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 14, 2016

Contents

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 it's 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* [1]

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 list[3]. 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

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

[2] This 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

[3] The 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. [3]

2.4 Current State and Key Features

Since Seaside is still in use in some places, it still has contributors[4] 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 3.2[5], 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.

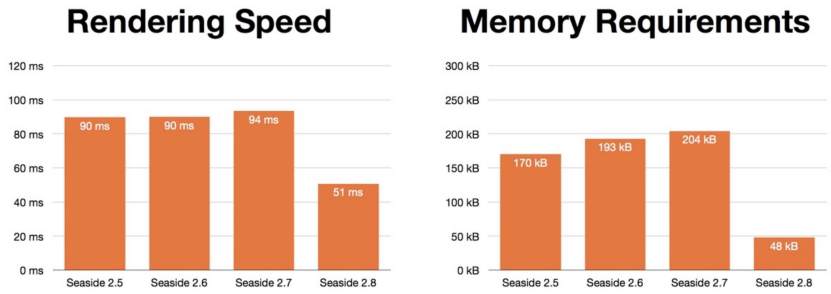

Figure 1: The substantial performance improvements in version 2.8

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 in a registry, 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 library[6]. 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 [5]. But furthermore, the framework provides productivity tools such as *live debugging* in the Smalltalk environment with the powerful Smalltalk debugger that not

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

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

[6] For documentation etc. visit `https://facebook.github.io/react`, accessed: 2016-09-13

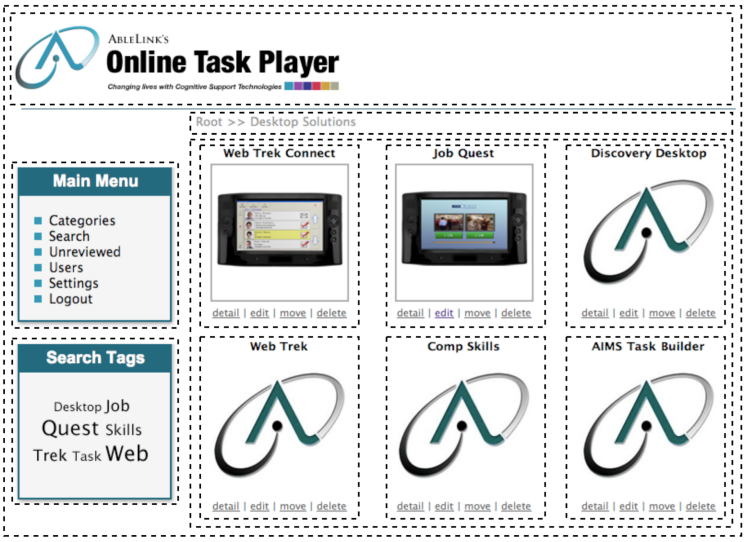

Figure 2: A webpage and the components it is made of represented by the dashed boxes

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[4].

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 Gemstone[7] 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]. Figure 3[9] presents the Smalltalk syntax which is helpful to be familiar with when approaching Seaside.

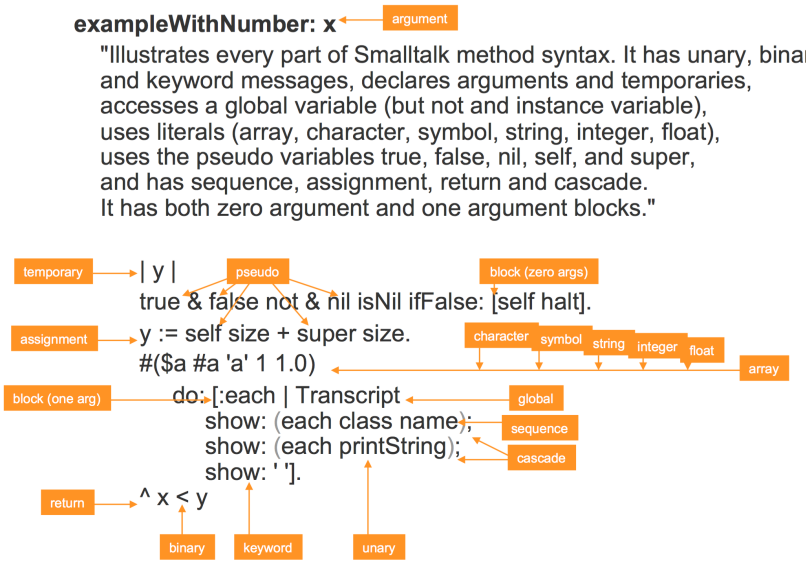

Figure 3: The Smalltalk syntax in a nutshell

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

[7] Squeak: `http://squeak.org` Gemstone: `https://gemtalksystems.com`, both accessed: 2016-09-11

[8] A good overview on the Smalltalk syntax was created by Chris Rathman and can be found at `http://www.angelfire.com/tx4/cus/notes/smalltalk.html`, accessed: 2016-09-10

[9] Taken from a software architecture course presentation of the HPI Software Architecture Group

by pasting the script shown in Listing 1 into a *Workspace*-window and then executing it by selecting it and pressing `cmd+d` or `ctrl+d`. However for other environments, like Gemstone, the installation process is a bit more complex, but can be found on Seaside's GitHub page[10].

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

Listing 1: Seaside installation script for Squeak and Pharo

3.3 Setting Up a Website

After opening the *Seaside Control Panel* vie the *Apps* tab in the environment, the GUI allows the user to serve the Seaside application through a chosen port. Figure 4 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.

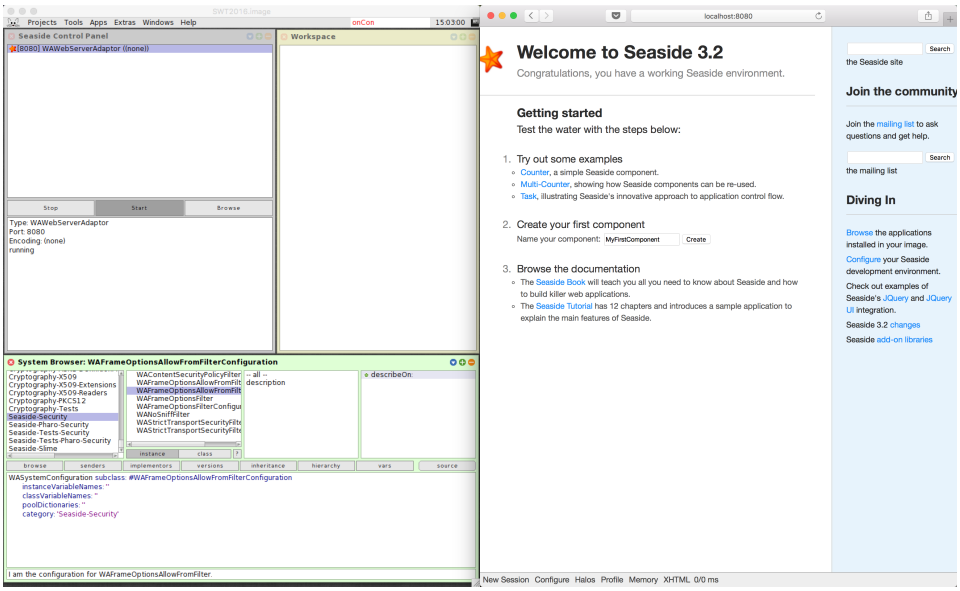

Figure 4: A Squeak environment with a running Seaside server and a browser showing the root site

[10] `https://github.com/SeasideSt/Seaside`, accessed: 2016-09-13

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 2 exemplifies the creation of such a component, here given the name `MyRootComponent`.

```
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'
```

Listing 2: 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. Listing 3 shows the necessary steps to get Seaside to render the new subcomponents whenever `MyRootComponent` gets rendered.

3.5 Debugging

A very prominent feature of Smalltalk environments is the powerful debugger, shown in Figure 5. On the top one can see the call stack, with the code of the currently selected method below. The method can be edited right then and there, with the changes having immediate effect, as Smalltalk environments are live systems. Below the view of the method one can inspect all involved Objects and their state at the time of the currently selected method call.

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

MyRootComponent>>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 3: Subdividing `MyRootComponent` into its subcomponents and making it only render their content

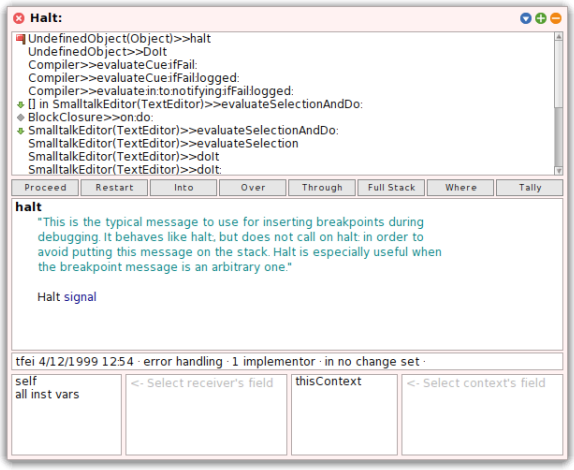

Figure 5: The debugger in a Squeak environment

If live debugging is enabled in Seaside, developers can test their website and whenever an error is raised, the website view in the browser will freeze and a debugging window will appear. Then bug can be fixed, saved, and the website will unfreeze again and do whatever it was supposed to.

3.6 User Sessions

Sessions, or sometimes also referred to as continuations, because they enable users to continue on a given Seaside web app exactly as when the left it the last time, are instances of the `WASession` class, or custom subclasses thereof, in case the developers need sophisticated statefulness, such as keeping a user logged in. Such a subclass would typically have an additional set of instance variables, such as `isLoggedIn`. The session object can also be used to hold other vital information that can be accessed by the components, such as the username to display it on the website. [2]

Unlike RESTful websites, Seaside keeps session information in the URL. On a locally running server, a common URL would be `http://localhost:8080/welcome?_s=_cz2gmrtjih_mfGk&_k=a7eMbFS3W04vCKCr`. This holds three different information: `welcome` refers to the currently visited subpage, which is linked to a particular component in Seaside. `s=_cz2gmrtjih_mfGk` is the *handler field* and signifies the ID of the unique session object, which are both linked in a `WARegistry` object that is unique to each application. `k=a7eMbFS3W04vCKCr` is the *action field* and refers to (up to 20) *handlers*, to which the session object forwards incoming requests and which can also redirect to the `WALRUCache`, or *least recent used cache*. [4]

4 Evaluation

In this Section we will first take another quick look at the fundamental differences between Seaside and other frameworks of its kind. Then we will discuss its problems and try to find possible reasons why it has not been adopted in a large scale. Afterwards, we share some thoughts on what Seaside can teach web developers of today and why it can be beneficial to experiment with it.

4.1 Comparison With Other Technologies

Seaside is not the only Smalltalk based web development technology. *Amber*[11] is a Smalltalk dialect that is used for client side web applications, bringing the benefits of Smalltalk to a more modern approach to web apps. Like Amber, Seaside is fully object-oriented, which makes it differ from the more script based view on web development of JavaScript and the likes the most. This makes a modular approach to website building possible, which over the time has become a popular way of thinking in JavaScript too, made possible by abstractions and frameworks such as React with its stateful components and *ember*[12] with its modular *handlebar*-templates. Firthermore, object-orientation hides issues with synchronicity, ridding the development of any need for callbacks, which is not an uncommon issue when working with JavaScript.

4.2 Problems

The most obvious problem with Seaside is that its applications run on the server side, which makes it difficult to scale well (imagine the *Facebook* web app running server side). Such an approach has also mostly become obsolete, since personal computers have become quite powerful and JavaScript is usually allowed to run in browser by default, complex and dynamic webpages can run smoothly on most machines of today. When using JavaScript with Seaside, the code will not be as easily debuggable anymore, taking

[11] `http://amber-lang.net/index.html`, accessed: 2016-09-14
[12] `http://emberjs.com/`, accessed: 2016-09-14

away large parts of the benefits it provides. Furthermore, the absence of human readable URLs is an issue that can not be ignored and which puts it behind other technologies that allow an easy implementation of such a feature.

Running a distributed Seaside system with the *GLASS* stack (**G**emStone, **L**inux, **A**pache, **S**easide, and **S**malltalk) can be very powerful and stable, albeit very expensive. Such a system can thus only be used in highly specialized fields with specific needs, and will not cater to larger development public.

At last, Smalltalk has been sort of forgotten about, possibly because of the uncommon approach of the live environment which to some might appear overwhelming or restrictive at first glance. Furthermore, Seaside goes against many web application development concepts that people know, and since it is part of our nature to be skeptical or even fearful of the unknown, it might have a rather deterring effect on many web developers.

4.3 What Seaside Can Teach

Web development has gotten more and more centred around a rather small number of technologies, making them look like unquestionable truths, which can create a sort of narrow frame in which people think about problems in this field. It could however be argued that it is important for people of any profession to sometimes leave their comfort zone and try other methods and technologies which they don't know yet, allowing them to think about the problems they face on a day to day basis in new ways.

Getting to know foreign systems and frameworks such as Seaside could thus allow developers to understand that their approach is not the only answer. Such a broader perspective not only helps in critical thinking, but also lays the groundwork of *thinking outside the box* and developing entirely new technologies and approaches.

5 Conclusion

In this work we have given a concise introduction to the object-oriented web application development framework Seaside.

First, its history as well as context were outlined and its core motivations and features have been presented. This was followed by a short overview of working with Smalltalk and Seaside, including the key features of components, debugging, and user sessions. After comparing it with other technologies of its kind, we discussed its problems and then described, that getting to know Seaside might still be beneficial by broadening one's view on the field of web development.

TODO: URLS IN BIBLIOGRAPHY!!!

References

[1] Avi Bryant. ”web heresies: The seaside framework” session notes, oscon 2006, July 2006.

[2] Stéphane Ducasse, Lukas Renggli, David Shaffer, and Rick Zaccone. *Dynamic web development with Seaside*. Square Bracket Associates, 2010.

[3] Julian Fitzell. Seaside history, 2008.

[4] 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.

[5] Trygve Mikjel H Reenskaug. The original mvc reports. 1979.