalltalk at Xerox PARC [6]. But furthermore, the framework provides productivity tools such as *live debugging* in the Smalltalk environment with the powerful Smalltalk debugger that not

⁴A full list can be found at <http://www.seaside.st/community/contributors>, accessed: 2016-09-12

⁵Releases can be found on the GitHub releases page: <https://github.com/SeasideSt/Seaside/releases>, accessed: 2016-09-13

⁶For documentation etc. visit <https://facebook.github.io/react>, accessed: 2016-09-13



Figure 2: A webpage and the components it is made of represented by the dashed boxes

only lets developers browse the call stack easily, but edit the code within the debugger and make the changes effective immediately, providing a smooth bug fixing experience. If enabled, Seaside allows inspection and editing of components within the page shown in the browser, and in error cases, the call stack can be viewed and the faulty code fixed from within the browser too.

When creating a website with Seaside, no actual HTML code has to be written, since the Smalltalk code which describes the components is used by the renderer to generate HTML code automatically. Further, such a website can also be rendered with *morphs*, the building blocks of most GUI based Smalltalk environments, enabling the development without the need of a browser.

Over the time, Seaside has come to support many common web technologies. Since *CSS* (**C**ascading **S**tyle **S**heets) is commonly used to make webpages visually appealing, it is supported by Seaside, as well as AJAX, Comet, and JavaScript.

3 Working with Seaside

In this Section we will shortly introduce the object-oriented programming language Smalltalk in which Seaside is written and used. Then we will explain the installation process and the basic workflow with Seaside, which entails the following two aspects. The first is configuration through visual menus for the initial setup or core structural changes. The second is actual writing of code in the Smalltalk environments' GUI *Browser*-windows[5].

3.1 Smalltalk

Smalltalk was created in the 70's at Xerox PARC by Alan Kay *et al.* for research and educational purposes. It is dynamically typed and purely object-oriented, meaning that *everything in a Smalltalk environment is an object* and can thus be easily interacted with. Smalltalk environments such as the free and open source Squeak or the proprietary Gemstone⁷ are normally shipped with the Smalltalk core classes (e.g. the fundamental `Object` class, which all other classes inherit from), as well as additional ones, depending on the environment's use cases. One of their most important features is *reflection*, with which the programmer is able to inspect and modify every part of the system at runtime. This is especially useful in cases where a system may never shut down, because the changed code does not need to be recompiled but comes into effect immediately.

The syntax can be described as minimalistic. Smalltalk code consists of sequences of messages sent to objects (`anObject message`)⁸. Listing 1 presents syntax basics that are helpful when approaching Seaside.

```
"Send the message 'not' to 'true' and assign the returned value to a."
a := true not.

"Messages can be chained."
b := a not not not.

"Blocks are sequences of executable code that return the value of the
last line. They can take any number of arguments (here x and y)."
aBlock := [ :x :y | z := x * y. z - (2 * x) ].

"aBlock is executed with varying parameters depending on b's state."
answer := b ifTrue: [aBlock value: 7 value: 8]
           ifFalse: [aBlock value: 0 value: 0].
```

Listing 1: The core syntax of Smalltalk

3.2 Installing Seaside

As with most Smalltalk packages, Seaside can be installed with a common package manager. In the cases of Squeak and Pharo, Metacello can automate the installation with the script shown in Listing 2. However for other environments, like Gemstone, the installation process is a bit more complex, but can be found on Seaside's GitHub page⁹.

⁷Squeak: <http://squeak.org> Gemstone: <https://gemtalksystems.com>, both accessed: 2016-09-11

⁸A good overview on the Smalltalk syntax was created by Chris Rathman and can be found at [http:](http://)

```
Metacello new
configuration: 'Seaside3';
repository: 'http://www.smalltalkhub.com/mc/Seaside/MetacelloConfigurations/main';
version: #stable;
load: 'OneClick'.
```

Listing 2: Seaside installation script for Squeak and Pharo

3.3 Setting Up a Website

After opening the *Seaside Control Panel* via the *Apps* tab in the environment, the GUI allows the user to serve the Seaside application through a chosen port. Figure 3 shows the Control Panel with a server instance and a browser showing the Seaside welcome screen, which is the default root page when setting up an application.

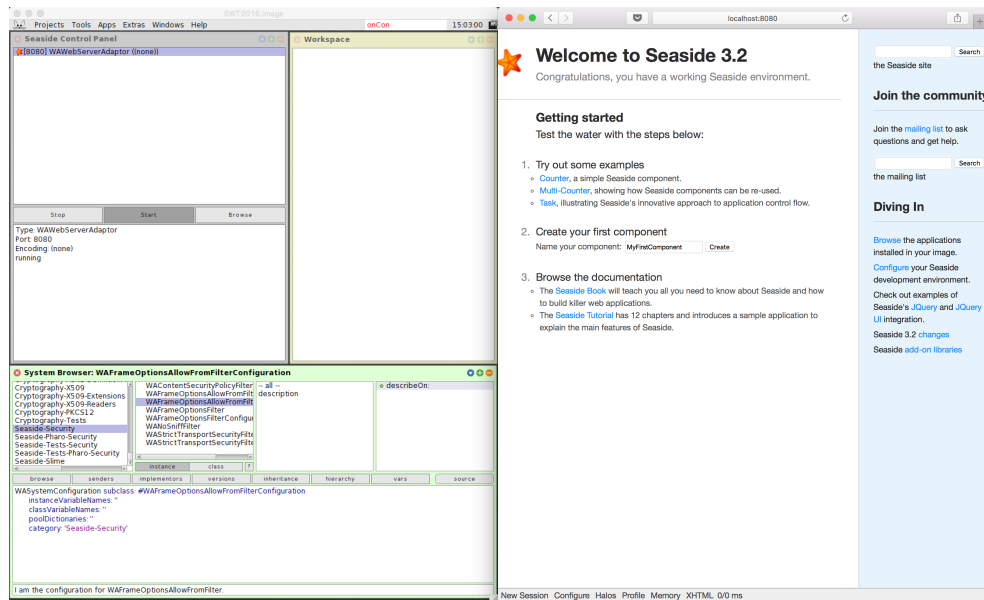


Figure 3: A Squeak environment with a running Seaside server and a browser showing the root site

When starting out with creating a website, one would typically create a root component, which will become the first thing that users will see, when visiting the website. Component classes are always subclasses of `WAComponent`, Listing 4 exemplifies the creation of such a component, here given the name `MyRootComponent`.

[//www.angelfire.com/tx4/cus/notes/smalltalk.html](http://www.angelfire.com/tx4/cus/notes/smalltalk.html)

⁹<https://github.com/SeasideSt/Seaside>, accessed: 2016-09-13


```
MyRootComponent class>>canBeRoot
  ^true

MyRootComponent class>>initialize
  | app |
  app := self registerAsApplication: #myWebsite.
  app libraries add: SULibrary.
  app preferenceAt: #sessionClass put: CustomSession.
  self children: OrderedCollection new

MyRootComponent>>renderContentOn: html
  html div: 'Hello World'

MyRootComponent>>children
  ^children
```

Listing 3: Making the `MyRootComponent` the website’s root and adding basic functionality

3.4 Rendering and Subcomponents

In Seaside, you can both render the View (`renderContentOn:`) or the Model itself (`renderOn:`). `MyRootComponent` will thus greet visitors with "Hello World".

The `children` method is necessary to enable the component to have *subcomponents*. Say we now want our website to be divided into three components, `header`, `currentBody`, and `footer`, and these are supposed to be subcomponents of `MyRootComponent`. Then we need to create these components as individual classes, just like their parent, and register them as instance variables as well as children.

```
MyRootComponent>>children
  ^ { header. currentBody. footer }

renderContentOn: html
  html div id: #header; with: [html render: header].
  html div id: #body; with: [html render: currentBody].
  html div id: #footer; with: [html render: footer]
```

Listing 4: Subdividing `MyRootComponent` into its subcomponents and making it only render their content

3.5 Debugging

3.6 Managing Sessions

4 Evaluation

TODO: <http://amber-lang.net/index.html> TODO: *object-orientation, continuations, and fast prototyping/development.*

- comparison of features:
- everything is an object
- hides synchronicity -> no callback hell
- Each session object can be multiple megabytes big If server crashes, all sessions are lost
- Gemstone distributed but expensive
- Javascript not really debuggable
- Very hard to implement human readable URLs
- Not on par with today's demands of a fast dynamic webpage since its server side

5 Conclusion

TODO: URLS IN BIBLIOGRAPHY!!!

References

- [1] Daniel G. Bobrow, Kenneth Kahn, Gregor Kiczales, Larry Masinter, Mark Stefik, and Frank Zdybel. Commonloops: Merging lisp and object-oriented programming. In *Conference Proceedings on Object-oriented Programming Systems, Languages and Applications*, OOPLSA '86, pages 17–29, New York, NY, USA, 1986. ACM.
- [2] Avi Bryant. "web heresies: The seaside framework" session notes, oscon 2006, July 2006.
- [3] Stéphane Ducasse, Lukas Renggli, David Shaffer, and Rick Zaccone. *Dynamic web development with Seaside*. Square Bracket Associates, 2010.
- [4] Julian Fitzell. Seaside history, 2008.
- [5] Michael Perscheid, David Tibbe, Martin Beck, Stefan Berger, Peter Osburg, Jeff Eastman, Michael Haupt, and Robert Hirschfeld. *An Introduction to Seaside - Developing Web Applications with Squeak and Smalltalk*. Hasso-Plattner-Institute, 2008.
- [6] Trygve Mikjel H Reenskaug. The original mvc reports. 1979.
- [7] Guy L. Steele Jr and Richard P. Gabriel. The evolution of lisp. *ACM SIGPLAN Notices*, 28(3):231–270, 1993.