Sangam – A Distributed Pair Programming Plug-in for Eclipse

Chih-wei Ho[1], Somik Raha[2], Edward Gehringer[1], Laurie Williams[1]

[1]Department of Computer Science, North Carolina State University
Raleigh, NC 27606, USA, {cho, efg, lawilli3}@ncsu.edu

[2]Industrial Logic
Berkeley, CA 94708, USA, somik@industriallogic.com

Abstract

Pair programming requires two programmers working together at one computer. However, the trend toward globally distributed organizations makes long-distance collaboration necessary. Sangam is an Eclipse plug-in for Eclipse users to share workspace so that developers may work as if they were using the same computer. In this paper, we will show how this is achieved in Eclipse platform, and discuss our experience of distributed and collocated pair programming during the development of Sangam.

1 Introduction

Thanks to the popularity of XP [2] and benefits of pair programming (PP) [3], more programmers are willing to pair with their colleagues during software development. By definition, PP requires programmers to work on the same computer. However, because of industry globalization and the trend towards distributed workforces, more and more programmers on the same team work at different sites, even on different continents. To enjoy the benefit of PP, distributed team members need to work as if they were sitting next to each other. We propose Sangam [1] (Sangam means [2] "confluence or a flowing or meeting together" in Sanskrit), a plug-in for Eclipse, as a tool for distributed PP. Sangam provides a user interface specifically for distributed PP (DPP) and synchronizes the development environments for both programmers. The plug-in itself is developed by a distributed team. In this paper, we will discuss the available tools supporting distributed collaboration and our experience with DPP. Additionally, the implementation of Sangam will be presented.

The rest of this paper is organized as follows. Section 2 provides the background of PP and distributed collaboration. Section 3 describes our experience of collocated and distributed PP. Section 4 presents the design of Sangam. Finally Section 5 gives the current status of our work and suggests future direction.

[1] http://sangam.sourceforge.net
[2] http://sangam.sourceforge.net/WhatDoesSangamMean.html

2 Background

PP has been practiced sporadically for decades [6], but the popularity of the practice has grown by its use within the Extreme Programming [2] software development methodology. While geographical distance is usually considered harmful to PP, there are several situations where distributed collaboration is necessary. This section provides background information on PP and distributed collaboration.

2.1 Pair Programming

PP is a practice, whereby two programmers work side by side at the same computer, continuously collaborating on the same design, algorithm, code, or test [6]. One of the programmers is the *driver*, who has the control of the keyboard and the mouse, actively implements the program, and explains the implementation to his or her partner. The other is the *navigator*, who watches the driver; reviews driver's design; detecting driver's errors; and is a brainstorming partner. After a period of time (usually less than one hour), the programmers switch their roles. Early evidence for the effectiveness of PP was only anecdotal. Recent researches find that pair programmers produce code with higher quality without sacrifice of productivity and have more job satisfaction [3] and have more confidence in their products [7].

The necessity of working on the same computer is a limitation of PP. Although research shows that collaboration at a distance is difficult to achieve [5], with industry globalization, distributed collaboration is becoming more common. Additionally, in academia, some students have schedule conflicts and cannot get physically together. To enjoy the benefit of PP when working at one computer is not possible, we need a virtual collaboration environment in which distributed developers can work as if they were sitting together, using the same input devices and looking at the same monitor.

2.2 Distributed Collaboration

Based on time arrangement, there are two forms of collaboration: asynchronous and synchronous. With asynchronous collaboration, the team members work at different times and integrate their work after they finish their tasks. There is typically a centralized server where the communication takes place for collaborative programming efforts. SourceForge[3] is one example of such a server. PP belongs to the other category, synchronous collaboration. There are two basic approaches for distributed synchronous collaboration [4]. The first one is to broadcast the display of the output of any application from a member to all the others. This approach requires a large amount of image information transmitted across the network, but all applications can be used in the environment without any modification. Some applications, including VNC[4] and Microsoft NetMeeting[5], use this method to share one person's desktop with collaborators. The second approach is to design the application specifically for distributed collaboration. It requires specifically-designed applications, but the user interface can be more sophisticated and effective for the purpose of collaboration. One example application is Yahoo Messenger[6]. In Yahoo Messenger, when a user sends a text message, the text, rather than the screen buffer of the chat window, is sent through the network. Also, its user interface is designed for composing text messages.

In their research, Baheti et al use NetMeeting as a tool for DPP [1]. They find out that, in terms of productivity and quality, DPP is as effective as collocated PP. Our experience shows that, while shared-desktop tools can be used for distributed PP, the display refresh rate can be too low for the programmers to understand what their partners are doing, especially when high-speed network connection is not available. Therefore, we propose a plug-in that is specifically designed for DPP in the Eclipse development environment. Rather than sending screen buffer information through the network, Sangam transmits messages that are important for PP. The information is significantly less than image data, so developers can use this plug-in without broadband network

3 Experience

During the development of Sangam, we gained some experience of both collocated and distributed PP. The team consists of 4 students in North Carolina State University (NCSU) and 4 independent programmers in California. When pairing with a distributed partner, we used VNC for desktop-sharing, Eclipse for the development environment, and Yahoo! Messenger for voice communication. The experience of this project is described in this section.

3.1 Collocated vs. Distributed

We had similar experience with this project. We held weekly long-distance conferences for iteration planning and velocity [2] tracking, and found out that we delivered approximately the same amount of functionality in the same amount of time with collocated pairing as with distributed pairing. Because we were so engaged in programming tasks, we felt satisfied (and exhausted)

[3] http://www.sourceforge.net/
[4] http://www.realvnc.com/
[5] http://www.microsoft.com/windows/netmeeting/
[6] http://messenger.yahoo.com/

after each pairing session, whether collocated or distributed. Furthermore, we found PP to be an efficient method for knowledge exchange. Although we did have a Sangam developer's guide[7], most of the required skills and techniques for this plug-in were exchanged during PP. We believe that this demonstrated the applicability of distributed PP.

One difference between collocated and distributed PP is the sense of presence. With distributed pairing, the navigator could lose concentration more easily because the voice of the driver came out from the speakers, not from someone sitting physically next to the navigator. However, this was not a problem for us. When the navigator lost the attention, the driver noticed that the navigator talked less and would ask the navigator to contribute something. When working collocated, it was easier to stay focused.

The other difference is in the social aspect of programming in pairs. When the project started, the developers did not know each other. The collocated developers became familiar friends after first few pairing sessions. However, for the developers in different locations, there was still a sense of unfamiliarity among them, although they could work very well when programming together.

3.2 Tool Support

The development team is distributed from coast to coast. Most of the time, we had the luxury of broadband Internet connection. The shared-desktop tool we used worked well. Nevertheless, we did notice the following shortcomings with the tools we used that made it suboptimal for PP:

1. Unlike collocated pair programmers, distributed pair programmers have control of his or her mice and keyboards and can move the mouse or type a keystroke at any time. It is irritating to the driver when the navigator hits the keyboard or moves the mouse accidentally.
2. Low display refresh rate can sometimes be confusing. Something significant may be lost in the remote display. For example, if the driver copies some text from an editor and paste it in another editor, the navigator may only see the new content of the later editor, without knowing where it came from.
3. It is better if both developers use the same resolution for their monitors. Otherwise, one may lose the trace of the mouse pointer.

These problems can be addressed with tools that are specifically crafted for DPP, such as Sangam.

4 Plug-in Design

We use an event-driven design for this plug-in. When the driver does something in Eclipse (e. g. cut and paste some code), the plug-in intercepts the event and notifies the plug-in at the navigator's end to automatically perform the same task. There are three basic components in our design: event interceptor, message server, and event reproducer. Following are the detailed descriptions of these components.

The responsibility of the event interceptor is to capture the event when the driver does something in Eclipse and then send it to the message server. Eclipse is an open environment and a user may install virtually unlimited number of plug-ins. It is impossible to capture all the events generated by every plug-in. Our goal is to write a plug-in to support synchronous development in Java programming language. Therefore, we only focus on Java editor events, program launching events, and resource change events.

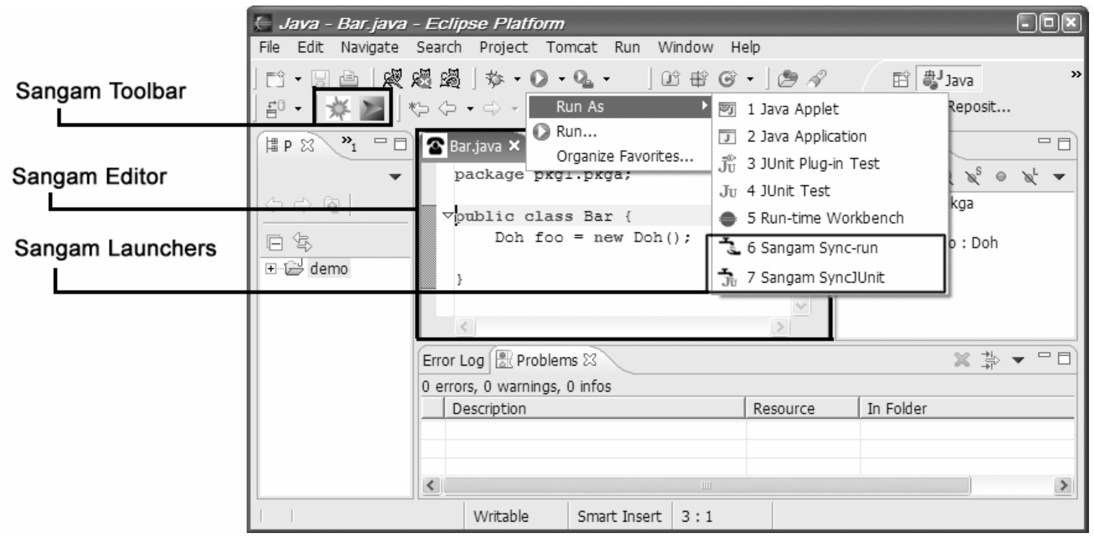
We use a centralized server for message handling. All developers who want to participate in a programming session need to connect to the same server. It is possible for more than two developers to join a programming session, but only one can be the driver at a time. We use Kizna SyncShare[8] as the message server because of its lightweight protocol and easy SDK for development. While the server can run on an individual machine, we also developed an Eclipse plug-in for SyncShare so that the server can also run within the development environment.

When the driver does something in Eclipse, the action needs to be reproduced at the navigator's computer. This is done by the event reproducer. When the event reproducer receives a message from the message server, it parses the message and interacts with Eclipse to perform the driver's action. Therefore, the navigator can see whatever the driver is doing in Eclipse.

Because Sangam is designed for DPP, we can specify the interaction between the driver and the

[7] http://sangam.sourceforge.net/SangamDocumentation.html

[8] http://www.kizna.com/products_sync.html



```
package pkg1.pkga;

public class Bar {
    Doh foo = new Doh();

}
```

Figure 1: Sangam Plug-in

navigator. Figure 1 shows an Eclipse workspace after installing Sangam. The Sangam Editor provides functionalities for the developers to edit source code synchronously. The Sangam Launchers enable the developers to launch a Java application or JUnit test together. The developers use the Sangam Toolbar to connect to or disconnect from the message server (the left button on the toolbar) and Start / Stop Driving (the right button on the toolbar). In our implementation, a developer becomes the driver when he or she presses the Start Driving button. The driver then has the control of mouse and keyboard until he or she hits the Stop Driving button. Although the navigator may also move the mouse, it would not affect the cursor on the driver's screen. Currently this plug-in does not prevent the navigator from typing, so paired programmers need to work out a protocol that one should not use the keyboard if he or she is not the driver.

We think it is important for the navigator to visualize all actions of the driver. In our design, the event reproducer uses the Eclipse API to perform the driver's actions. The navigator is guaranteed to see all of the driver's actions, and the actual layout of the Eclipse window does not matter at all.

Currently, the lack of extensibility is the greatest limitation of this plug-in. Because the tool can only catch and reproduce predefined events, it cannot support PP in other programming languages or editors. For example, this plug-in cannot be used with CDT (C/C++ Development Tools) because it does not know how to intercept CDT events.

5 Results and Future Work

The development of Sangam is an ongoing effort. We keep adding new features to make Eclipse a platform for PP. In the current version, Sangam provides the following features:

- Editor synchronization: Including typing, selection, opening, closing, and view port scrolling synchronization.
- Launching synchronization: The programmers can launch or debug the same Java application or JUnit test at the same time.
- Resource synchronization: When the driver adds, deletes, or modifies the files in his or her local disk using Eclipse, the same change will also be reflected in the navigator's local disk.
- Refactoring synchronization: From the aspect of Eclipse API, refactoring is a complicated form of resource and editor change. A single refactoring may affect many files and the content of different editor windows. Sangam is designed to deal with some special cases raised with refactoring within Eclipse.

We try to synchronize the workspace of the programmers participating a pair session. The synchronization, however, conflicts with CVS support in Eclipse. After a pairing session, because every programmer has the newest version of the files, these files will be labeled as outgoing change. Nonetheless, only one programmer can check in the code. The other needs to update it from CVS, although the source code is already up to date. Additionally, supporting software development process in Sangam is our long-term goal. To provide complete support for synchronized distributed collaboration, we plan to continue this project in three different directions:

1. More workspace synchronization: In the near future, this plug-in will support all refactoring synchronization. The issues with CVS should also be resolved.
2. Data collection: The event-driven design makes this plug-in an ideal tool for collecting accurate data in PP, such as the amount of time spent driving or navigating and the amount of time required to pass a JUnit test. The collected data will be used for further research of PP and distributed collaboration.
3. Development activity support: Coding is but a part of software development. Other development activities, like the planning game [2], can also benefit from distributed collaboration. Our ultimate goal is to make Eclipse a collaborative environment for the whole software life cycle.

Acknowledgements

The author would like to thank Sree Vidhya Lakshmi, Francine Gatewood, Benny Sadeh, Jan Chong, and Brad Failor, for their participation in the development of this plug-in. The development of the Sangam was funded in part by an IBM Eclipse Innovation Award.

About the Authors

Chih-wei Ho is a Ph. D. student of Computer Science in NCSU. His interest is in agile software process and software testing. Somik Raha is the founder of the Sangam project. He is a professional XP coach. Ed Gehringer and Laurie Williams are a professor and assistant professors at NCSU, respectively.

References

[1] P. Baheti, E. Gehringer, and D. Scotts. Exploring the efficacy of Distributed Pair Programming. In *Proceedings of Extreme Programming and Agile Methods – XP/Agile Universe 2002*, pages 208-220, Chicago, Illinois, USA, 2002.

[2] Kent Beck. *Extreme Programming Explained: Embrace Change*, Addison-Wesley, 2000.

[3] A. Cockburn and L. Williams. The Costs and Benefits of Pair Programming. In Extreme Programming Examined, pages 223-247, Addison-Wesley, 2001.

[4] C. A. Ellis, S. J. Gibbs, and G. Rein. Groupware: Some Issues and Experiences. In *Communications of the ACM*, Vol. 34, Issue 1, Pages 39-58, January 1991.

[5] G. M. Olson and J. S. Olson. Distance Matters. In *Human-Computer Interaction*, Vol. 15, pages 139-179, 2000.

[6] L. Williams and R. R. Kessler. *Pair Programming Illuminated*, Addison-Wesley, 2002.

[7] L. Williams, R. R. Kessler, W. Cunningham, and R. Jeffries. Strengthening the Case of Pair Programming. In *IEEE Software*, Vol. 17 Issue 4, pages 19-25, July/August 2000.

Appendix A: Copyright Notice