ARTICLE

Reddit.st - In 10 elegant Smalltalk classes

Implementing a Reddit style web application in Smalltalk

Using Seaside, Glorp and PostgreSQL

Sven Van Caekenberghe (Beta Nine)

July 2010, First Revision

This is a tutorial showing how to implement a small but non-trivial web application in Smalltalk using Seaside, Glorp and PostgreSQL. Reddit, is web application where users can post interesting links that get voted up or down. The idea is that the 'best' links end up with the most points automatically. Many other websites exist in the area of social bookmarking, like Delicious, Digg and Hacker News.

In 2005 Reddit switched their implementation from Common Lisp to Phyton. As a reaction I published a Lisp Movie (screen cast) called Episode 2: (Re)writing Reddit in Lisp in 20 minutes and 100 lines also known as Reddit.lisp - In less than 100 lines of elegant code. Although the title was provocative, to get attention and for fun, and this was lost on some people, the key message was just to show that it is perfectly possible to write nice web applications in Lisp and to give an example of how to do so. The reasons for their switch were social, not technical.

Reddit.st is yet another variant of this example, reimplementing the example in Smalltalk. It adds **persistency** in a relational database, **unit tests** as well as web application **components** to the mix.

The 10 main sections of this article follow the development of the 10 classes making up the application. The focus of the Smalltalk version is not so much on the small size or the high developer productivity, but more on the fact that we can cover so much ground using such powerful frameworks, as well as the natural development flow from model over tests and persistence to web GUI.

Contents

North Sea Beach, Houvig, Denmark

1. RedditLink

In this article I assume that you know enough Smalltalk to read it, understand what web applications are and how Seaside basically works. If not, you could start by reading my other article, A Day At The Beach (Exploring the Seaside Web Application Framework for Smalltalk - A Dramatic Improvement in Abstraction and Productivity) for an introduction. I also will assume that you have a basic understanding of relational databases and/or SQL.

The central object of our application will be `RedditLink`, an interesting URL with a title, a created timestamp and a number of points. It has the following properties:

- id
- url
- title
- created
- points

These will become the instance variable of our class. Create a new `Object` subclass by editing the class template.

```
Object subclass: #RedditLink
  instanceVariableNames: 'id url title created points'
  classVariableNames: ''
  poolDictionaries: ''
  category: 'Reddit'
```

Next, we use the class refactor tool to automatically generate accessors (getters and setters) for all our instance variables. With these implemented we can write our `#initialize` and `#printOn:` methods.

```
initialize
  self points: 0; created: TimeStamp now

printOn:
  super printOn: stream.
  stream nextPut: $(.
  self url printOn: stream.
  stream nextPut: $,.
  self title printOn: stream.
  stream nextPut: $)

posted
  ^ TimeStamp now - self created
```

We also added a `#posted` method that will return the `Duration` of time the link now exists. We will need that when rendering links later on. To make it a little bit easier for others to create new instances of us, we add a class method called `#withUrl:title:`.

```
withUrl: url title: title
  ^ self new
      url: url;
      title: title;
      yourself
```

Apart from creating and displaying `RedditLinks`, we want to be able to vote them up and down. So lets add two action methods, `#voteUp` and `#voteDown`.

```
voteUp
  self points: self points + 1

voteDown
  self points > 0 ifTrue: [ self points: self points - 1 ]
```

The core of the `RedditLink` object is now finished. Everything is ready to make instances and use them.

2. RedditLinkTests

Units tests are very important, not so much in small examples like this one, but especially in larger applications. Having a good set of unit tests with descent coverage helps protect the code during changes. At the same time, unit tests function as working documentation. Instead of writing scratch test code in some workspace, you can just as well write a unit test. We make a `TestCase` subclass named `RedditLinkTests` and add 3 test methods.

```
testInitialState
  | link |
  link := RedditLink new.
  link assertContractUsing: self.
  self assert: link points isZero

testCreate
  | link url title |
  url := 'http://www.seaside.st'.
  title := 'Seaside'.
  link := RedditLink withUrl: url title: title.
  link assertContractUsing: self.
  self assert: link points isZero.
  self assert: link url = url.
  self assert: link title = title

testVoting
  | link |
  link := RedditLink new.
  link assertContractUsing: self.
  self assert: link points isZero.
  link voteUp.
  self assert: link points = 1.
  link voteDown.
  self assert: link points isZero.
  link voteDown.
  self assert: link points isZero
```

After creating a `RedditLink` object, and/or manipulating it, these methods assert that certain conditions hold. To improve code sharing (we'll need it again in section 5) we add a method called `#assertContractUsing:` to `RedditLink` that checks the basic contract of the receiver, using `#assert:` on an arbitrary object. For completeness, we also implement a general `#isValid` test.

```
assertContractUsing: object
  object perform: #assert: with: (self url isNil or: [ self url isKindOf: String ]).
  object perform: #assert: with: (self title isNil or: [ self title isKindOf: String ]).
  object perform: #assert: with: (self created isKindOf: TimeStamp).
  object perform: #assert: with: (self points isKindOf: Integer).
  object perform: #assert: with: (self posted asSeconds >= 0).
  object perform: #assert: with: (self printString isKindOf: String)

isValid
  self assertContractUsing: self
```

Smalltalk has integrated tools to run these tests quickly. There is a separate Test

Runner. But you can run tests directly from a Browser as well. The browser even has a permanent indication for successful and failed tests!

3. RedditSchema

To make our application less trivial, we are going to make our collection of links persistent in a relation database. For this we are going to use Glorp, an object-relational mapping tool. Glorp will take care of all the SQL! To do its magic, Glorp needs a `DescriptorSystem` that tells it 3 things: the class models involved, the tables involved and the way the two map (which it calls a descriptor). In this simple case we are just mapping one object into one table, so the descriptor system might look a bit verbose. Just remember that Glorp can do much, much more advanced things. Furthermore, there exist extensions to Glorp that implement ActiveRecord style automatic descriptors.

```
constructAllClasses
  ^ (super constructAllClasses)
      add: RedditLink;
      yourself

classModelForRedditLink: aClassModel
  #(id url title created points) do: [ :each | aClassModel newAttributeNamed: each ]
```

So first we tell Glorp about our class model, `RedditLink`, and which instance variable are to be persistent attributes. Instead of some XML description, a much more powerfull Smalltalk object model is being built. Conventions in methods names are being used to glue things together.

```
allTableNames
  ^ #( 'REDDIT_LINKS' )

tableForREDDIT_LINKS: aTable
  (aTable createFieldNamed: 'id' type: platform serial) bePrimaryKey.
  aTable createFieldNamed: 'url' type: (platform varchar: 64).
  aTable createFieldNamed: 'title' type: (platform varchar: 64).
  aTable createFieldNamed: 'created' type: platform timestamp.
  aTable createFieldNamed: 'points' type: platform integer
```

The second step is to describe which tables are involved. So here we list all the fields (columns) of our table `REDDIT_LINKS`. Glorp shields us from the differences between different SQL dialects. Note how id is designated to be a primary key. The serial type will result in an SQL sequence being used.

```
descriptorForRedditLink: aDescriptor
  | table |
  table := self tableNamed: 'REDDIT_LINKS'.
  aDescriptor table: table.
  (aDescriptor newMapping: DirectMapping) from: #id to: (table fieldNamed: 'id').
  (aDescriptor newMapping: DirectMapping) from: #url to: (table fieldNamed: 'url').
  (aDescriptor newMapping: DirectMapping) from: #title to: (table fieldNamed: 'title').
  (aDescriptor newMapping: DirectMapping) from: #created to: (table fieldNamed: 'created').
  (aDescriptor newMapping: DirectMapping) from: #points to: (table fieldNamed: 'points')
```

The third and final step is the actual descriptor describing the mapping between the class model `RedditLink` and the table `REDDIT_LINKS`. In this simple case, direct mappings are used between attributes and fields. As we will see in the next sections, Glorp is now ready to do its work.

4. RedditDatabaseResource

Let's assume you installed and configured PostgreSQL on some machine and that you created some database there. We now have to specify how Glorp has to connect to PostgreSQL. We do this using an `Object` subclass called `RedditDatabaseResource`, where we add some class methods (as well as a class variable called `DefaultLogin`).

```
login
  DefaultLogin ifNil: [ DefaultLogin := self createLogin ].
  ^ DefaultLogin

login: aLogin
  "see #createLogin for an example of how to create a Login object"

  DefaultLogin := aLogin
```

```
createLogin
  ^ (Login new)
      database: PostgreSQLPlatform new;
      username: 'svc';
      password: 'secret';
      connectString: 'localhost:5432_playground';
      yourself
```

The username and password speak for themselves, the connectString contains the hostname, port number and database name (after an underscore). Glorp accesses a database through sessions. A session is most easily started from a descriptor system given a login as argument, that's what we do in the `#session` helper class method.

```
session
  ^ RedditSchema sessionForLogin: self login

createTables
  "self createTables"
  "This has to be done only once, be sure to set #login"

  |session |
  session := self session.
  session accessor login; logging: true.
  session inTransactionDo: [ session createTables ].
  session accessor logout
```

Glorp can even help us to create our `REDDIT_LINKS` table, that is what the `#createTables` class method does. So we truely don't have to use any SQL! The flow should be familiar: get a session, login, do some work in a transaction and logout. By setting logging to true, the generated SQL statements will be printed on the Transcript (comment this out for production use).

5. RedditDatabaseTests

With our `RedditSchema` descriptor system and our `RedditLinksDatabaseResource` we are now ready to test the persistency of our model. We create another `TestCase` subclass named `RedditDatabaseTests`. These tests need an instance variable called `session` to hold the Glorp session, as well as `#setUp` and `#teardown` methods.

```
setUp
  session := RedditDatabaseResource session.
  session accessor logging: true; login

tearDown
  session accessor logout

testQuery
  | links |
  links := session readManyOf: RedditLink.
  links do: [ :each |
    each assertContractUsing: self.
    self assert: (each isKindOf: RedditLink) ]

testCreate
  | link url title id |
  url := 'http://www.seaside.st'.
  title := 'Seaside Unit Test'.
  link := RedditLink withUrl: url title: title.
  session inUnitOfWorkDo: [ session register: link ].
  id := link id.
  self assert: id notNil.
  session reset.
  link := session readOneOf: RedditLink where: [ :each | each id = id ].
  link assertContractUsing: self.
  self assert: link url = url.
  self assert: link title = title.
  session delete: link

testUpdate
  | link url title id |
  url := 'http://www.seaside.st'.
  title := 'Seaside Unit Test'.
  link := RedditLink withUrl: url title: title.
  session inUnitOfWorkDo: [ session register: link ].
  id := link id.
  session inUnitOfWorkDo: [ session register: link. link voteUp ].
  session reset.
  link := session readOneOf: RedditLink where: [ :each | each id = id ].
  self assert: link points = 1.
  session delete: link
```

Our first test reads all RedditLinks from the database, making sure they are valid and of the expected type. Querying doesn't have to be done in a unit of work or transaction. The second test creates a new RedditLink and then registers it with the session inside a unit of work. This will effectively save the object in the database. The id of the RedditLink will have a value afterwards. Next we reset the session and query the RedditLink with the known id. After making sure that what we put in got out of the database we delete the object. The third test checks if updating an existing persistent object works as expected. Note that the actual modification, the #voteUp, has to be done inside a unit of work to a registered object for it to be picked up by Glorp.

6. RedditSession

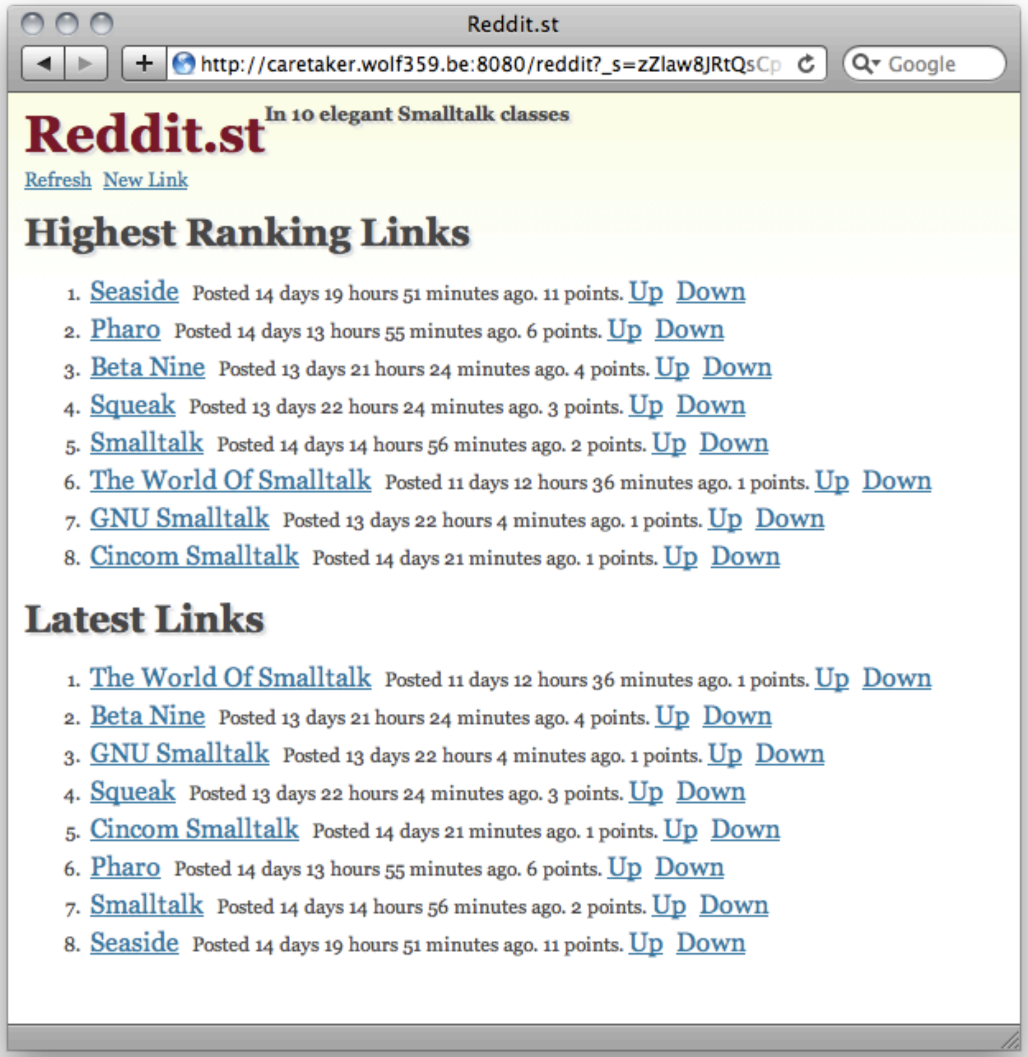
We are almost ready to start writing the GUI of our actual web application. Seaside web applications always have a session object that keeps the application's state during the user's interaction with it. We need to extend that session with a database session. Our WASession subclass RedditSession has an instance variable called glorpSession to hold a Glorp session to the database.

```
glorpSession
  glorpSession ifNil: [ glorpSession := self newGlorpSession ].
  glorpSession accessor isLoggedIn ifFalse: [ glorpSession accessor login ].
  ^ glorpSession

newGlorpSession
  | session |
  session := RedditDatabaseResource session.
  "session accessor logging: true."
  ^ session

unregistered
  super unregistered.
  self teardownGlorpSession

teardownGlorpSession
  self glorpSession logout
```

Note how we are using lazy initialization in the #glorpSession accessor. In #newGlorpSession we're making use of our RedditDatabaseResource. The #unregistered is a hook called by Seaside whenever a session expires, we use it clean up our Glorp session by doing a log out.

7. WAReddit

We can finally start with our web app itself. Figure 1 shows the main page of the Reddit.st app. There are four sections in this page: a header or title section, some action links, a list of some of the highest or top ranking links and a list if some of the latest or most recent links.



Reddit.st In 10 elegant Smalltalk classes

Refresh New Link

Highest Ranking Links

1. Seaside Posted 14 days 19 hours 51 minutes ago. 11 points. Up Down
2. Pharo Posted 14 days 13 hours 55 minutes ago. 6 points. Up Down
3. Beta Nine Posted 13 days 21 hours 24 minutes ago. 4 points. Up Down
4. Squeak Posted 13 days 22 hours 24 minutes ago. 3 points. Up Down
5. Smalltalk Posted 14 days 14 hours 56 minutes ago. 2 points. Up Down
6. The World Of Smalltalk Posted 11 days 12 hours 36 minutes ago. 1 points. Up Down
7. GNU Smalltalk Posted 13 days 22 hours 4 minutes ago. 1 points. Up Down
8. Cincom Smalltalk Posted 14 days 21 minutes ago. 1 points. Up Down

Latest Links

1. The World Of Smalltalk Posted 11 days 12 hours 36 minutes ago. 1 points. Up Down
2. Beta Nine Posted 13 days 21 hours 24 minutes ago. 4 points. Up Down
3. GNU Smalltalk Posted 13 days 22 hours 4 minutes ago. 1 points. Up Down
4. Squeak Posted 13 days 22 hours 24 minutes ago. 3 points. Up Down
5. Cincom Smalltalk Posted 14 days 21 minutes ago. 1 points. Up Down
6. Pharo Posted 14 days 13 hours 55 minutes ago. 6 points. Up Down
7. Smalltalk Posted 14 days 14 hours 56 minutes ago. 2 points. Up Down
8. Seaside Posted 14 days 19 hours 51 minutes ago. 11 points. Up Down

Figure 1: Main page of Reddit.st's web app

We create a `WAComponent` subclass called `WAReddit`. This will become our central or root web app component. We start by writing the rendering methods.

```
renderContentOn: html
  html heading: 'Reddit.st'.
  html heading level: 3; with: 'In 10 elegant Smalltalk classes'.
  self renderActionsOn: html.
  self renderHighestRankingLinksOn: html.
  self renderLatestLinksOn: html

renderActionsOn: html
  html paragraph: [
    html anchor callback: [ ]; with: 'Refresh'.
    html anchor callback: [ self inform: 'Not yet implemented' ]; with: 'New Link' ]

renderHighestRankingLinksOn: html
  html heading level: 2; with: 'Highest Ranking Links'.
  html orderedList: [
    self highestRankingLinks do: [ :each | self renderLink: each on: html ] ]

renderLatestLinksOn: html
  html heading level: 2; with: 'Latest Links'.
  html orderedList: [
    self latestLinks do: [ :each | self renderLink: each on: html ] ]

renderLink: link on: html
  html listItem: [
    html anchor url: link url; title: link url; with: link title.
    html text: ' Posted ', (self class durationString: link posted), ' ago. '.
    html text: link points asString, ' points. '.
    html anchor callback: [ self voteUp: link ]; title: 'Vote this link up'; with: 'Up'.
    html space.
    html anchor callback: [ self voteDown: link ]; title: 'Vote this link down'; with: 'Down' ]
```

Starting with the main `#renderContentOn:` method, the rendering of each section is delegated to its own method. Note how `#renderLink:on:` is used 2 times. For now, we're not yet implementing the 'New Link' action. Our rendering methods depend on 5 extra methods: `#highestRankingLinks`, `#latestLinks`, the class method `#durationString:` and the actions methods `#voteUp:` and `#voteDown:`. Only the first three are needed to render the page itself.

```
highestRankingLinks
  | query |
  query := (Query readManyOf: RedditLink)
     orderBy: [ :each | each points descending ];
     limit: 20;
     yourself.
  ^ self session glorpSession execute: query

latestLinks
  | query |
  query := (Query readManyOf: RedditLink)
     orderBy: [ :each | each created descending];
     limit: 20;
     yourself.
  ^ self session glorpSession execute: query

durationString: duration
  ^ String streamContents: [ :stream | | needSpace printer |
      needSpace := false.
      printer := [ :value :word |
        value isZero ifFalse: [
          needSpace ifTrue: [ stream space ].
          stream nextPutAll: (value pluralize: word).
          needSpace := true ] ].
      printer value: duration days value: 'day'.
      printer value: duration hours value: 'hour'.
      printer value: duration minutes value: 'minute'  ]
```

In `#highestRankingLinks` and `#latestLinks`, we explicitely build up and execute `Query` objects with some more advanced options. For duration string conversion we use the powerful `#pluralize` method. With these in place we can already render the page. Since we did not yet add any CSS styling, the result will look rather dull.

```
voteUp: link
  self session glorpSession inUnitOfWorkDo: [ :session |
    session register: link.
    link voteUp ]

voteDown: link
  self session glorpSession inUnitOfWorkDo: [ :session |
    session register: link.
    link voteDown ]
```

Voting links up or down is trivial, like with our database test, we only have to make sure to do the object modifications inside a Glorp unit of work and the actual SQL update will be done automatically.

8. RedditFileLibrary

To style our web app, we'll be reusing the CSS file from Reddit.lisp. This CSS code references one small GIF for its background gradient. We need to make sure our application makes use of the CSS file and that we serve the actual files. Seaside can serve these files in a couple of ways, we'll be using the FileLibrary approach. This is a class where each resource served is implemented as a method.

Our `WAFileLibrary` subclass `RedditFileLibrary` will thus have 2 methods: `#mainCss` and `#bgGif`, returning a string and bytes respectively. These are long methods which are not our main focus so we do not list them here. Based on some naming conventions, Seaside will figure out what mime types to use.

```
updateRoot: anHtmlRoot
  super updateRoot: anHtmlRoot.
  anHtmlRoot title: 'Reddit.st'.
  anHtmlRoot stylesheet url: (RedditFileLibrary urlOf: #mainCss)
```

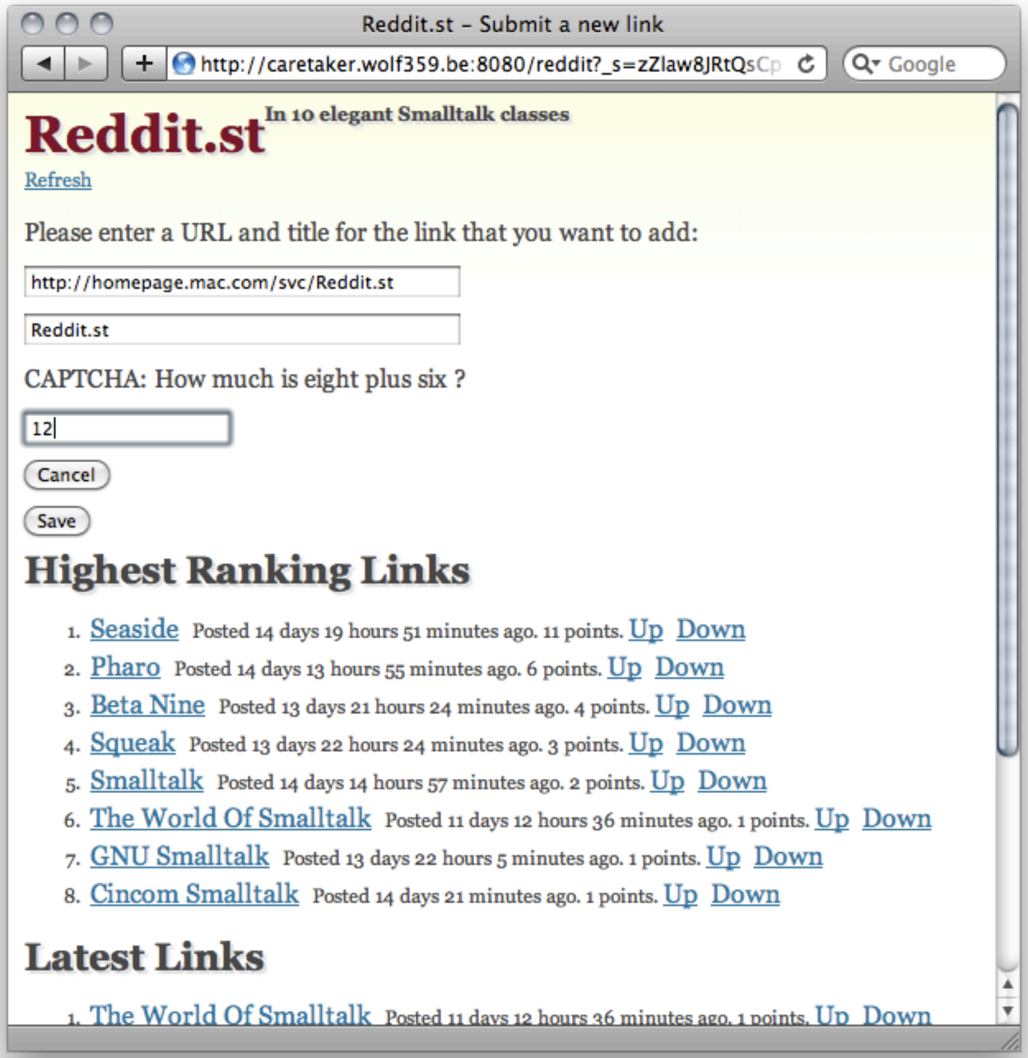
By implementing the `#updateRoot:` hook method on `WAReddit`, we can set our page title and CSS. Note again how everything happens in Smalltalk. To install a Seaside application, a class side `#initialize` method is typically used.

```
initialize
  (WAAdmin register: self asApplicationAt: 'reddit')
     preferenceAt: #sessionClass put: RedditSession;
     addLibrary: RedditFileLibrary
```

We register our application under the handler 'reddit' so its URL will become something like 'http://localhost:8080/reddit'. Then we tell it to use our custom session class and finally add our file library. We now have a nicely styled, working web app.

9. WARedditLinkEditor

One of Seaside's main advantages over other web application frameworks is its support for components. Especially for large and complex projects this makes a huge difference. We'll be introducing a new component to allow the user to enter the necessary information when adding a new link. Consider the difference between figure 1 and figure 2: when the user clicks the 'New Link' anchor, we'll add an editor just below (while hiding the 'New Link' anchor). The editor will have its own 'Save' and 'Cancel' buttons. Both of these will dismiss the editor, saving or cancelling the new link.


Reddit.st - Submit a new link

http://caretaker.wolf359.be:8080/reddit?_s=zZlaw8JRtQsCp

Reddit.st In 10 elegant Smalltalk classes

Refresh

Please enter a URL and title for the link that you want to add:

http://homepage.mac.com/svc/Reddit.st

Reddit.st

CAPTCHA: How much is eight plus six ?

12

Cancel

Save

Highest Ranking Links

1. Seaside Posted 14 days 19 hours 51 minutes ago. 11 points. Up Down
2. Pharo Posted 14 days 13 hours 55 minutes ago. 6 points. Up Down
3. Beta Nine Posted 13 days 21 hours 24 minutes ago. 4 points. Up Down
4. Squeak Posted 13 days 22 hours 24 minutes ago. 3 points. Up Down
5. Smalltalk Posted 14 days 14 hours 57 minutes ago. 2 points. Up Down
6. The World Of Smalltalk Posted 11 days 12 hours 36 minutes ago. 1 points. Up Down
7. GNU Smalltalk Posted 13 days 22 hours 5 minutes ago. 1 points. Up Down
8. Cincom Smalltalk Posted 14 days 21 minutes ago. 1 points. Up Down

Latest Links

1. The World Of Smalltalk Posted 11 days 12 hours 36 minutes ago. 1 points. Up Down

Figure 2: Reddit.st with the New Link editor open

How is Seaside's component model powerful ? As we will see next, the component is written without any knowledge of where it will be used. Its validation logic is independent. It is used just by embedding it and by wiring it to its user in a simple way. The subcomponent functions indepedently from its embedding parent while each keeps its own state: whether the component is visible or not, you can keep on voting links up or down, and doing so will not alter the contents of the component.

To prove our point, we'll be using yet another component inside our link editor: a simple CAPTCHA component. This will be implemented in the final section, but used here as a black box.

The first step is to make the necessary additions and modifications to `WAReddit` to accommodate the link editor component. We add an instance variable called `linkEditor` with its accessors.

```
renderContentOn: html
  html heading: 'Reddit.st'.
  html heading level: 3; with: 'In 10 elegant Smalltalk classes'.
  self renderActionsOn: html.
  self linkEditor notNil ifTrue: [ html render: self linkEditor ].
  self renderHighestRankingLinksOn: html.
  self renderLatestLinksOn: html

renderActionsOn: html
  html paragraph: [
    html anchor callback: [ ]; with: 'Refresh'.
    self linkEditor isNil ifTrue: [
      html anchor callback: [ self showNewLinkEditor ]; with: 'New Link' ] ]

showNewLinkEditor
  self linkEditor: WARedditLinkEditor new.
  self linkEditor onAnswer: [ :answer |
    answer ifTrue: [
      self session glorpSession inUnitOfWorkDo: [ :session |
        session register: self linkEditor createLink ] ].
    self linkEditor: nil ]

children
  ^ self linkEditor notNil
      ifTrue: [ Array with: self linkEditor ]
      ifFalse: [ super children ]
```

There are 2 possible states: either we have a link editor subcomponent or not. So the main `#renderContentOn:` method conditionally asks the link editor to render itself. Likewise, in `#renderActionsOn:` the 'New Link' anchor is only rendered when there is no link editor yet.

In the `#showNewLinkEditor` action method we instanciate our subcomponent and hook it up. We could have reused just one instance, creating a new one is easier and clearer. The wiring is done by supplying a block to `#onAnswer:`. A component can answer a value, in our case true or false for save or cancel respectively. So when the link editor answers true, we save a new link object and hide the editor.

In Seaside, the `#children` method is a hook method that has to be implemented to list all subcomponents. Again this happens conditionally.

We can now implement the component itself: `WARedditLinkEditor` is a subclass of `WAComponent` with 3 instances variables and their accessors: url, title and capcha.

```
renderContentOn: html
  html
    form: [
      html paragraph: 'Please enter a URL and title for the link that you want to add:'.
      html textInput size: 48; title: 'The URL of the new link'; on: #url of: self.
      html textInput size: 48; title: 'The title of the new link'; on: #title of: self.
      html render: self captcha.
      html submitButton on: #cancel of: self.
      html submitButton on: #save of: self ]

initialize
  super initialize.
  self url: 'http://'; title: 'title'; captcha: WARedditCaptcha new

children
  ^ Array with: self captcha

updateRoot: anHtmlRoot
  super updateRoot: anHtmlRoot.
  anHtmlRoot title: 'Reddit.st - Submit a new link'.
  anHtmlRoot stylesheet url: (RedditFileLibrary urlOf: #mainCss)

cancel
  self answer: false
```

```
save
  self isUrlMissing ifTrue: [ ^ self inform: 'Please enter an URL' ].
  self isTitleMissing ifTrue: [ ^ self inform: 'Please enter a title' ].
  self captcha isSolved ifFalse: [ ^ self inform: 'Please answer the correct sum using digits' ].
  self isUrlValid ifFalse: [ ^ self inform: 'The URL you entered did not resolve' ].
  self answer: true

isTitleMissing
  ^ self title isNil or: [ self title isEmpty or: [ self title = 'title' ] ]

isUrlMissing
  ^ self url isNil or: [ self url isEmpty or: [ self url = 'http://' ] ]

isUrlValid
  ^ (WebClient httpGet: self url) isSuccess

createLink
  ^ RedditLink withUrl: self url title: self title
```

Most of the code should be familiar by now. New is how `#cancel` and `#save` use `#answer:` to return to whoever called upon this component. Before `#save` returns successfully, a number of validation tests are done. When one of these tests fails, a message is shown and the operation is aborted. The `#isUrlValid` method actually tries to resolve the URL. Finally, `#createLink` instantiates a new `RedditLink` instance based on the valid fields entered by the user. Note how the CAPTCHA is used as a true component.

10. WARedditCaptcha

The last and simplest web component is a CAPTCHA that presents a simple addition in words. This component does not need answer logic. `WARedditCaptcha` is again a `WAComponent` subclass with the following instance variables and accessors: x, y and sum.

```
renderContentOn: html
  self x: 10 atRandom.
  self y: 10 atRandom.
  html paragraph: 'CAPTCHA: How much is ', self x asWords, ' plus ', self y asWords, ' ?'.
  html textInput title: 'This functions as a CAPTCHA, type the answer using digits'; on: #sum of: self

initialize
  super initialize.
  self x: 0; y: 0; sum: 0

isSolved
  ^ self sum asInteger = (self x + self y)
```

Each time the CAPTCHA is rendered, x and y get a new random value between 1 and 10. Next, the addition is presented in words. The `#isSolved` method checks if the user answered correctly.

11. Resources

The source code discussed in this article is available from a SqueakSource project called ADayAtTheBeach. Look for the package called **Reddit**.

Lines of source code is not a good measure for the size or complexity of a Smalltalk application. Reddit.st consists of 10 classes for a total of 8 class methods and 75 instance methods. More than half are just one (1) line long, the rest averages just a few lines.

If you want to try out **Reddit.st**, you can visit http://caretaker.wolf359.be:8080/reddit where the application is running live (along with several other Seaside examples).

These are some pointers to the technologies used in this example:

- Seaside - Smalltalk's most advanced web application framework
- Glorp - Smalltalk's Generic Lightweight Object-Relational Persistence framework
- Pharo - A clean and professional open source Smalltalk implementation
- Squeak - The original open source Smalltalk implementation