Next-Generation DPP with Sangam and Facetop

Kanyamas Navoraphan, Edward F. Gehringer
Department of Computer Science
North Carolina University
Raleigh, NC 27695-8206
{knavora, efg}@ncsu.edu

James Culp, David Stotts
Department of Computer Science
University of North Carolina
Chapel Hill, NC 27599-3175
{culp, stotts}@cs.unc.edu

ABSTRACT
This paper describes a state-of-the-art environment for distributed Extreme Programming that results from combining the Sangam editor, developed at NCSU and the Facetop user interface, developed at UNC-Chapel Hill. Sangam facilitates distributed Extreme Programming by sending events back and forth between a driver and a navigator working under the Eclipse development environment. Concurrently, Facetop allows the distributed pair to recapture some of the face-to-face communications that are lost in no-video distributed pairing sessions. The integrated tool is a quantum leap forward for distributed Extreme Programming as well as distributed agile development.

Categories and Subject Descriptors
D.2.6 [**Software Engineering**]: Programming Environments – *integrated environments*; D.2.3 [**Software Engineering**]: Coding Tools and Techniques – *programming editors*.

General Terms
Management, Design

Keywords
Sangam, Facetop, pair programming, distributed pair programming

1. INTRODUCTION
Since the beginning of the current century, agile development methodologies such as Extreme Programming (XP) [1], and Pair Programming (PP) [2] in particular, have exploded in popularity. At the same time, global broadband networking has made it increasingly possible to pair with colleagues in distant locations, even internationally. Preliminary data [3] indicates that distributed pair programming (DPP) seems competitive with collocated PP in terms of software quality and development time.

To support distributed pair programming, we created the Eclipse plug-in Sangam [5] to synchronize Eclipse workspaces at different sites. The Facetop application [6] addresses a different problem, allowing the members of a team to see each other while working together.

The idea to combine Sangam and Facetop originated to fill the void that exists in most collaboration tools currently in use, namely, the ability for distributed team members to communicate as if they are sitting next to each other in front of the same workstation. By this we mean the sense of presence, facial expressions, as well as a way to point at the shared work being discussed. The combination gives us the best of both worlds, with Eclipse's powerful development environment through Sangam, and the video capabilities of Facetop at the other end.

The paper first introduces Sangam and Facetop as separate tools, then proceeds to describe how the two tools were merged, what major issues were encountered during the merge, and finally how users will be studied to analyze the benefits of the tool for DPP.

2. SANGAM
Sangam is an Eclipse plug-in that has been specifically designed for DPP. It is a robust collaboration tool that synchronizes the development environments for two programmers, allowing them to work together remotely over a shared workspace as if they were using the same computer.

One of the main advantages of Sangam is that it is event-driven. It only transmits messages that are important for PP; therefore it can perform efficiently even without a broadband connection. With Sangam, each developer is free to customize his/her screen as desired, since both sides are running a separate instance of Eclipse. Unlike other desktop-sharing tools, Sangam allows its users to move the mouse independently of each other.

With Sangam, one user is designated as a driver, while the other is a navigator. The roles can be interchanged as many times as necessary during a pairing session. The plug-in intercepts keystrokes and events at the driver's end, then sends them across a message server to the other plug-in at the navigator's end. These events include Java editor events, program-launching events, and resource-change events. The message server parses the message and triggers Eclipse to perform the driver's action on the navigator's machine. This allows the navigator to see whatever the driver is doing in Eclipse on his/her machine.

Sangam uses Kizna SyncShare as its message server due to its lightweight protocol. SyncShare was developed as an Eclipse plug-in so that it can run alongside Sangam within the Eclipse IDE itself.

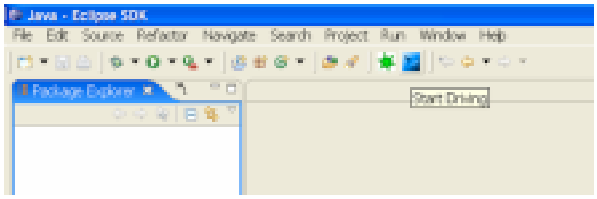

Figure 1: Sangam buttons on the Eclipse toolbar.

Figure 1 above shows the Sangam editor with two extra buttons on the Eclipse toolbar. The developers use the Connect button to connect or disconnect from the message server. Once all participants are connected to the server, one developer can use the Start/Stop Driving button to control or release control of the session. Apart from allowing developers to edit source code synchronously, Sangam also allows synchronous launch of the program being edited over the network.

3. FACETOP

Facetop [7, 9] is a collaborative system developed at the University of North Carolina at Chapel Hill for the purposes of facilitating distributed pair programming while retaining a sense of physical presence for the users.

The software displays transparent video overlays of both the local and remote users on the screen such that collaborators may point, gesture, and otherwise naturally interact with one another despite being geographically separate. They see one other over the entire desktop and retain the ability to click and type through the image into other programs.

A design goal of the system was to use inexpensive, off-the-shelf components such as normal webcams instead of more sophisticated equipment. This decision was made to encourage and enable users to use Facetop spontaneously, rather than require them to schedule time in a specially set-up room or office to collaborate. Thus, since the camera may be set up haphazardly by the user or might even be moved mid-conference, displaying the local image is important. The remote user's image exists to allow the local user to see her. The local image is displayed for self-registration purposes, so that said local user can see her own image and use it to register her hand movements relative to on-screen elements in accordance with what the remote user will see her doing. The result is that Facetop effectively provides users the ability to naturally interact with each other and their displayed materials, while not requiring any significant compromise of flexibility or investment in hardware.

Facetop has been expanded to include video filtering in several forms to address certain problems inherent in overlaying video over arbitrary other data. On textual data like code, Facetop's overlays are very easy and natural to see through. The human brain has an easy time distinguishing which parts of the image come from the video, and which come from the text. However, graphical data poses some new problems. For example, a displayed X-ray image will have a decidedly human form to it, and the brain may confuse pieces of the X-ray with pieces of the participants' faces. Facetop solves this by allowing the user to run edge-detection algorithms on its video streams to distinguish Facetop video from the data underneath, effectively reducing the level of detail displayed in the users' faces. This helps the user focus on the important data while retaining gesturing and pointing.

In order for pointing to have meaning between computers, a shared workspace must be in place. To date, Facetop has used VNC for full desktop sharing or otherwise collaborative programs like SubEthaEdit that synchronize themselves.

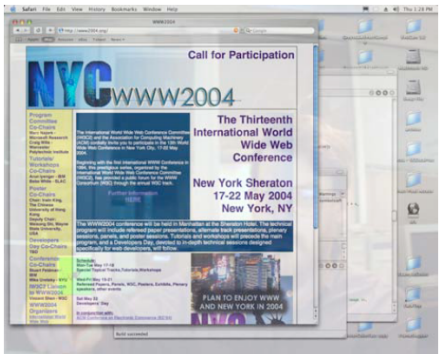

Figure 2: Facetop at fully transparent level.

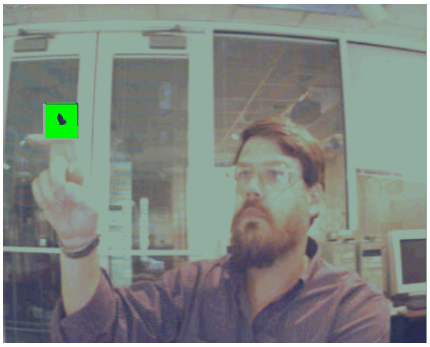

Figure 3: Facetop at fully opaque level.

4. SANGAM AND FACETOP

Combining Sangam and Facetop turned out to be a relatively non-trivial task. We realized early on that certain features of both Sangam and Facetop needed to be harmonized. The majority of the issues that we encountered were largely because Sangam had previously been tested only on the Windows platform, whereas Facetop in the two-user mode is currently supported only on the Macintosh platform. Even though an effort is currently underway to make Facetop available on the Windows platform, no release is scheduled. Therefore, we made the Macintosh our main platform for the combined tool.

4.1 Merging Sangam and Facetop

One of the original advantages Sangam was designed to deliver to its users was the ability of the parties to customize their own displays as desired. This was possible due to the fact that the plug-in only synchronizes keystrokes and events, and not any of the underlying Eclipse screen components. With the original Sangam, the driver could have been looking at a Java perspective on his screen, while the navigator would be looking at a Debug perspective on her screen, and they could still work together on the same piece of code without difficulties. However, with Facetop incorporated, it would not

have made sense for the driver to point to and start a conversation on a particular object on his screen and expect the navigator to comprehend, when in fact that particular object might be either located at a totally different position on her screen or not even present at all.

This pointing interaction satisfies a potential shortfall in the original Sangam. Sangam transmits only Eclipse events over the synchronization server. Participants could be focusing on different parts of the display. They could not get each other's attention with the mouse because simple mouse motions are not sent. The driver could select, cut, and paste a piece of text in order to communicate but this seems a very clumsy way to attract attention. With Facetop, each party sees the other any time he points.

In order for the developers to communicate through Facetop, it became obvious that the developers' screens would need to be synchronized in every aspect. First of all, the Eclipse perspective of each screen would need to match, with each and every view part located at exactly the same position. To make sure that each component is of the exact same size, both screens' resolution settings would also need to be similar. Finally, as the Facetop image would take up the entire screen, the Eclipse application window would need to take up the entire screen as well. That way, the different desktop contents at both ends would not appear in the background.

In the resulting combined tool, as soon as both users are connected to Sangam's message server and one of them starts to drive, the plug-ins at both ends begin to exchange information regarding their current perspective and resolution settings. Later on, when Facetop is successfully invoked, the display settings are automatically synchronized as needed. The Eclipse application window is maximized to take up the entire screen.

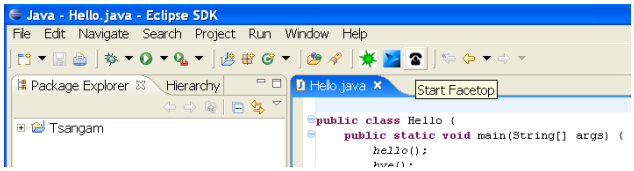

Figure 4: Start/Stop Facetop icon on the Eclipse toolbar.

For the Eclipse perspective, if the original settings are different, we simply forced the navigator to change his/hers to match that of the driver's. Resolution adjustment, on the other hand, turns out to be a bit more complicated. Since maximum resolution varies between machines, we made sure that a request would only be made to lower a setting, never to raise it. In other words, the screen with the lower resolution setting would keep it, while the higher-resolution screen would receive a request to lower the setting to match its counterpart.

4.2 Unsupported SWT_AWT on Mac OS X

To programmatically reset a screen resolution in Java, we initially opted for the least complicated way of creating an SWT_AWT bridge to run an AWT thread within the Eclipse SWT application, then using the AWT to directly manipulate the screen resolution setting as needed. This method works fine on the Windows platform. However, we discovered that the SWT_AWT feature was not supported on the Mac OS X platform.

According to the Eclipse bug report #67384 [8], the problem originated from the fact that Eclipse uses the Carbon framework for its SWT implementation, while Apple uses the Cocoa framework for its AWT implementation. Both toolkits have an event dispatcher running on the main thread. When an SWT application is started on the main thread (thread 0), it is not possible for another AWT thread to run as thread 0. Running the AWT in another thread however results in an event-thread deadlock.

The Eclipse and Apple teams eventually joined forces to have the issue resolved once and for all in May 2006, almost two years since the bug was first reported in June 2004. However, by that time, we had already decided to go instead with an unobtrusive way of explicitly displaying a pop-up window requesting the user to change his/her resolution setting manually to match the other partner's. This way, the user has absolute control over the setting and is well aware of the change that needs to take place.

4.3 Cross-Platform Character-Offset Issue

Another major issue arose while we tested Sangam communication across platforms, with a Windows machine at one end and a Mac OS X machine at the other end. We began to notice that the driver's keystrokes were being replicated at the navigator's end at positions that were a few locations off from the originals'. After a careful analysis, we found out that the issue was caused by the difference in character representations on the two operating systems, specifically the newline character. On the Mac OS X, the newline character is represented with only LF (Line Feed, 0x0A), while the Microsoft Windows uses CR+LF (Carriage Return, 0x0D, followed by Line Feed, 0x0A) to represent the same thing.

When Sangam computes the offset for the location of the first character being typed by the driver on a Windows platform, each newline gets counted as two characters. Thus, when the keystroke message reaches the navigator's plug-in running on the Mac, the specified character position is always a few characters ahead of the actual location, depending on how many newline characters are present preceding the position, as each counts here as only one character instead of two. Similarly, when the driver is typing on a Mac, the specified character position always falls a few characters behind the actual position when viewed on a Windows.

To fix the problem, every time a user starts to type, the plug-in internally recalculates character offset position, taking into account the representation difference and adjusts the offset accordingly before sending the message to its counterpart at the other end.

4.4 Licensing Issues

The last issue is licensing incompatibility. Sangam is open-source and has been distributed under the GNU General Public License, whereas Facetop is patent-pending and not openly distributed. According to the terms imposed by the GNU GPL, in order for Sangam to be distributed with Facetop, Facetop would have to adopt the GPL license. This option is, however, not possible under current circumstances. Therefore, we cannot combine the two products together in the same bundle. Negotiation is still under way as to what the final packaging will look like. In the meantime, users will need to acquire the tools separately. Specific permission is needed in order to gain

access to Facetop. Once the tools are installed, users need to tell Sangam through its Preferences page as to where the Facetop installation can be found.

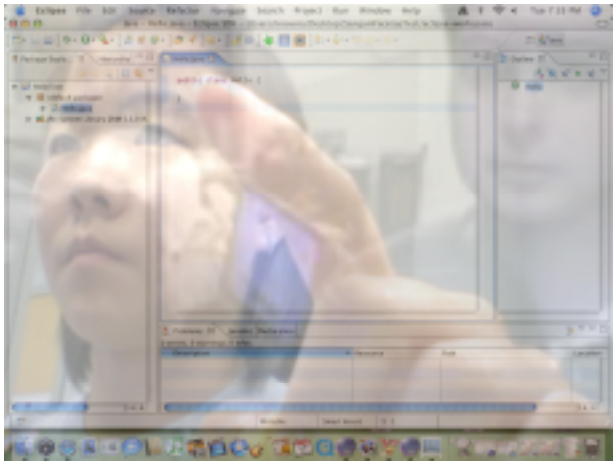

Figure 4: Sangam and Facetop communication between two users both running Mac OS X.

5. USER STUDIES

It is difficult to perform scientifically valid studies of pair programming, because, if variability is to be kept to a minimum, we need many pairs performing the same programming task, some with Sangam/Facetop and some without. The second author's CSC 517, Object-Oriented Languages and Systems class, offers one of the best environments for carrying out such studies. It is a large (60-to-100 student) graduate class consisting of experienced programmers who can participate in the experiments. Moreover, 1/3 to 1/2 of them are distance-education students, who have a vested interest in doing distributed pair programming. In a class, unlike industry, it is possible to compare teams implementing on the same specifications in distributed vs. collocated environments.

The plan is to use the abovementioned CSC 517 class in the Fall 2006 semester to carry out our user studies on Sangam/Facetop. The class has a total of 73 students currently enrolled, 53 of which are on-campus and the remaining 20 are distance-education students. They will be divided into 4 different groups according to the results from a preliminary questionnaire which collects information including:

- whether the student is attending the class as on-campus or distance-education
- the type of machine the student owns
- whether the student owns a web camera
- whether the student prefers Eclipse, or Netbeans, or have no preference over either IDEs, and
- whether the student has access to high-speed Internet

The groups will be given the same programming assignment to be coded in Java. The first group of students will be carrying out the task as collocated pairs. The second group will be distributed pairs using VNC to accomplish the same task. The third group will be using Sangam for their distributed PP. The last group will be using Sangam and Facetop together for the same purpose. We expect the last three groups to be using an instant messaging tool of their choice for voice communication during the pairing sessions. Quantitative data will be collected using Hackystat [4], a software development metrics collection tool developed by the Collaborative Software Development Laboratory (CSDL) at the University of Hawaii. After finishing the assignment, qualitative surveys will be administered to collect feedbacks on the efficacy and ease of use of the programming environments each group was assigned with. The data collected from each group will then be analyzed and compared against one another.

If all go as planned, we hope to show that the distributed pairs using Sangam and Facetop provides a result closest to that of the collocated pairs, in terms of productivity, quality, and user satisfaction when compared to those distributed pairs who use either Sangam alone or VNC.

6. CONCLUSIONS

The Sangam and Facetop combination promises to be the effective collaborative software development tool that distributed partners were looking for, as it provides not only a powerful integrated development environment, but also a way for them to communicate visually while they work. Our user studies will be completed in December 2006. We expect the results to indicate that we are one step closer to making DPP experience comparable to that of collocated PP.

7. ACKNOWLEDGMENTS

Funding for the current project comes from the corporate sponsors of the Center for Advanced Computing and Communications (CACC) at North Carolina State University. We gratefully acknowledge IBM for their 2004 Eclipse Innovation Grant for Sangam development. James Branigan of the IBM/OTI Raleigh lab was the first to suggest integrating Sangam and Facetop.

8. REFERENCES

[1] Beck, K., *Extreme Programming Explained: Embrace Change*. Reading, Massachusetts: Addison-Wesley, 2000.

[2] Baheti, P., Williams, L., Gehringer, E., and Stotts, D., "Exploring pair programming in distributed object-oriented team projects," *OOPSLA 2002: Object-Oriented Programming Systems, Languages, and Applications (Educators' Symposium)*, Seattle, WA, Nov, 2-6, 2002.

[3] Baheti, P., Gehringer, E., and Stotts, D., "Exploring the efficacy of distributed pair programming," *Proc. XP Agile Universe 2002*, Chicago, August 4-7, 2002, Springer-Verlag *Lecture Notes in Computer Science* 2418.

[4] Hackystat (Research Summary), Retrieved August 30, 2006 from http://csdl.ics.hawaii.edu/Research/Hackystat/ .

[5] Ho, C., Raha, S., Gehringer, E., Williams, L., "Sangam – A Distributed Pair Programming Plug-in for Eclipse," *Proc. Eclipse Technology Exchange*, OOPSLA 2004, Vancouver, October 24, 2004.

[6] Smith, J., *Facetop – Transparent Video Interface: A quick discussion of Facetop to dispell some misconceptions that have been showing up*. Retrieved August 30, 2006 from http://www.cs.unc.edu/~smithja/facetop.

[7] Stotts, D., Smith, J., and Gyllstrom, K., "Support for Distributed Pair Programming in the Transparent Video Facetop," XP/Agile Universe 2004, Calgary, Aug 15-18, pp. 92-104.

[8] SWT_AWT not implemented for Mac, *BugZilla Bug 67384*. Retrieved August 30, 2006 from https://bugs.eclipse.org/bugs/show_bug.cgi?id=67384.

[9] Stotts, D., Williams, L., et al., "Virtual Teaming: Experiments and Experiences with Distributed Pair Programming," TR03-003, Dept. of Computer Science, Univ. of North Carolina at Chapel Hill, March 1, 2003.