SeeThroughTalk - a collaborative image space

2nd period final report

1 Abstract

This document describes the development results of SeeThroughTalk - a collaborative Smalltalk. First, we will look back the concept of the system concisely. Next, we will explain what has been done until the end of this period. We have successfully implemented most of the features in the original proposal -Translucent Remote Desktop, Change Propagation, and UltraSwiki. However, we have found more improving points in the development. In the future work section, we will point out them and define a release plan.

2 Background

In Smalltalk, users are able to write programs in a tremendously interactive way. In that environment, they can freely browse and manipulate programming objects. Users send arbitrary enquiry messages to some part of the codes, and the system immediately answers the results. In general, it is very important for people to get instant feedback in order to develop interesting ideas. Therefore, this environment is ideal for many programmers. Most modern programming languages are trying to achieve such responsiveness in their IDE, but they do not reach the level of the original Smalltalk environment. However, the classic Smalltalk environment targeted only individual persons. It does not have direct supports for programming (or scripting) in collaboration. The Smalltalk virtual image is sometimes described as a 'lonely place'. In order to develop interesting ideas, we also need feedback not only from the system, but also from various people. Making the current Smalltalk programming environment sharable has been demanded for a long time.

Although some attempts to build team development environments have been done in commercial Smalltalks **[1] [2]**, these works have been too much influenced by traditional file-based languages and do not support creative interactions with other programming partners (after all, they only deal with packages - bunch of source codes, not people).

Thus, we would like to propose a true collaborative working space in Smalltalk. It focuses on human interactions and supports comfortable 'Ma's (Japanese word) in the collaborative environment. It encourages dynamic interactions while not preventing individuals' intensive activities

In order to realize this, SeeThroughTalk extends a popular desktop metaphor. It introduces an "overlapping" translucent desktop system to the existing Morphic GUI. Users can "see through" their partners' activities as live objects. SeeThroughTalk enables us to develop shared works collaboratively while not disturbing each individual's concentration.

In Squeak, there is a unique desktop sharing system, called Nebraska **[3]**. In that system, users can easily export their desktop and start interactive sessions with other remote users. The system is very useful for tiny tile scripting, but it is not so convenient for more complicated scripting or programming. In SeeThroughTalk, we will provide rich supportive user interfaces (avatars, air chatter, etc.) for collaboration. In addition, any programming changes made to a local image are automatically updated to the other users. It does not force users to lock the editing classes or methods explicitly. This feature eliminates the many tedious check-in/check-out steps that are thought to be essential to many traditional source code management systems. Basically, all images in a group have the same classes and methods at any period, but only their overlapped desktop focuses are different.

3 Project Summary

We are aiming at providing powerful sharable spaces for all Squeak users. In order to attain this, we have developed the idea of multi-layered sharable desktops.

The concept is pretty simple. It is just overlapped desktops in one place. However, those desktops are translucent so that users can "see through" other members' activities.

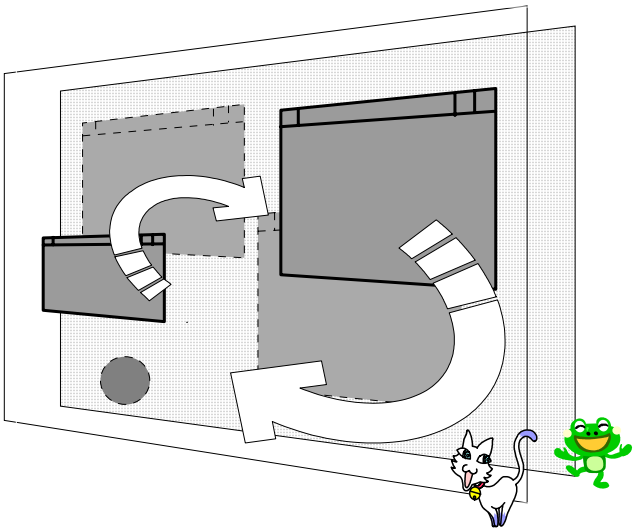

Figure1: Layered sharable desktops

The virtual space has four features:

1. Each person has his/her own panel (desktop)
2. Each panel is translucent, so you can see other panels
3. You can visit other panels
4. Every object is sharable

Although this interface seems not to be normal, we believe it has notable strong points. It enables users to see other activities consciously or unconsciously. Switching between private and collaborative modes is very easy. It also saves desktop areas remarkably, compared with the flat tile-style remote desktops.

The purpose of our project is to implement this conceptual interface in Squeak.

4 Development Results

4.1 Overlapping Remote Desktops

To implement remote desktop functions, we did not use Nebraska codes directly, because it has three major problems:

- The codes are so complicated and difficult to understand. They include bad name conversions, unclear class responsibilities, and debugging/profiling tweaks that were forgotten to be removed.
- It requires a rather big bandwidth per session. In SeeThroughTalk, since we need to establish multi remote desktop sessions between users, the traffic would be doubled (or even tripled). Apparently, we need to explore a new, more efficient protocol.
- It has a bug in dragging windows. In Nebraska, remote users cannot drag partner's windows, which should be supported in SeeThroughTalk.

We decided to rewrite a Nebraska-like remote-desktop system from scratch, but we also referred to Nebraska codes many times and arranged them when appropriate. Now the new system is clean and understandable. Moreover, it needs less bandwidth than Nebraska. The advantages that we have already implemented are as follows:

- Modularity

The components of the new system are carefully made to be independent. You can even run the remote desktop projection and remote control cursors separately. The remote method invocation layer is generic, so that it can be used for developing other networking environments in Squeak. (Actually, in the next version of NetMorph **[4]**, we will adopt this remote messaging layer).

- Well-defined protocol

In Nebraska, an ad-hoc text-based protocol is used for communication. However, there are no convincing reasons that it must be a text protocol. In fact, the protocol is a sort of a weird one, because it has mixed binary data. We believe “StringSocket” should not be used for that purpose. In SeeThroughTalk, we have developed a new binary protocol. Now “SocketStream” does the communication job. It is quite understandable than the old one. Moreover, in the new protocol, object serialization scheme is designed to be pluggable, so you can select a new serialization scheme to that protocol. (Currently #CompactBytes, #DataStream, #ReferenceStream, and #SmartRefStream schemes are selectable).

- Efficiency

We have made many efforts so that "SeeThrough" interface never slows down the existing Squeak environment. In Nebraska, form and font caching is used for improving performance, but it does not take care of the duplicated-sends of the damaged graphical forms. In SeeThroughTalk, damages are never sent as raw forms. Only rectangular areas are sent to a peer side and damaged forms are recomputed locally. It is very effective especially in dragging rather big windows. We have customized caching schemes for other objects than form and font. For example, big strings are managed by id and only the differences are sent to a peer. It brings us a good performance gain in displaying Transcript, Browser, etc.

It is also important to eliminate unnecessary remote message calls for performance. SeeThroughTalk uses a thin-out drawing scheme in resizing morphs. It stops sending redundant display commands and drastically reduces network traffic. Additionally, the arguments of typical display commands are strictly optimized. For example, "drawLine" mostly draws a black, 1 width line, thus such parameters can be omitted if we define an optimized "draw1WidthBlackLine" command. SeeThroughTalk achieves performance gains by the accumulation of these techniques.

- Reliability

SeeThroughTalk desktop server is built on a “Service” mechanism of Comanche 6 **[5]**. It supports auto re-run, detects already used ports, and raises an error if you try to start a server on the same port. Nebraska uses plain ConnectionQueue, which does not have these reliable functions. In a client side, SeeThroughTalk periodically sends ‘heartbeat’ messages to a server. Even if one side is accidentally shut down, the other side can disconnect and release the socket gracefully (Nebraska usually raises a “Socket primitive failed” notifier in such situation).

- Bidirectional Display

SeeThroughTalk supports bidirectional display. Consequently, users can be a client and a server at the same time. This means users are able to have many sharable spaces simultaneously. In Nebraska, someone has to take the role of server and he cannot participate in other Nebraska desktops while being the server.

This function was very hard to implement, because bidirectional display tends to be in infinite displaying loops. We avoided the problem by adding "isLocal" attribute to Morph. "Local" morphs are displayed only on the local canvas. They never send display commands to peers. "RemoteDesktopMorph" is a typical local morph. It is a screen of a remote image. If it were not a local morph, the displayed remote desktop image

would be sent to other peers repeatedly. That causes disasters. SeeThroughTalk bypasses such situation by the simple "isLocal" mechanism.

4.2 Avatars and Dock

Avatars in SeeThroughTalk are placed in a dock. It is like a taskbar in a normal desktop. Generally, taskbar is used to switch active applications. It lists all the applications by a row of small icons. When we click one of the icons, the application window represented by the icon becomes forward and active. In SeeThroughTalk environment, avatars represent users. When we click one of the avatars, the related user's desktop windows become active and clear. We can change the focus of the desktops by just clicking an avatar.

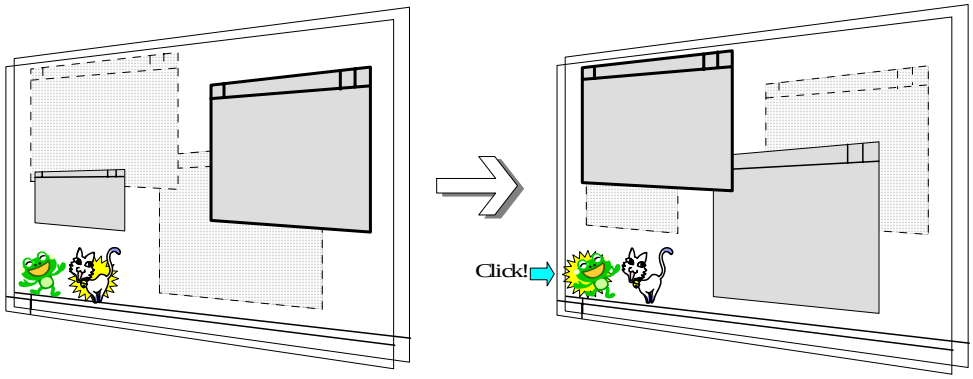

Figure 2: Avatars for changing a presently interesting desktop

Avatars also play the role of status indicators for users. Developing such AvatarMorph was not so difficult. It displays animations according to their states. Currently #normal, #excited, #sleeping, and #troubled status are supported. Animations are made from plain jpeg images. You can plug-in new animations and states very easily.

#normal and #sleeping are used for showing user's log-on status. #troubled avatar is effective for notifying network errors. #excited expression appears when source code changes are propagated from the avatar's user.

Figure 3: A frog avatar notifies a network error

4.3 "Air Chatting" Interface

In collaborative activities, it is very important to know partners' intentions and feelings. To support this, typically chat interfaces are implemented in most of the collaborative environments. However, users still feel some difficulties in expressing their feelings on demand, if they only use plain text-based chat interfaces.

In SeeThroughTalk, users can show their intentions more livelily. Each cursor has a small avatar figure and a user is able to make the small avatar talk in a balloon by just typing characters on a blank area of the

root desktop. Thus, users can express their thoughts in a very handy way. (It is so-called "typing on air" as MathMorph's term)**[6]**.

This interface has many advantages. First, users can start chatting at any timing. Second, while chatting, they can directly point the related-objects by the cursor. They can mix some attractive gestures with normal text words. It also needs only a little traffic compared with voice chats (though we do not deny using voice chats in SeeThroughTalk). We believe this "Air Chatting" interface effectively helps users to communicate actively by that simple mechanism.

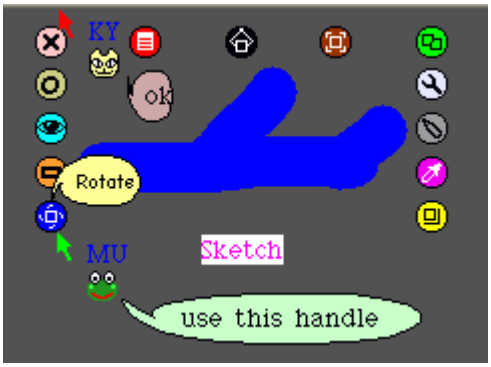

Figure 4: Chatting by avatar cursors

4.4 Desktop Color Scheme

In order to support overlapping translucent desktops, we have developed a “filter”. Filters are attached to the desktop panels when they go backward. Consequently, background desktops become darker than the foreground desktop.

However, it was not sufficient for distinguishing layered morphs at a glance. To supplement visual signs more, we have developed desktop color schemes. ColorScheme object controls how the tools (Browser, Inspector, etc) on the desktop should be colored. The schemes are named and applied to arbitrary morphs by just sending #apply: message. Currently, eight color schemes are supported and users can select their favorite scheme. The scheme is automatically applied to a desktop when a user starts SeeThroughTalk.

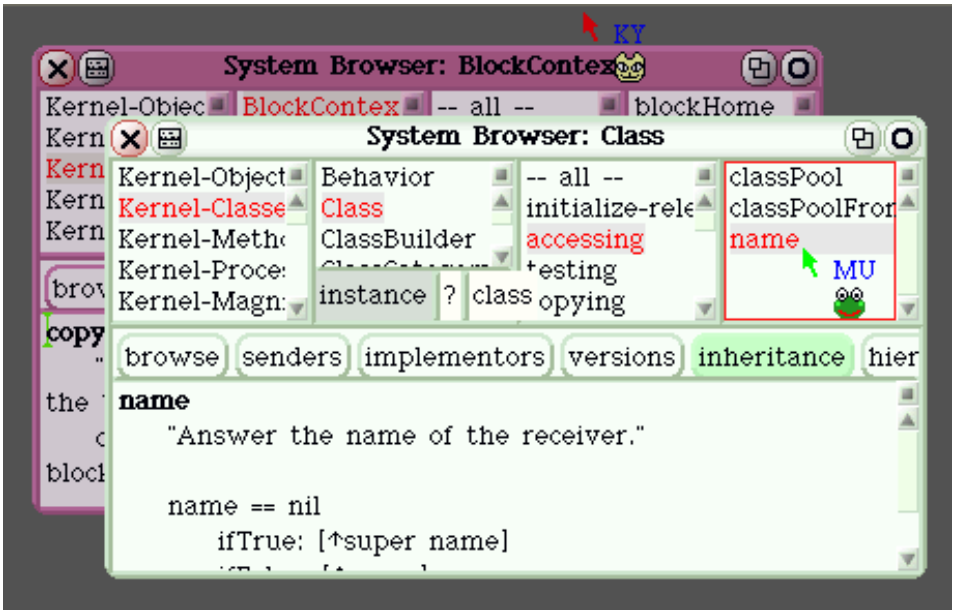

Figure 5: ColorScheme applied browsers

4.5 Morph Migration

As explained in "Project Summary", SeeThroughTalk can be thought as one big virtual desktop, which has many panels for users. The users share those panels and they freely go back and forth among those panels as avatar cursors. The same capability is given to ordinary morphs. In SeeThroughTalk, morphs on the panels are free to migrate to other panels. This function is very useful when users would like to do some activities in someone's desktop with their "taken-out" morphs.

Sending and receiving morphs between desktops should be effortless, because it gives great opportunities of participating in collaborative sessions. In order to implement this function, we adopted NetMorph. We thought it is ideal for transferring living morphs. The system is able to send any live morphs to remote images. Morphs' scripts and relations to other morphs are still kept even if they move onto other desktops. However, its "world map" settings are a little tougher for most of the end-users (It would be improved in the next release - actually I will do). Since we already had nice avatars for representing users, we simply decided to use these avatars as "gates" to the other desktops (though maps still can be set optionally). The interface is intuitive. A user can send a morph by just dragging it to a target avatar.

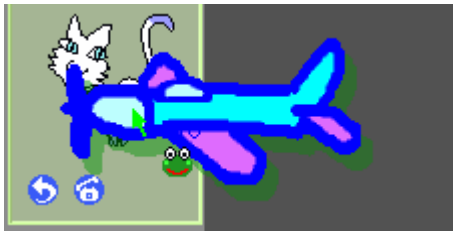

Figure 6: Sending a morph to a cat's desktop

For system window morph, we have customized the title bar to supply a convenient menu for switching the panels it is placed on. Since system windows are not draggable by default, we think the interface is more natural in this case.

Figure 7: Sending a Browser by selecting a menu item

4.6 Project Migration

Tile-scripting users generally save their works as "projects". A project usually includes live morphs, scripts, and playing instructions. In Nebraska, users are able to export their projects to remote servers by the "PUBLISH IT!" button. Other users download and play the published works. SeeThroughTalk extends this feature more. It supports exchanging projects directly between peer images. This is handier in small classroom collaboration because it does not need a central server.

Project window morph displays a project in thumbnail. Users freely place the morph on one of the shared desktop panels. The moving operation is just the same as other normal window morphs. You can simply select the "move to" location from the title bar menu.

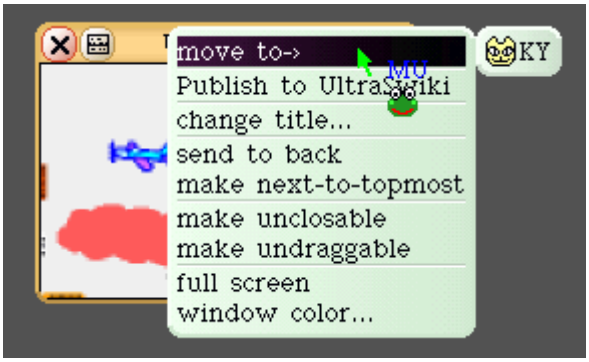

Figure 8: Sending a project to a friend

This function leverages collaboration more, because the remote user can instantly try the project she has just received, and the sender is able to interact with her by diving into her desktop.

4.7 Change Propagation

Change Propagation supports programming in collaboration. In SeeThroughTalk, any changes made to a local image are automatically updated to other members' remote images. This mechanism eliminates the many tedious file-in/file-out tasks so that programmers can concentrate on their collaborative programming.

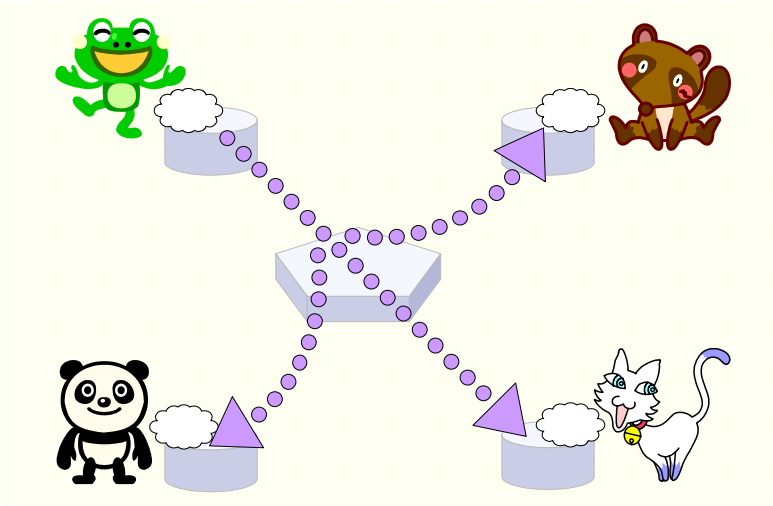

Figure 9: Change Propagation

In order to implement this function, we added hooks to ChangeSet. All the changes related to ChangeSet are raised as events. These events are broadcasted to other participants and applied to their images. We have abolished using "ChangeManager", which performed as a central server in the previous version of SeeThroughTalk. Now all the images equally send and receive changes. It is far simpler than the former one.

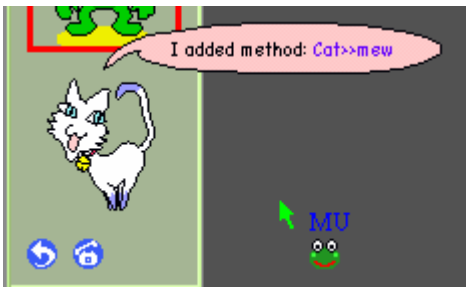

Figure 10: A method change from the remote (clicking the link will show the method)

For the limitation of time, we were not able to support category-related changes. Since ChangeSet does not include category-related changes, we have to append more hook-mechanisms. Roel Wuyts's SystemChangeNotification package would be a desirable solution **[7]**. We will implement category change propagation until the first official release of SeeThroughTalk.

Furthermore, there are no transaction supports in the current change-propagation. The problem is that we could lose parts of changes if there are some network troubles in a team programming session. For the present, users are able to synchronize source codes manually in such case. This function is called drag & drop code propagation.

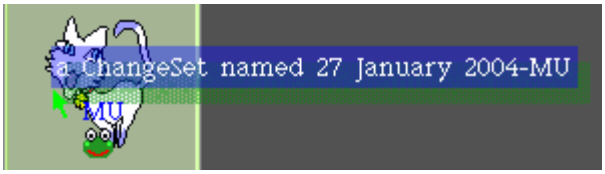

Figure 11: Applying a ChangeSet to a member by D&D

In the manual Change Propagation, users can drag and drop any parts of code (classes, methods, etc) from Browser and ChangeSorter. It is useful when team members want to apply a bulk of changes to a disconnected user.

Although this is a convenient feature, we think we should also provide an "auto change update" function. We will cover this in a future release.

4.8 Enhanced UIs for Change Propagation

Since existing development tools (Browser, ChangeSorter, etc) are not designed for multi-users, users tend to feel it difficult to know who is doing something. To solve the problem, we have extended those common development tools. For example, a message-list now shows each method name with an avatar icon, indicating who last edited the method.

ChangeSorter also displays the owner of the ChangeSet. In SeeThroughTalk, users can mark ChangeSet with an avatar icon for the reminder of the owner of the ChangeSet.

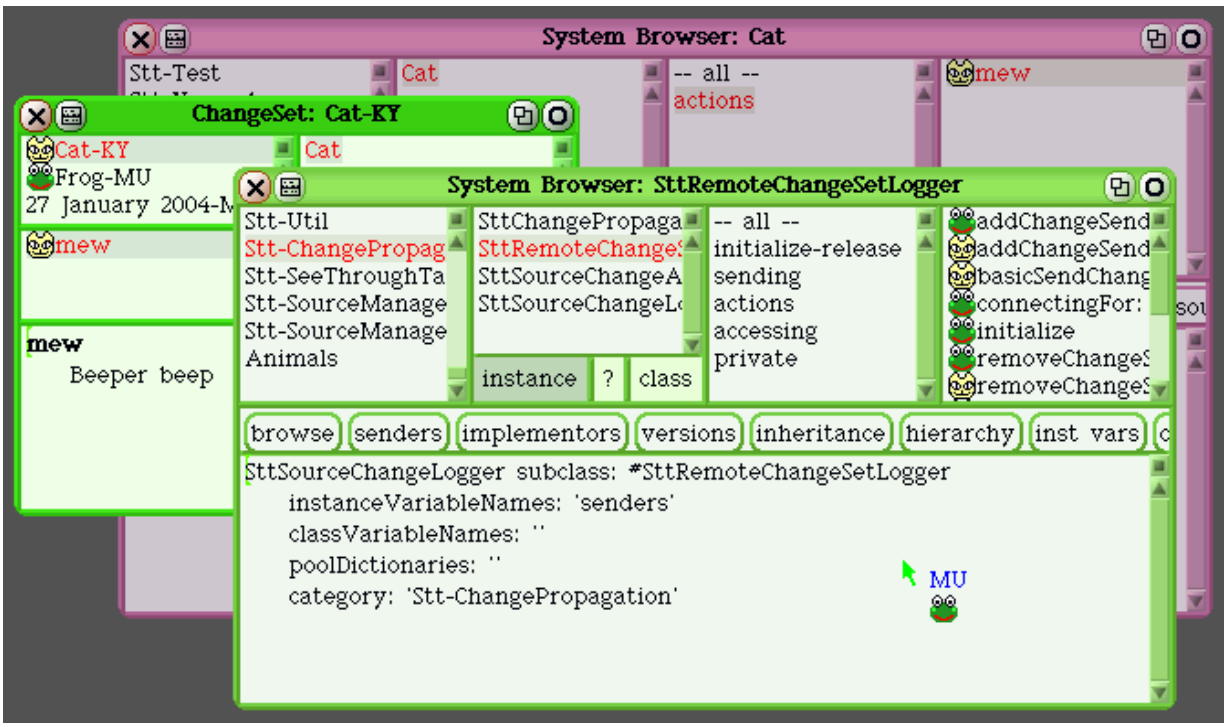

Figure 12: Enhanced Browser and ChangeSorter

In collaborating programming, it is also important to grasp the history of the changes. Since changes are applied not only from local image, but also from various remote images, the need to manage them becomes more serious. Users have to go back to an old version if they have found someone's latest change is wrong. VersionBrowser and Recent Submissions Viewer have been extended to show such information more vividly.

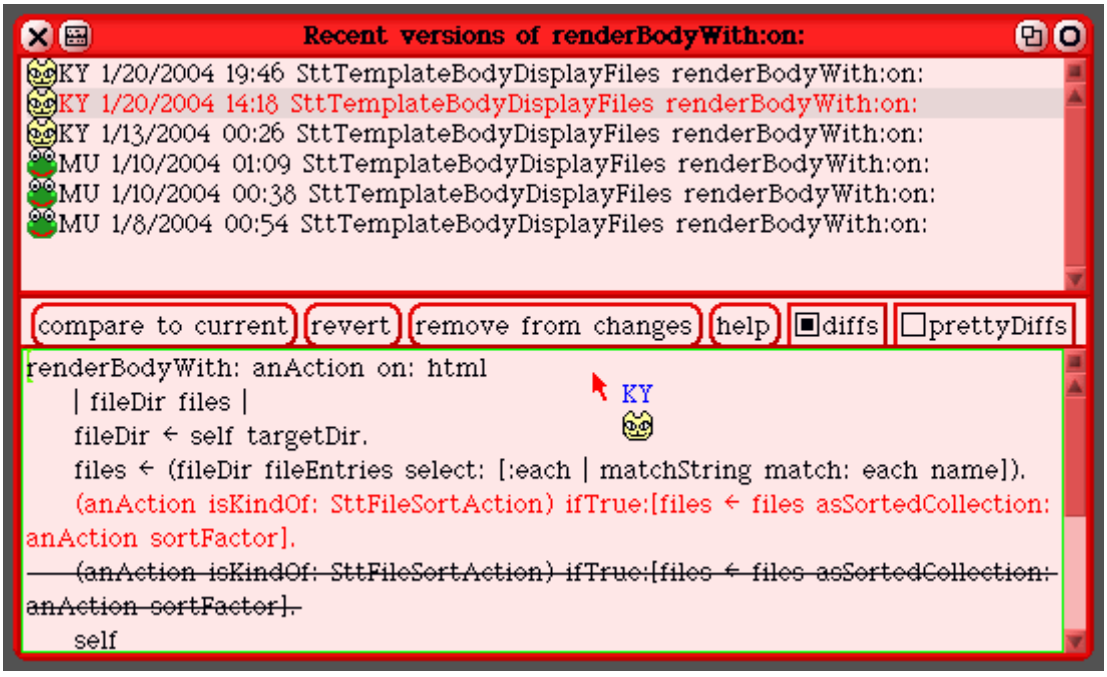

Figure 13: VersionBrowser showing avatar icons

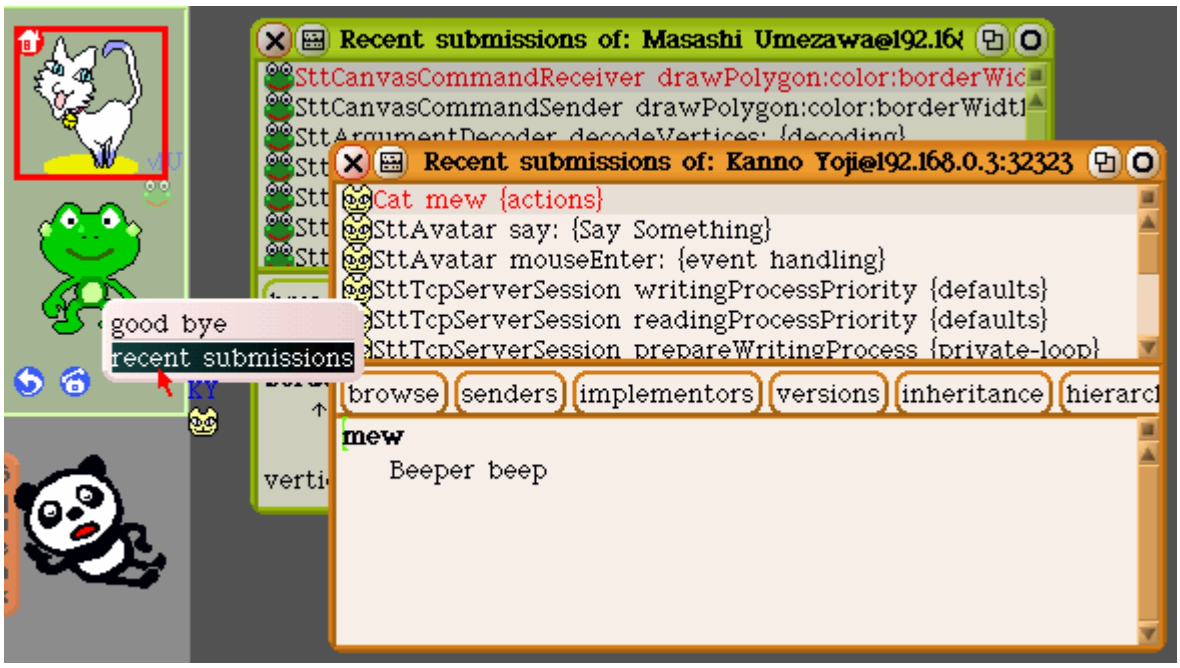

Figure 14: Recent submissions filtered by avatar

4.9 Publishing Collaborative Works (UltraSwiki)

We have implemented a new wiki, called "UltraSwiki", so that users can show and try their works easily. In Nebraska, tile-scripting users can publish their works to SuperSwiki by just clicking a button **[8]**. They can also browse and download the works through normal web browsers. However, publishable works are restricted to tile-scripting projects. Our focus is to make other forms of programming works publishable by the same one-click operation. The new developed wiki would be a superset of SuperSwiki. Thus, we call it "UltraSwiki".

SuperSwiki was originally developed in Squeak 2.7. It is becoming rather outdated and difficult to maintain. For the reason, we investigated modern wiki implementations in Smalltalk, and decided to adopt SmallWiki **[9]**. Its codes are very simple and easy to understand. The implementation is still not mature and does not provide rich set of functions compared with the popular, classical ComSwiki. However, the developers supply many tests and we felt we could add the needed functions with little effort because of the clean codes.

After all, our expectation was not wrong. In spite of the rather short period for the development (about 3 weeks), we have successfully implemented these features on the top of SmallWiki.

1. "All Teams" page

   It is the top page of UltraSwiki, which lists all the teams registered.

2. "Team" page

   It is the page for each team. It lists all projects the team is engaged in.

   It also stores team profiles and members can load these settings when starting SeeThroughTalk.

3. "Project" page

   It is the page for each project. It displays thumbnails and lists ChangeSet and Project files.

In SuperSwiki, there is a problem of "gallery page". It displays all the works flatly, and slows down the site as the uploaded projects increase. The hierarchical structure of the UltraSwiki solves this problem.

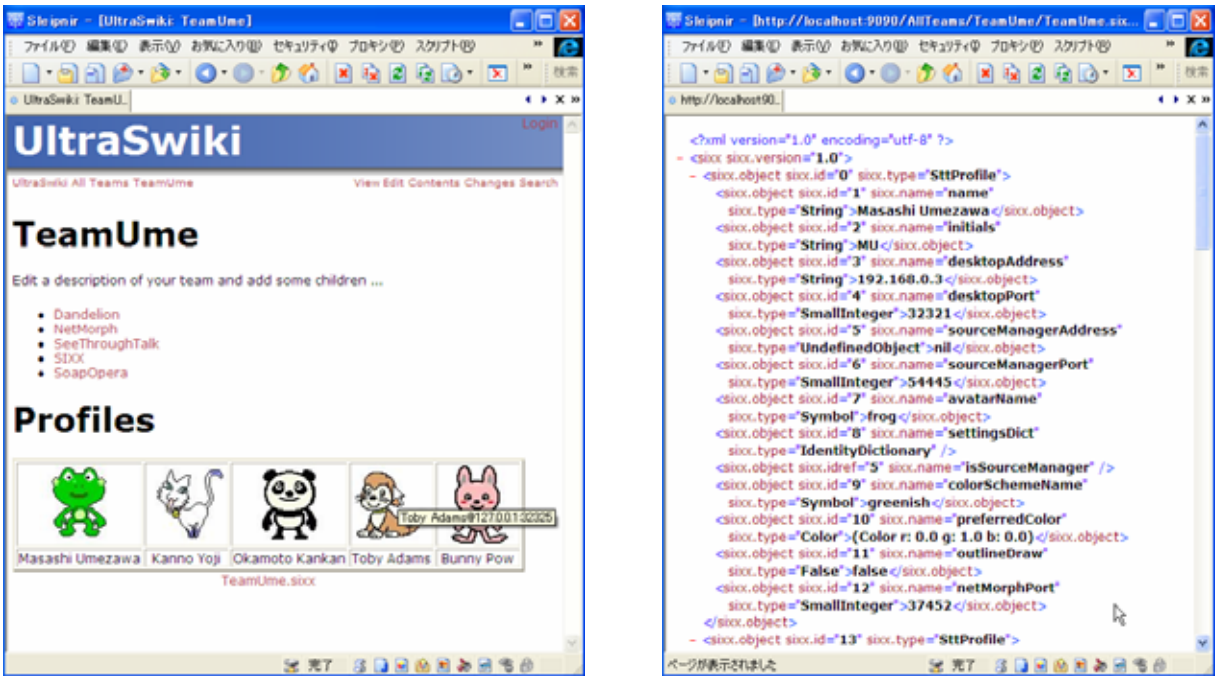

Figure 15: UltraSwiki's team page. It stores team profiles in XML

From the client side, users can upload their collaborative works easily. Currently, pull-down menus are provided for publishing ChangeSet and Scripting Project.

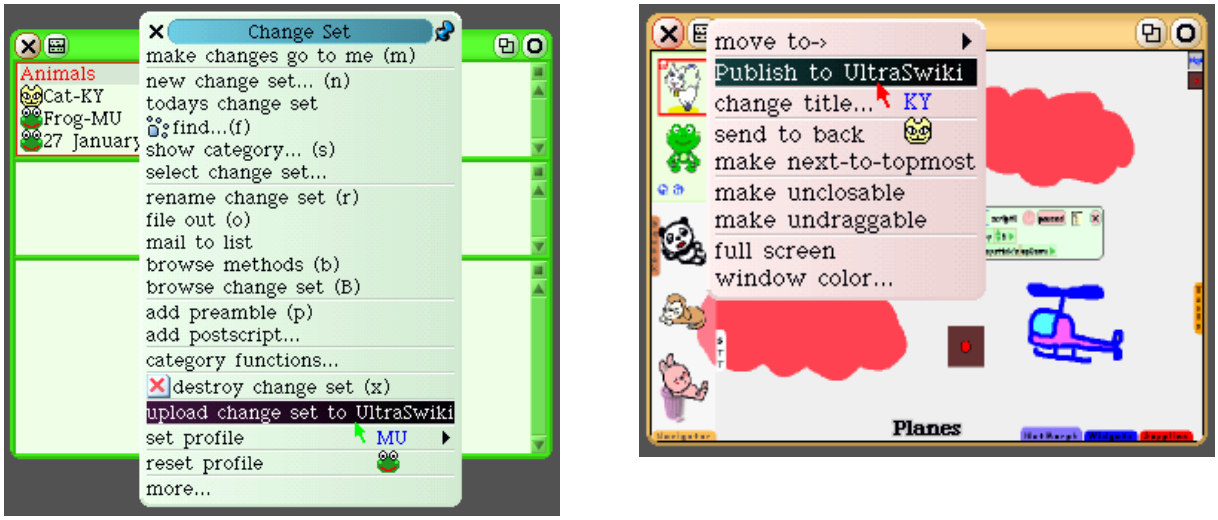

Figure 16: Uploading collaborative works to UltraSwiki

The uploaded works are listed in the Project's page. On the header of the lists, there are sort links, which enable the user to see the lists in their favorite order (date, size, etc.).

Figure 17: Uploaded works in UltraSwiki

In our perspective, UltraSwiki would be a portal site for SeeThroughTalk users. In order to be a real portal, we feel it needs more features for assisting collaboration. For example, to support a finder of someone's works is very important. We should also provide a nifty download UI for Squeak image (forcing the users to download from web browsers is not a good idea). Additionally, we would need a bulletin-board interface for visitors to put comments on team's projects.

5 Features of the Development Results

Here, we will summarize main features of the system that have already been implemented.

- Overlapping Remote Desktops

SeeThroughTalk provides virtual shared desktops. Users can share desktop experiences while seeing other desktops through translucent backgrounds. By just clicking an avatar, users can dive into the most interesting desktop. It reduces the overhead of collaboration and encourages it.

- Morph and Project Migration

Objects on the virtual desktops are all sharable. Users can move a morph to a remote desktop by just dragging it to a target avatar. For the sake of NetMorph, "live" morphs with scripts and relations migrate to other desktops.

Moreover, it is possible to exchange scripting projects directly between images. This is effective for developing scripting projects with instant feedback from others.

- Change Propagation

Source codes are "objects" in Smalltalk. Thus, it is natural for SeeThroughTalk to support sharable source codes. Any source-changes made to a local image are applied to the other images automatically. This is an essential function for programming with dynamic collaboration. To deal with network errors, users can also send the parts of codes by drag & drop from Browser. (With the above Morph Migration, users can even pitch half-written Browser itself to other images).

- Collaboration-supportive UIs

Cursors and windows are easily identifiable by ColorScheme. Cursors support "air chat", which is useful to communicate lightly. Avatars represent users and store user settings. They effectively display users' status. Traditional development tools were enhanced to list information with avatar icons so members can grasp the change history of codes at a glance.

These UIs are not equipped in Nebraska. They are small improvements, but help smooth collaboration.

- (A part of)Ultra Swiki

UltraSwiki is a new wiki for publicizing collaborative works. It stores team profiles for easy maintenance. By a simple operation, users can upload their projects (programs, tile-scripts) to the server. UltraSwiki displays the screenshots and lists their achievements in HTML. Others can download and try them by normal web browsers.

6 Future Work

- More tests

SeeThroughTalk is still in an "alpha" stage. Although it has no fatal bugs, it is not so stable. We feel we have to perform more tests for the first beta public release.

- Improving usability

We also think we should do field tests in order to make SeeThroughTalk usable. The system should be tried in real situations. We have sometimes used ChangePropagation in developing SeeThroughTalk itself. It was very convenient because it automatically updates both client and server images. However, other features have not so much been tested in actual uses. Especially in GUI, feedbacks from real users are very important. From the first beta release, we will make efforts to grow test-users in SeeThroughTalk.

- Optimization

For the limitation of time, we did not implement some display optimizations. The current version becomes slow when multi users (3 or more) access the shared desktop. We had noticed that it is because of the redundant serializations for each client. It should be fixed in the beta release.

- Re-evaluate the protocol of ChangePropagation

For the simplicity, we have implemented a protocol of ChangePropagation from scratch. However, we feel we may be able to use TeaTime for synchronizing source code objects **[10]**. In Smalltalk, source codes are represented as a tree of objects, and TeaTime provides auto-synchronization of the object tree. Consequently, we might use them as a base mechanism to implement Change Propagation. It would resolve the existing problems relating to the "auto-sync" of source codes in network failures.

- Investigate other chat GUIs

"Air Chat" has its own strong points. However, voice chat is sometimes more efficient in transferring intentions of members. We have actually used Yahoo voice chat for the net meetings of SeeThroughTalk. We may be able to incorporate a voice chat interface to SeeThroughTalk, or it might be desirable to use some independent voice chat systems. We need to do more investigations on this topic.

Considering the above "to do" lists, the first beta release of the SeeThroughTalk will be in April. The system will be a sar package and listed in SqueakMap for the easy installation. We will keep our efforts until the end of our passions for the collaborative Smalltalk.

7 Differences from the Original Specification

The term “Prevailing Images” has been changed to “Change Propagation”. The basic concept is the same, but the new term represents the feature more clearly.

8 Development Assignments

Four members are assigned in this project. The main developer is Masashi Umezawa. Other three members are employed as part-time workers.

Members’ roles are as follows:

- Manager and Chief Developer
  - Masashi Umezawa
- Support Developers (part-time)
  - Kazumi Okamoto (ColorScheme, ProfileEditor)
  - Yoji Kanno (UltraSwiki, Research and evaluation of Monticello, Seaside, and SmallWiki)
- Beta Tester and Art Work(part-time)
  - Maki Nishihara (Checking usability, Avatar, Cursor design)

9 Appendix

9.1 References and Web Sites

[1] Joseph Pelrine, Alan Knight, and Adrian Cho, "Mastering Envy Developer", Cambridge University Press, 2001

[2] Cincom Systems, Inc., "Team development in VisualWorks", http://www.cincom.com/pdf/store-twp-1020-01.pdf, 2000

[3] Lex Spoon and Bob Arning, "Nebraska", http://minnow.cc.gatech.edu/squeak/1356, 2000

[4] M. Umezawa, K. Abe, S. Nishihara, and T. Kurihara, "NetMorph - an Intuitive mobile object system", http://swikis.ddo.jp/NetMorph, 2002
[5] Stephan Pair, “Comanche”, http://people.advantive.com/~spair/comanche/, 2003
[6] Luciano Notarfrancesco and Leandro Caniglia, "MathMorphs: An Environment for Learning and Doing Math", in "Squeak - Open Personal Computing and Multimedia", Prentice Hall, 2002, pp295-340
[7] Roel Wuyts, “SystemChangeNotification”, http://map1.squeakfoundation.org/sm/packagebyname/systemchangenotification, 2003
[8] Bob Arning, "SuperSwiki, http://minnow.cc.gatech.edu/squeak/1775, 2000
[9] Lukas Renggli, "SmallWiki", http://c2.com/cgi/wiki?SmallWiki, 2003
[10] David Reed, David Smith, and Andreas Raab, "Croquet", http://www.opencroquet.org/, 2002