Sourcetalk

Smalltalk Code Repository

Andrea Quadri
Supervised by Lukas Renggli
Software Composition Group
University of Bern, Switzerland

December 2008

Abstract

SqueakSource is a highly successful source code repository for Squeak based on the distributed source code management system Monticello. Monticello is not designed to be cross platform. Moreover SqueakSource is old and not up-to-date with web technology. SqueakSource is not extensible and it was built with an old Seaside version. It does not use Magritte and is not Web 2.0 conform. Sourcetalk is based on Monticello 2. It uses Seaside 2.8 for the view, Magritte for extensibility and Pier for the integrated wiki.

Acknowledgments

I would like to thank my supervisors Prof. Dr. Oscar Nierstraz and Lukas Renggli for their support, guidance and review of my work. In addition my thanks go to Samuel Morello for the graphical design of the site. Last but not least, I would like to thank all people who tested Sourcetalk and reported bugs.

Contents

Chapter 1

Introduction

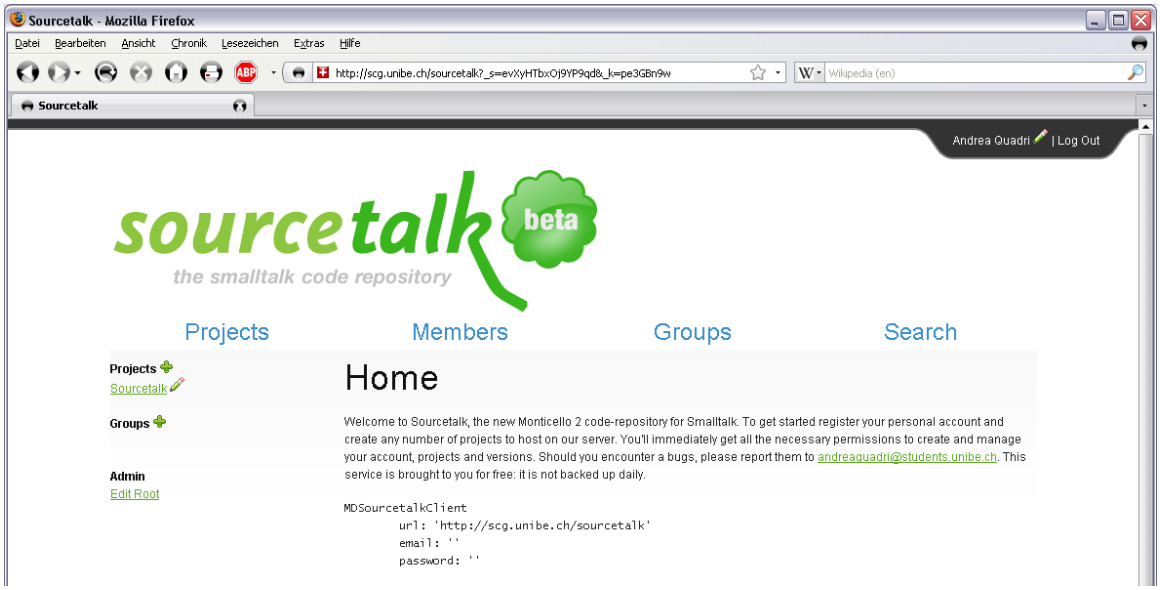

Figure 1.1: Sourcetalk code repository in Firefox.

1.1 Context

Versioning systems help groups of developers to work together. By using a versioning system the developers have more possibilities to manage their code, such as restoring old versions, branching and merging source code, finding changes, managing releases

and so on. Monticello 1 provides all these features but has the drawback of only supporting Squeak and GemStone. It always takes a full snapshot every time you commit. Because of the introduction of *slices*, Monticello 2 does not rely any more solely on packages. A *slice* is a set of elements - an arbitrary slice of code in the image. Squeak itself removed packages with well-defined and relatively stable boundaries, therefore Monticello 2 is adopting this representation. There are different kind of *slices* for packages, change sets, modules and others. With the change to *slices* Monticello 2 also aims to be more platform independent than the previous one.

Monticello 2 does not come with remote repository support. The current practice is to set up an ad-hoc solution like a shared network drive. Unfortunately this is not easily possible, especially when projects have to be shared among large groups of developers and when user management is crucial. Sourcetalk provides an easy-to-use web interface that allows one to create and manage projects. This includes managing permissions and providing per project RSS feeds. Registering as a new member to Sourcetalk and creating a new project is a matter of a few clicks. Source code can then be committed and loaded from within any Smalltalk image with Monticello 2 support.

1.2 Problem

There is no interface to the new Monticello 2 on the web. Therefore every team, especially open source teams, wanting to use a versioning system is limited to Monticello 1. Furthermore development teams are limited to the Squeak dialect of Smalltalk, because Monticello 1 only handles source code of Squeak and GemStone.

Why did we not just refactor SqueakSource to use Monticello 2? The model of Monticello 2 is completely different from that of Monticello 1 and would therefore require significant changes in the user-interface. Besides SqueakSource is based on an ancient version of Seaside and therefore there are no RESTful URLs. RESTful URL means that a model, like projects, RSS feeds or subpages, get their own constant URL, so it can be bookmarked. Furthermore the tag system in SqueakSource allowed users to tag their projects only with a given set of tags. So it seemed obvious to start from scratch, but use the old SqueakSource as reference, since it was that successful.

1.3 Solution

To build the new web interface for Monticello 2 we decided to aim for the following goals:

Object-Oriented Design. Object-oriented design, which is the strength of Smalltalk, is an obvious choice for achieving extensibility. We applied patterns like Singleton, Constructor Parameter Method, Debug Printing Method and Setting Method from the books *Smalltalk Best Practice Patterns* [2] and *The Design Patterns Smalltalk Companion* [1].

Usability. The idea was to make sure that creating and managing projects is intuitive and only provides the functionality absolutely necessary to get things done. As we learned from SqueakSource, people are quickly overwhelmed by complicated features. The amount of clicks necessary for one goal is kept as low as possible. RESTful URLs provide more usability by allowing users to bookmark pages and pass around links.

Extensibility. Certainly people will want to extend Sourcetalk, therefore using object-oriented design and readable code is crucial. Tests are also helpful, therefore we made sure that every model method is well tested following the suggestions of *Squeak by Example* [3]. Furthermore this report provides the necessary background to understand Sourcetalk.

Security. Not necessarily every project hosted on Sourcetalk is an open source project. Some projects are hidden, or read-only. There are many ways to access Sourcetalk data either by Monticello 2, the Sourcetalk web application or the RSS feeds, therefore security of projects and passwords is essential.

Open Source. Sourcetalk is released under the MIT license which grants unrestricted rights to copy, modify, and redistribute as long as the original copyright and license terms are retained.

1.4 Overview

Chapter 2: Applied Technology lists all the used frameworks for Sourcetalk and explains what they contributed to the Sourcetalk project.

Chapter 3: Object Model explains the structure of the model, view and server. It is documented with UML diagrams and takes a look at important parts within those structural parts.

Chapter 4: Security Concerns shows the choices made during the development regarding the security of the system, especially the Security of passwords and project access. This includes the authorizations for reading, writing and editing a project.

Chapter 5: Extensions shows you what would be the best way to extend Sourcetalk.

Chapter 6: Conclusion summarizes this documentation.

Chapter 2

Applied Technology

2.1 Monticello 2

Monticello 2 is a open source versioning system and the successor of Monticello, a popular piece of software in the community. Monticello 2 breaks Smalltalk systems down into their fundamental units, which are called *elements*. Those *elements* represent classes, class comments, methods, instance variables, class variables, class instance variables and pool Imports. A *variant* captures the state of a particular element. A *version* of an element associates a variant of a particular element with the ancestry of that element. *Slices* group *elements* together. The *slices* are independent of one another and can overlap. A given *element* may be in any number of *slices* or no slice at all. *Snapshots* capture the state of a slice. A *hashstamp* is a unique identifier given to each *version* or *slice*. This means that Monticello 2 can be used to version arbitrary snippets of code, unlike Monticello. Sourcetalk is a web interface for Monticello 2. Each project in Sourcetalk gets a Monticello 2 repository.

2.2 Seaside

Seaside is a framework for developing sophisticated web applications in Smalltalk [4]. It provides a layered set of abstractions over HTTP and HTML that let you build highly interactive web applications quickly, reusably and maintainably. The view of the Sourcetalk implementation was written in Seaside, using version `Seaside2.8a1–lr.578`. The Sourcetalk model is independent of the view, and therefore could be easily replaced with a different end-user interface.

2.3 Magritte

Magritte is a dynamic meta-description framework which can be used to automatically build different views, editors, and reports of described objects [6]. Every model object in Sourcetalk has its own description so a Seaside component for editing is built automatically. This feature is mostly used when instantiating or editing objects. Required or read-only values and selectionable entries are declared in the description. Some minor extensions have been written to get all the functionality wanted, such as differentiation between creation and editing or access to more information for superusers. Furthermore Magritte was used to display reports as it provides automatic search, sorting and filtering functionality. Sourcetalk needs the version `Magritte–All–lr.280`.

2.4 Pier

Every member, project or group in Sourcetalk has its own wiki page provided by Pier [5]. So each user can edit his own wiki page or the one of projects or groups he is an admin of. This gives you the freedom to enter custom information and helps you to represent yourself, your projects and your groups. The version used is `Pier–All–lr.309`.

Chapter 3

Object Model

3.1 Model

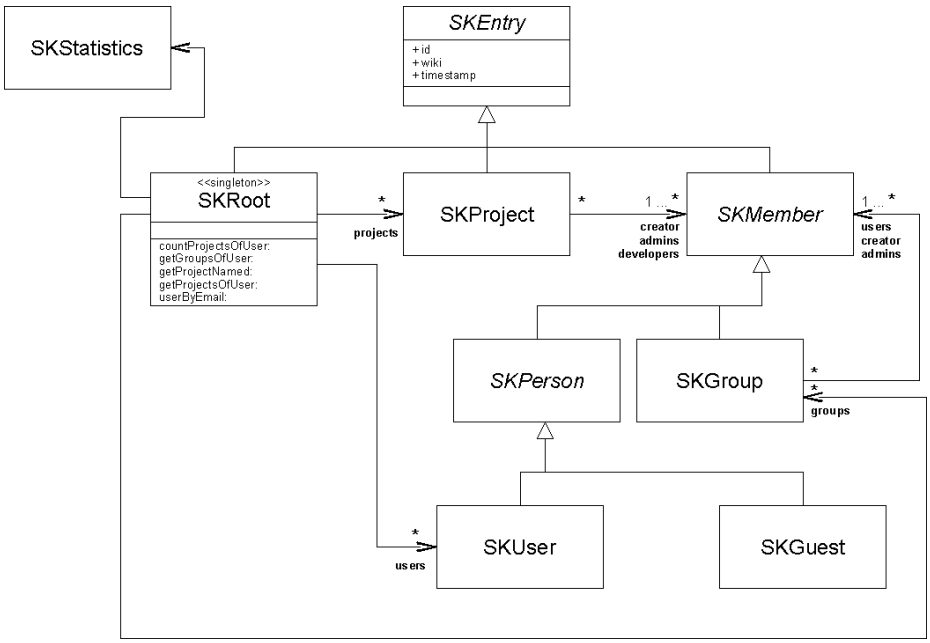

Figure 3.1: UML diagram of the Sourcetalk model.

The UML diagram in Figure 3.1 shows the most important parts of the model. Every class in `Sourcetalk` has the prefix `SK` set as namespace. The `SKRoot` class is a globally accessible singleton and the root of the complete model. It keeps track of

newly created users, projects and groups and provides additional search functionality to the SKStatistics class and the view. An SKGroup has SKMembers as creator, admins or users, but must have at least one admin and a creator. An SKProject contains instances of SKMember as admins, developers and one creator. The SKEntry is an abstract class that provides each subclass with a unique id and instance variables for tags, wiki and creation timestamp. An SKMember is an abstract superclass of the SKGroup and the SKPerson class. This SKPerson class is another abstract superclass, which is necessary because both subclasses SKGuest and SKUser are session users. SKGuest is the default user of a session but has no rights. There is one instance of SKGuest referenced by the SKRoot.

The SKStatistics class keeps track of the ten newest projects, users and groups. This class can easily be extended to keep track of more things as it gets notified of changes to the model.

Because of the importance of the model, all functionality in this category is covered by test cases.

3.2 View

All Seaside components and RSS readers are in this category. The SKRootComponent seen in Figure 3.2, is the root of the site. It is a subclass of WAComponent and it renders the four parts of the web page, namely the login at the top, the main navigation called menu, the userMenu and the content in which every page is displayed. Every view that is displayed within the content knows about its parent and ultimately about the root. The reason why is because they all inherit from SKComponent. On the class side there is a method SKComponent class≫on: aComponent which triggers the instance side method SKComponent ≫initalizeOn: aComponent, which in turn sets the chosen component as the parent.

The SKSession keeps track of the current user and is initialized with an SKGuest instance. The SKLoginView manages the registration, login, logout and forgotten password functionality and dispatches session changes to SKSession. SKSession is a subclass of WASession, the Seaside implementation of a session handler.

The SKModelComponent is an abstract superclass for SKProjectView, SKGroupView and SKUserView. The edit method of each of those classes calls the represented model element as components, based on their Magritte descriptions. The SKTableComponent is an abstract superclass of SKProjectsView, SKGroupsView, SKMembersView and

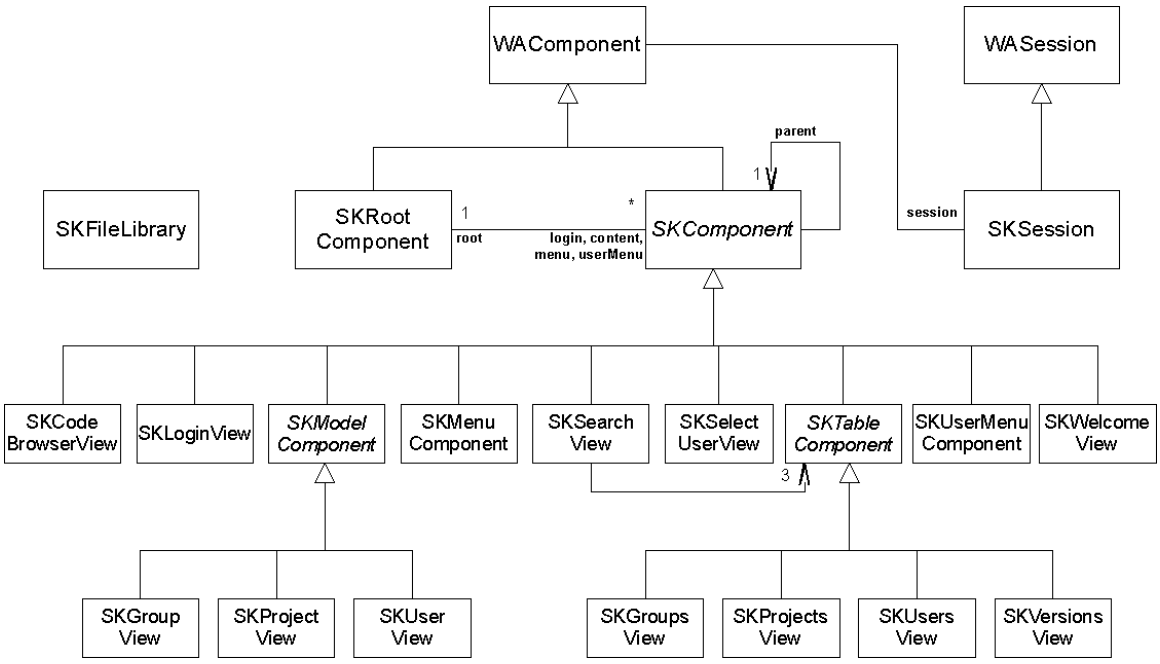

Figure 3.2: UML diagram of the Sourcetalk-View category without RSS.

SKVersionsView. They each represent a table but have different content. They use the MAReport component from the Magritte framework. Such a table allows one to automatically filter, sort, and search. Furthermore it has a batching functionality which can be used to set a maximum number of table lines per site. The Searching functionality of MAReport is essential because the SKSearchView forwards a given search input to the SKGroupsView, SKProjectsView and the SKUsersView. Afterwards the MAReport component of each class displays only the entities that are matching the given search input.

All components have access to the SKSession class, which provides the currently logged in user. The SKFileLibrary is also registered and provides the static needed files for the web interface, namely the CSS and PNG files.

The RSS feed classes use SKRssFeed as their common superclass, seen in Figure 3.3, which itself is a subclass of RRComponent. This comes from the RSRSS package, a Seaside extension to create RSS feeds. The subclasses of SKRssFeed populate the feeds with the model.

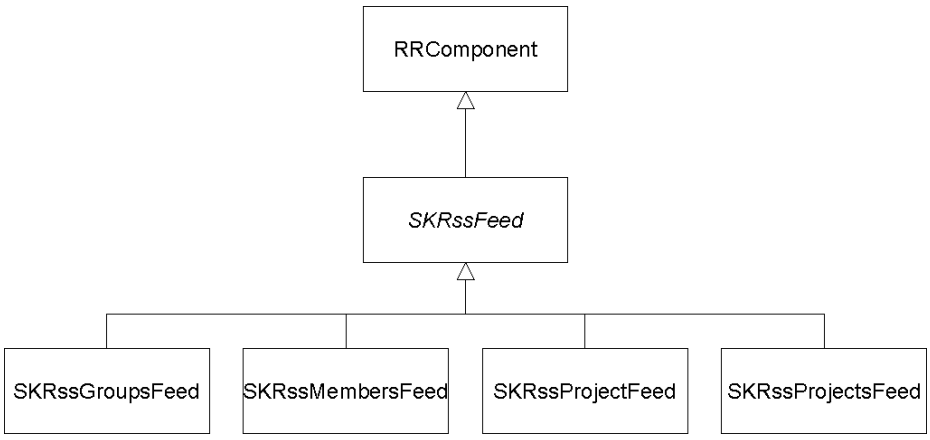

Figure 3.3: UML diagram of the RSS classes in Sourcetalk.

3.3 Server

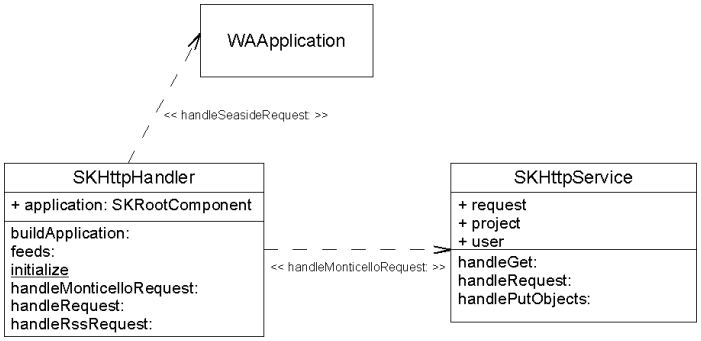

Figure 3.4: UML diagram of the Sourcetalk server category.

The server category contains only two classes. The class SKHttpHandler registers Sourcetalk in the WADispatcher with SKHttpHandler class≫initialize, as seen in Figure 3.4. This method calls SKHttpHandler≫buildApplication which registers the SKFileLibrary, the SKSession, both seen in Figure 3.2, and the SKRootComponent as a Seaside entry point. Moreover the SKHttpHandler detects the kind of request that was sent to the Sourcetalk server. It decides what to answer: either an RSS feed, a Monticello 2 response or the Sourcetalk application.

1. If the entered URL respectively the request ends with `.rss` the SKHttpHandler≫

`handleRssRequest: aRequest` will be called, which checks if such a feed exists and then dynamically composes and returns it.

2. If the request comes from a Monticello 2 browser it will be handled by `SKHttp-Handler>>handleMonticelloRequest: aRequest` that hands the responsibility to the `SKHttpService` class. Handling uploads and downloads of code, if the user has the required rights, is the main function of this class. The `SKHttpService>>handleRequest: aRequest` gets the Request from `SKHttpHandler`. All the handler methods in `SKHttpService` decode the request and hand it over to the Monticello 2 model. For example `SKHttpService>>handlePutObjects: message` will be used for upload and `SKHttpService>>handleGet: message` for download.

3. In all other cases the application will pass the request on to the Seaside web application handler.

Chapter 4

Security Concerns

Security is a crucial part of Sourcetalk. Every aspect of the security system in the model has been tested in one of the 57 test cases in the `Sourcetalk–Test` package. In the following sections we explain how rights to read versions, commit code and edit projects are determined. However superusers are always allowed to read, write or edit in any group, user or project. Only registered users can be promoted to superusers. This can be accomplished only by people who set up the Sourcetalk system and therefore have physical access to the deployed system.

4.1 Passwords

Thanks to Magritte, every entered password is represented in the web interface only as a String of *, independently of whether you register, log in or change your password. Furthermore each password is saved in `SKUser≫password: aString` as a one way hash value. The hash value is based on the secure hash algorithm (SHA) implementation of Squeak. Passwords are verified in `SKUser≫verifyPassword: aString`. The input that has to be verified is first hashed and then compared to the stored hash value of the password. The accessor for `User≫password` returns a fixed String of length eight. Thus the current password of a user is not saved itself and is therefore safe even from people that have physical access to the server.

If a user of Sourcetalk forgets the password, he has to enter his email address. Afterwards he will receive a link by email that he can use to navigate to the web application and set a new password.

Superusers are not allowed to reset their password in this manner. This has been implemented to prevent an unauthorized user from getting control of a superuser

account. If necessary they can reset their password with physical access to the system and change it directly.

4.2 Visibility of Projects

```
SKProject≫isVisibleBy: aUser

"Visible means seeing the project on the site"

        ^ aUser isSuperuser
                or: [ self visibility = #public
                or: [ self visibility = #protected
                or: [ (aUser isWithin: self developers)
                or: [ aUser isWithin: self admins ] ] ] ]
```

If a project is set to *public* or *protected* everyone can see the project from the web and access the source stored with Monticello 2.

Private means that a project can only be seen on Sourcetalk and its code can only be accessed by project admins or developers. Registered users who are not part of the project do not see the project. This is not only true for the project page but also for the RSS feed of the project, a Monticello 2 request or any other listing of projects.

4.3 Writeablity of Projects

```
SKProject≫isWritableBy: aUser

"Write means being able to write in the repository"

        ^ aUser isSuperuser
                or: [ self visibility = #public
                or: [ (aUser isWithin: self developers)
                or: [ aUser isWithin: self admins ] ] ]
```

If a project is set to *public*, everyone has the right to store source code within the project using Monticello 2. *Public* could be used for open source projects that

lack developers and want to encourage others to contribute. If the project is set to *protected*, only registered members of Sourcetalk are able to store source code within that project. *Protected* could be used for open source projects that wish bug fixes from the Sourcetalk community. If *private* is chosen only admins and developers of the project are able to store code within the project. *Private* should be the choice for non open source projects.

4.4 Editability

```
SKProject≫isEditableBy: aUser

"Edit means changing the SKProject settings and the project page"

        ^ aUser isSuperuser
                or: [ aUser isWithin: self admins ]
```

Only admins of a project or group have the right to edit or delete the project or group.

```
SKUser≫isEditableBy: aUser

        ^ aUser isSuperuser
                or: [ self = aUser ]
```

Every member of Sourcetalk has the right to edit or delete himself.

Chapter 5

Extensions

Sourcetalk is designed with extensibility in mind. There are many possible extensions. An import for projects from SqueakSource could be one. This could be done with a conversion to Monticello 2 or by extending Sourcetalk to use also the old Monticello. Furthermore the search functionality could be expanded to search not only for projects but for versions too. Visualizations of project timelines, ancestry or versions could be added to Sourcetalk. These are just some examples - many more are possible. Therefore some recommendations follow:

- If you change the model, you should add tests for the new functionality in the `Sourcetalk–Test` category. Be sure not to break existing tests.
- To add a new RSS feed, create a new class inheriting from `SKRssFeed` and place it in the `Sourcetalk–View` category. Then add it to `SKHttpHandler≫feeds`.
- Every view should inherit from `SKComponent` and be put in the category `Sourcetalk–View`. To have a view displayed in the content frame of the application it must be done like this: `self root content: (SKSearchView on: self root)`
- Extending statistics should be best done in the `SKStatistic` class.
- The `SKUserMenuComponent` class represents the user menu on the left. Additional user specific functionality should be placed here.
- If you would like to change the style, you find all the CSS files in the `SKFileLibrary`. Uploading new CSS, PNG or other static files is easily done in the Seaside configuration, by clicking on `configure` at the entry `files` and then click `configure`

next to `SKFileLibrary`. Now you will be able to select files you would like to add.

- Store additional information in a model element by adding a new instance variable and its accessors to the particular model and then adding a Magritte description on the class side of the model.

Chapter 6

Conclusion

Sourcetalk is a web interface for Monticello 2. Besides Monticello 2, Sourcetalk is based on Seaside, Magritte and Pier. By using Magritte, and a object-oriented design extensibility is assured. Thanks to new versions of Seaside and Pier, Sourcetalk is up to date and Web 2.0 ready. Every functionality of the model in Sourcetalk is tested. The root in the model is the connection between the model and the view. The view contains each component, the file library, the session and all RSS feeds. The server category is responsible for the distinction of requests sent to Sourcetalk. It returns then the wanted RSS feed, the web application or decodes the request and hand it over to the Monticello 2 model. Security for user passwords is guaranteed by using a secure hash algorithm (SHA) and therefore not giving anyone access to a current password. Administrators choose the permission settings for their projects and therefore are in charge of the visibility, writeablity and editability of their project.

Bibliography

[1] Sherman R. Alpert, Kyle Brown, and Bobby Woolf. *The Design Patterns Smalltalk Companion*. Addison Wesley, 1998.

[2] Kent Beck. *Smalltalk Best Practice Patterns*. Prentice-Hall, 1997.

[3] Andrew Black, Stéphane Ducasse, Oscar Nierstrasz, Damien Pollet, Damien Cassou, and Marcus Denker. *Squeak by Example*. Square Bracket Associates, 2007. http://SqueakByExample.org/.

[4] Stéphane Ducasse, Adrian Lienhard, and Lukas Renggli. Seaside: A flexible environment for building dynamic web applications. *IEEE Software*, 24(5):56–63, 2007.

[5] Lukas Renggli. Magritte — meta-described web application development. Master's thesis, University of Bern, June 2006.

[6] Lukas Renggli, Stéphane Ducasse, and Adrian Kuhn. Magritte — a meta-driven approach to empower developers and end users. In Gregor Engels, Bill Opdyke, Douglas C. Schmidt, and Frank Weil, editors, *Model Driven Engineering Languages and Systems*, volume 4735 of *LNCS*, pages 106–120. Springer, September 2007.