Saros: An Eclipse Plug-in for Distributed Party Programming

Stephan Salinger
Institut für Informatik
Freie Universität Berlin
14195 Berlin, Germany
stephan.salinger@fu-berlin.de

Christopher Oezbek
Institut für Informatik
Freie Universität Berlin
14195 Berlin, Germany
christopher.oezbek@fu-berlin.de

Karl Beecher
Institut für Informatik
Freie Universität Berlin
14195 Berlin, Germany
karl.beecher@fu-berlin.de

Julia Schenk
Institut für Informatik
Freie Universität Berlin
14195 Berlin, Germany
julia.schenk@fu-berlin.de

ABSTRACT

This paper describes the social practice of distributed party programming as a natural extension of pair programming in a distributed context with two or more software developers working together. To this end we provide an overview of the Eclipse plug-in Saros, a software implementation supporting this practice, and explain its technical architecture. The central contribution of this paper is a detailed description of four concrete scenarios of distributed collaboration where one of them is distributed party programming. Furthermore it will be shown how each scenario is supported by Saros. The paper closes with a discussion of preliminary findings about establishing Saros in Open Source projects.

Categories and Subject Descriptors

D.2.6 [**Software Engineering**]: Programming Environments - Integrated environments; D.2.9 [**Software Engineering**]: Management - Programming teams; K.4.3 [**Computing Milieux**]: Organizational Impacts - Computer-supported collaborative work

General Terms

Human Factors

Keywords

Eclipse, Collaboration, Awareness, Distributed Development, Pair Programming, Distributed Pair Programming, Distributed Party Programming

1. INTRODUCTION

Software developers who want to collaborate face several problems when not co-located, for example awareness issues. Saros is a plug-in for the integrated development environment (IDE) Eclipse[1] which offers a solution to some of these obstacles. It gives developers the ability to work simultaneously on their projects over a network by allowing concurrent editing access to shared work artefacts. When editing documents, developers can immediately see the modifications made by all other participants and Saros visualizes who is responsible for each change.

The Saros software is the main result of ongoing research into collaborative programming activities between distributed parties. Initially we examined the issues surrounding distributed *pair* programming and began to develop a software solution for this [12]. But our work has evolved to cover the broader scope of distributed collaborative software development and distributed pair programming is just possible type of collaboration considered.

In developing a broader solution we have considered how each party within a distributed collaboration needs to be able to interact with every other party. One party becomes aware of another through a variety of methods, but these methods can be used differently depending on the purpose of the work, so it is envisaged that Saros is useful in a number of different collaborative scenarios.

This paper will discuss the Saros software, outline a number of scenarios in which it is judged to be useful, and provide information on its present state by discussing successes, problems, and work currently underway. In section 3, the Saros software is introduced from a technical perspective by providing an overview of the architecture and explaining how each module provides functionality. How it generates awareness between collaborating parties is stated in section 4. After giving an impression of how the software functions, the paper then, in section 5, discusses the primary development scenarios in which Saros is intended to be most useful by outlining difficulties in some aspects of distributed development and demonstrating how Saros ad-

[1]`http://www.eclipse.org`

dresses them. Section 6 outlines the successes the project has experienced so far and also the problems the project members have come up against. The paper then discusses ongoing work to provide new features and uses for the Saros software in section 7. After this, some tools related to the topic are briefly discussed in section 8, and some concluding remarks follow in the final section.

2. RELATED WORK

Several ideas behind Saros are based on the practice of pair programming (PP), popularised by the agile development methodology eXtreme Programming (XP) [3]. In PP, two developers are working together on the same artifact (e.g. code or design), sitting side-by-side at one computer. The developers work in two roles: One developer, the so-called driver, "has control of the pencil/mouse/keyboard and is writing the design or code. The other person continuously and actively observes the work of the driver – watching for defects, thinking of alternatives, looking up resources, and considering strategic implications of the work at hand. The roles of driver and observer are deliberately switched between the pair periodically" [45]. Although this role model is commonly accepted, some studies have questioned the underlying assumption that the driver is working on a medium level of abstraction, while the observer is both working on a higher and lower level. They rather find that the pair normally moves through different abstraction levels together [6, 8, 18].

Advocates of PP assume the following advantages: improved knowledge transfer, more significant final outcomes and enhanced team-building processes. Accordingly, a lot of studies analysed PP regarding the resulting design or code quality (in particular defect rate), speed, cost, knowledge transfer and programmers satisfaction. The results of these studies are often contradictory [37], but there is evidence that in some settings PP improves code quality (e.g. [21, 31]) and leads to faster progress (e.g. [21, 31]), better diffusion of knowledge and respectively better learning (e.g. [5]), higher work satisfaction (e.g. [13, 31]) and greater confidence regarding their own work (e.g. [31]). But it also seems that PP can lead to higher programming costs and effort (e.g. [21, 31]). At the moment it is not well understood in which situations an investment in PP will pay off later in the development process or which factors determine the success of PP.

Variants of PP are distributive pair programming (DPP) and side-by-side programming (SbS). In DPP the partners are not co-located in front of a single computer as in PP. They work together simultaneously on the same artifact (usually code), but from different locations. "This means that both must view a copy of the same screen [respectively files], and at least one of them should have the capability to change the contents" [1]. For this purpose the developers need technological support. Cox and Greenberg [10] identified several important design principles for supporting the collaboration of distributed teams via software: "(1) Provide a common, visually similar environment for all participants; (2) Provide timely feedback of all actions within the workspace; (3) Support gesturing and deictic references (pointing gestures accompanied by verbal cues such as 'here' or 'this one'); (4) Support workspace awareness ('the up-to-the-moment understanding of another person's interaction with the shared space' [19]" [20].

There are only a few studies concerning the effects of using DPP (see Section 8 for a discussion on available tools). Baheti et al. conducted an experiment with students who were virtually paired using Microsoft NetMeeting. The results indicate that DPP is comparable with "co-located or virtual teams (without pair programming) in productivity and quality" [1]. Similar conclusions are drawn by two case studies (DPP vs. traditional virtual teams) with students done by Stotts et al. [42], using different COTS solutions (e.g. Microsoft NetMeeting and pcAnywhere (Symantec)). In an experiment conducted by Canfora et al. [7] students used VNC "to share the desktop and Instant Messaging of NetMeeting to support communication" (no audio channel). Canfora et al. found that distributed pairs tended to stop cooperating and began to work as two solo programmers. A possible reason for this behaviour could be "that the lack of an audio channel interfered with the pairs' ability to effectively collaborate" [20].

Side-by-side programming was first described by Cockburn [9] and can be explained in the following way: "Side-by-side programming is like solo programming in that the two programmers involved each use their own computer and normally work alone on a (sub)task. However, side-by-side programming resembles pair programming in that the programmers can switch to a pair mode at any moment: their two computers are located very closely side-by-side to one another. The idea is that they will be cooperating directly for a while whenever this appears particularly useful" [32]. Nawrocki et al. [29] compared the productivity of SbS to solo programming and PP. They found a 20% overhead compared to solo programming, while pair programming had an overhead of 50%. Dewan et al. [11] have done an exploratory, qualitative study on distributed side-by-side programming (DSbS), providing a classification of work modes: concurrent uncoupled programming, concurrent coupled programming, pair programming, concurrent programming/browsing and concurrent browsing. Similar modes of collaboration, called cooperation episode types, have been found by Prechelt et al. [32]: exchange project details, exchange general knowledge, discuss strategy, discuss step, debug work product, integrate work products, and make remarks.

3. TECHNICAL ASPECTS

Saros is realised as a plug-in for the IDE Eclipse written in Java and consists of 30,000 non-comment lines of code. Saros uses a layer architecture with event passing and consists of five major modules: (1) The *network layer* is responsible for connecting users of Saros to each other and is based on the Extensible Messaging and Presence Protocol (XMPP) [35]. (2) The *concurrency control layer* based on the Jupiter algorithm [30] is in charge of synchronizing the activities of users in a Saros session. (3) Saros connects to existing Eclipse components using several classes in the *bridge layer* such as for instance the *text editor bridge* which is responsible for capturing local text editing events and replaying remote ones. (4) To detect and repair synchronization issues caused outside of Eclipse or by network failures, there is a *consistency watchdog* component. (5) Lastly, the *feedback module* gathers usage and process data from each Saros session and direct users to a survey, if they have enabled this during installation.

Individual modules in Saros are composed using the dependency injection design pattern [16] via the PicoContainer

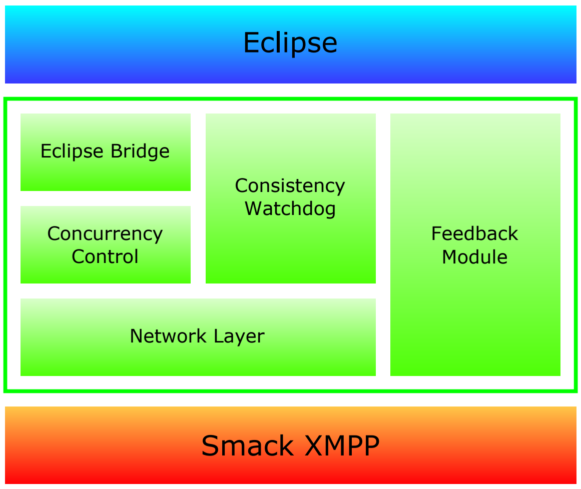


Figure 1: High-level architecture of Saros

framework[2], which helps to keep the architecture transparent and modifiable. The resulting software architecture is shown in Figure 1 and each component is discussed in more detail below.

The network layer of Saros is responsible for establishing bi-directional communication between the participants and enable them to find each other. The layer was built ontop of the Java Open Source XMPP-client library *Smack*[3], which enables developers to use existing XMPP accounts and contact lists for instance from Jabber.org or Google Mail. XMPP was chosen as an underlying middleware because of being an open standard. XMPP is also a client-server communication protocol which is well suited for users to find and connect to each other. However, Saros also needs preferably low-latency communication during a session and high-bandwidth communication while synchronizing the project between participants. To this end, Saros uses the XMPP Extension Protocol Jingle [36] to establish a peer-to-peer connection between session participants and transfer large binary objects outside the XMPP band. Jingle includes provisions for Network Address Translation (NAT) traversal using Session Traversal Utilities for NAT (STUN) [34] to achieve P2P connections when both users are behind firewalls or routers. If NAT traversal fails, SOCKS5 proxy servers can be used to relay communication.

The concurrency layer of Saros uses the Jupiter algorithm for maintaining eventual consistency between concurrent editing operations. The algorithm is based on the GOTO inclusion transformation (cf. [44]) to allow each participant to write in real-time on his or her local document while remote modifications of other participants are merged into the document concurrently. Jupiter is an algorithm with a centralised server component operated by the host of a Saros session and local client components for each session participant. The initial implementation of Jupiter used in Saros was taken from the editor ACE [4] and subsequently tested, corrected and expanded for the use in Eclipse [47] and to support concurrent undo following the algorithm from [43].

Connecting to Eclipse using the bridge component needs to be achieved on several levels. First and foremost, it is necessary to capture editing events from the text editors in Eclipse and to be able to replay these—possibly adjusted by the Jupiter algorithm multiple times—on a remote client. This also includes being notified, when users want to undo their own changes. Second, saving files and closing editors without saving need to be captured, as these changes reset internal representations of the editors. Third, interactions with editors and viewports such as scrolling, selecting text or opening new files must be detected and executed. Fourth, the file system—called workspace in Eclipse—must be monitored for files and folders being modified by other means than editors such as by build-scripts, file-operations or other applications.

Most of these operations could be captured and replayed by studying the standard text editors available in Eclipse and replicating their behavior. Unfortunately it has been found that complications arise when one considers saving and reverting in the distributed context and when trying to prevent users in the role of observers from modifying files and their content. For instance, we found that no mechanism exists in Eclipse to reliably prevent a file from being saved by an observer, which implied that files cannot be kept consistent at the disk level using Saros. Rather, only when a driver saves we can replicate the saving to disk.

This leads to the fourth important component in Saros—the consistency watchdog—which is responsible for checking the consistency of files in cases when the Jupiter algorithm cannot help us, such as files being changed externally to Eclipse and thus outside the mechanisms by which we can capture them as incremental changes. The consistency watchdog to this end will regularly calculate check-sums of open files on the host side and verify files on the client side. The consistency watchdog is integrated with Jupiter to discard check-sums, which are outdated because of concurrent editing activities. It is described in more detail in [23].

Finally, Saros includes a feedback module for scientific data gathering, asking users to take surveys at the end of session. Following sound ethical conduct for Internet research based on activity data [17], the data gathering is strictly opt-in and anonymous, and can be disabled at any time by the user. We maintain a web-site which lists all the information we currently collect[4] such as length of session, number of participants, role-changes or characters written concurrently.

4. AWARENESS

Saros is designed to manage and map ways to collaboration in a distributed environment which are normally performed co-locately. This implies that most of the benefits of human face-to-face interaction need to be compensated by Saros. For instance, being co-located with others in front of a computer, one can easily see whether the other person is paying attention, how she is feeling or what she is doing through body-language, facial expressions and gestures. The term awareness is used to denote the understanding one person has about the shared environment and activities of others [14]. Awareness support in the context of software means how an application is able to convey such information about a collaboration partner who is not co-located.

In the case of DPP, several awareness aspects are relevant

[2]`http://www.picocontainer.org/`
[3]`http://www.igniterealtime.org/projects/smack/`
[4]`https://www.inf.fu-berlin.de/w/SE/DPPFeedback`

to maintain the benefits and work quality improvements of co-located collaboration in a distributed environment. In most cases this information can be gathered from the context information, which then has to be used or visualised to improve the usability and usage value for the distributed team members in a collaboration session.

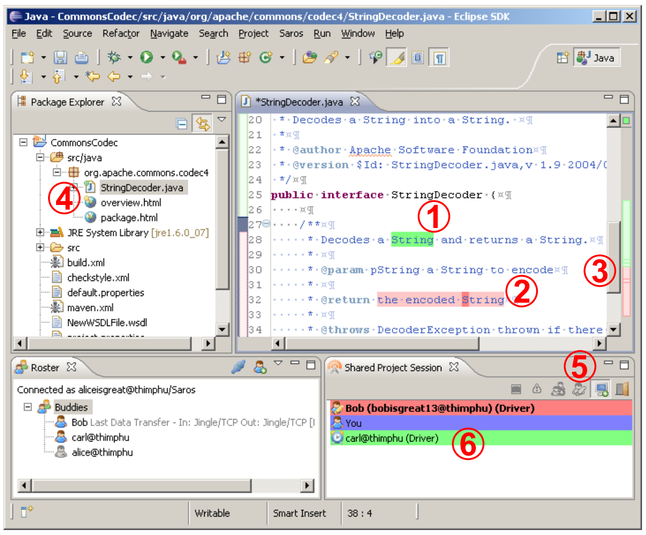
First, presence awareness is necessary to establish context information about who is present and available in a virtual team. Second, workspace awareness refers to the information a user has about work artefacts currently viewed, used or being worked on by others. Third, task awareness can be used as a term orthogonal to workspace awareness asking for the goal and its immediate steps currently pursued by a user.

Saros uses the following features to create awareness of participants, which are highlighted in Figure 2: (1) The cursor and text selection of each user is shown in his or her color. Users can point out and highlight code sections to each other by using this feature. (2) The last twenty text edits written by a driver are highlighted in each editor. Just as a "mouse trail" can be used to raise the awareness of a fast moving mouse cursor [2], the additional colored trail makes it easier to track the contributions of individual authors. (3) The relative position of a user within each file is shown as a colored bar inside the scroll-bar. Users can click on this viewport annotation to jump to the screen that is visible by the respective user. (4) Files opened or currently viewed by a user are highlighted in the package explorer using several small dots. Hereby it becomes easier for a user to identify the sections in a project which are currently being viewed or edited and could potentially be used to show only a partial view of interest of the project as in Eclipse Mylyn [26]. (5) Combining the awareness information inside each file and between files, Saros offers to keep the local viewport automatically in sync with a chosen user. This enables a "follow-mode" in which the local user can concentrate on the viewport as seen by the user being followed, thus staying close to the observer role in the pair programming paradigm. Unlike co-located PP in DPP, it is possible to stop following at any time and peruse code independent of the partner. (6) Saros accentuates whether a user is currently using Eclipse or has another application such as web browser active. This is necessary since users have frequently expressed that while they see that another user has the same viewport in Eclipse they are unsure whether the other user is seeing this viewport or is currently doing something else.

Similar to [7] and [20] we found that an audio channel is essential to compensate for the lost face-to-face interaction and is superior to text-based chat. As Saros does not include Voice over IP (VoIP) capabilities, a separate application such as Skype or Mumble must be run in parallel.

5. SCENARIOS

Software development includes several forms of collaboration. In companies which interact globally or in Open Source projects it is often not a given condition that the team members are co-located. The goal of Saros is to realise collaboration which normally takes place face-to-face in distributed environments. This section will illustrate four usage scenarios of distributed collaboration to give an impression of the concrete possibilities of using Saros. For all scenarios we thus assume the following: (1) Handsfree audio communication is readily available for the participants



```
 * Decodes a String into a String.
 *
 * @author Apache Software Foundation
 * @version $Id: StringDecoder.java,v 1.9 2004/(
 */
public interface StringDecoder {
    
    /**
     * Decodes a String and returns a String.
     *
     * @param pString a String to encode
     *
     * @return the encoded String
     *
     * @throws DecoderException thrown if there
```

Connected as aliceisgreat@thimphu/Saros

Bob Last Data Transfer - In: Jingle/TCP Out: Jingle/TCP [I

carl@thimphu

alice@thimphu

Bob (bobisgreat13@thimphu) (Driver)

You

carl@thimphu (Driver)

Figure 2: Various awareness features used in Saros such as (1) selection, (2) text edits, and (3) viewports highlighted in each users' color, (4) opened and active files by current drivers, (5) button for activating the follow-mode, and (6) information about Eclipse being the foreground window.

either via a normal telephone or VoIP. (2) Each session can be confined to some parts of a software project such as a particular module or layer. This might be interesting to speed up initial synchronization or for information hiding reasons, and is supported by the partial sharing feature in Saros and thus not further discussed below.

The scenarios resulted from studying the literature (e.g. [9]) as well as observing sessions in real work contexts.

5.1 Code Walk-through

A first possible use case of distributed collaboration is the introduction of a new member in a software development team. To familiarise the new developer with the source code, it is often necessary to show and explain certain code parts. Most likely an experienced developer will act as teacher and navigate through the code and highlight important code sections. The student will try to follow the mentor's thoughts and ask questions to gain a better understanding of what the teacher is talking about.

In a distributed environment or if the teacher is not co-located, it would be nice to have a tool which supports the roles of the leader and one or more students from each person's remote computer. Saros can realise this scenario in which one or more students can see the currently opened file and viewport in this file and the selections of another person.

To follow the teacher moving around in the source code, Saros provides the follow-mode in which students can track their teacher.

At this point, a simple screen sharing application would equally be able to provide this scenario, but Saros offers the following advantages: (1) At the end of Saros a session, each participant will have an identical copy of the shared project and could continue learning or start working with it. With a screen sharing application, only the host of a session has a copy locally. (2) Each student has the possibility to use his or her individual cursor to highlight interesting parts in the code to facilitate gesturing. A screen sharing application only provides a shared cursor and thus only one selection. (3) As will be shown in the next scenario, the student also has the possibility to leave the follow-mode and investigate code independently.

5.2 Code Review

A second scenario in software development is performing a code review, which is particularly common in Open Source development for assuring quality [41]. Here one or more experienced developers will look at one particular method, class, or module searching for defects and architectural weaknesses. In contrast to the previous scenario, which is essentially a unidirectional transfer of knowledge in the code review case, all developers are expected to contribute and find problems. Each will bring their own experience with the particular code and their own need for clarifying existing assumptions about the code into the session.

Assuming again that people want to conduct a code review which is not co-located, Saros can support this scenario by default. The main extension to the previous scenario is the desire to have both periods of shared investigation and independent analysis. Saros is ideally suited for this, since each developer has the opportunity to leave the follow mode, go their own way and resume following any of the other developers later on.

5.3 Distributed Pair Programming

Distributed pair programming comes into play when developers want to do pair programming but are not co-located. Then they need a tool which maps the roles of driver and observer and their tasks and responsibilities by allowing real-time editing, common code view and controlling the availability of mouse and keyboard (so-called floor control).

Saros maps the roles driver and observer and their corresponding tasks closer to reality. Two distributed developers can pair together in a programming session. To this end, one of them—the host—invites the other and naturally assumes the role of the driver. He or she has control over mouse and keyboard. The other developer is assigned to be the observer initially. Role changes can then be performed by the host to support the frequent changes in role assignment suggested by XP [3].

5.4 Distributed Party Programming

In contrast to the previous scenario, the collaboration in the case of SbS is much more loosely coupled. Developers ask for each other's help frequently but only if needed and for limited periods of time. In this scenario it is easy to see the downside of co-located SbS because only a limited number of co-developers can sit in close proximity.

Saros provides a solution to this problem, which we call distributed party programming. All members of a distributed team have the possibility to form a virtual team by adding each other to their project roster which is equivalent to a buddy list in an instant messaging application. Thus everybody can work independently in his or her project but by using the project roster he or she is aware of the presence of all other members in the virtual team. When one team member wants to ask one or more team members for help, a Saros session can be started. Then the asking person can navigate through the project's directory tree and highlight certain parts which are relevant for explaining the problem and its context while the other participants of the session are in follow-mode. Once a solution is found, team members can 'leave the room' and switch back to working independently.

6. PRESENT STATE AND PROBLEMS

This section will outline the present state of success the Saros project has achieved and will also discuss relevant problems to overcome.

In terms of the classification system for Open Source projects offered by English and Schweik, Saros can be considered a "success in the growth stage" [15]. The project has been under active development for more than one year, consistently creating a functioning release once every month. According to SourceForge.net statistics, the project has been downloaded at a growing rate, averaging approximately 5000 downloads per month in the period August – December 2009.

Furthermore, thanks to our user feedback and survey systems, we can argue that Saros has been actively utilized and found useful by a number of people. Our on-line survey system asks users to voluntarily fill out a short questionnaire regarding their experience of using the software. One question is multi-choiced – "How did your latest Saros session work out?" – to which the responses are:

- Everything went great.
- Everything went as expected.
- There were some minor problems, but we were able to complete our task(s).
- There were some major problems that prevented us from completing our task(s).
- Other

Our survey system, as well as being a useful form of feedback, has recorded 29 completed surveys so far, 22 of which reported either "Everything went great" or only minor problems.

In addition to this involvement with essentially anonymous users, we have been attempting to integrate Saros into specific existing software projects in a systematic and measured way. In the ongoing process of trying to introduce Saros into Open Source projects, we have so far experienced some difficulties. After selecting an Open Source project, we typically contact the administrator(s), informing them of Saros and the possibilities that should arise from using the software as part of their development work. Despite many very positive reactions, some reservations have been expressed by those we contacted. The reasons for these reservations can mostly be attributed to either being doubtful of PP, perhaps because of unfavourable past experience, or being doubtful that DPP is actually a useful pursuit.

As an ongoing concern we are seeking ways to overcome such doubts and reservations on the part of potential users. As stated above, there sometimes exists bias against PP, but, as has already been shown in section 5, Saros is useful for a number of different modes of collaboration which differ from PP. As a consequence we have decided to emphasise the different scenarios, show PP simply as one of many possible scenarios and point out the other possible usage scenarios of Saros to the contacted projects. When possible we now try to identify typical work flows or events in the respective projects and to point out how Saros may be used in those contexts. When we identify such a project specific context, we argue the benefits of using Saros in this context and also offer our assistance establishing Saros.

7. ONGOING WORK

Saros continues to be developed by a team of researchers and students at the Freie Universität Berlin. This work includes on-going maintenance and the addition of new features. At present a number of these features under development are particularly noteworthy because of their potential for improving the collaborative abilities of Saros.

7.1 Saros as a Research Platform

Since Saros is primarily useful in the context of distributed work, it becomes an interesting question how to conduct research on DPP in general and the use of Saros specifically. In contrast to a laboratory experiment, it is much more difficult to observe users of Saros and gather their feedback. To alleviate this problem, a feedback module is included in Saros, which can gather usage data and redirect users to a survey after the end of session. In particular, the survey (described in Section 7) has offered us several insights about the needs of our users that would have been hard to discover otherwise. For instance, various users have asked us to support the sharing of several projects in Eclipse simultaneously, because their build environment using Apache Maven[5] splits a conceptual project into many Eclipse projects along module boundaries. The data gathering component is similar to Hackystat [25] or ElectroCodeoGram [38] and captures "actual process" data in medium granularity, such as number of concurrent editing events, role changes between observer and driver, etc. We have only begun to explore the potential of this data for increasing our understanding of DPP.

7.2 Screen Sharing

In general, screen sharing allows users to transmit either a portion or the entirety of their screen image for other remote users to see. Saros will provide screen sharing functionality, thus enabling users to share with their collaborators a real-time image of what is displayed on their screen in a convenient way.

This feature is designed so that session participants can control a "virtual camera" on their desktop with the mouse pointer. The portion of the screen captured within the view is sent via a real-time stream for all other collaborators to see.

7.3 VoIP

As has already been explained, Saros makes use of the XMPP Smack library to enable Saros clients to communicate. Jingle, the aforementioned extension to XMPP, provides multimedia interactions for clients, and current work by the Saros project includes using Jingle to implement VoIP functionality. This will integrate into Saros the ability to provide collaborating developers with spoken communication, which is essential for raising awareness and providing fast interactive communication. This functionality will capitalise on previous work done in CSCW[6] research that demonstrates the importance of verbal communication in on-line collaboration (e.g. [24]), and will also address requests for this feature from the user feedback received by Saros.

[5]`http://maven.apache.org`

[6]Computer Supported Cooperative Work

7.4 Plug-in Compatibility

Saros reuses and extends existing Eclipse components whenever possible to leverage the development effort already spent by third parties. As, however, Saros is not alone in this respect and can be expected to be used alongside any number of other plug-ins, Saros and other plug-ins must function acceptably with one another. We focus a systematic testing effort towards verifying that a highly relevant subset of plug-ins reach a level of acceptability.

At present we are testing programming language plug-ins. So far we have tested the functioning of plug-ins for Java, C/C++, PHP, Rails, Jaxer, Python, Adobe Air and TypoScript.[7]

We will continue to submit these plug-ins to testing as environmental factors change over time, and will include further plug-ins in our testing approach, notably built tools such as Apache Maven and plug-ins for version control such as Subclipse[8]. Plug-ins that are found to function erroneously with Saros will be investigated and we will try to adapt Saros (if required) to support them also.

8. RELATED TOOLS

Over the last decade several tools supporting DPP has been developed all with their relative strengths and weaknesses [46]. At the highest level there are two kinds collaboration tools. On the one hand there are screen sharing and remote desktop applications such as VNC or Microsoft Netmeeting. These applications replicate the desktop of a user and permit remote users to collaborate by taking control of mouse and keyboard [20]. On the other hand are collaboration awareness tools which extend an application in specific areas only to create a shared collaboration space. Collaboration awareness tools can then be further sub-classified into collaborative editors such as SubEthaEdit or A Collaborative Editor (ACE) [4] (which are designed for editing arbitrary texts), and specialised editors (which focus on a particular application domain such as software development, and are often in integrated existing applications). The latter camp consists primarily of research prototypes and notably include COPPER [27], TUKAN [39], Moomba [33], Sangam [22, 28], and XPairtise [40]. Unfortunately none of these tools are publicly available, with the exception of XPairtise, whose last release was in 2008. A recent development is the increased availability of collaborative editors as web-applications such as Mozilla Bespin[9] for web-design in HTML, CSS and JavaScript and Google Docs[10] for collaborative editing of spreadsheets and text documents.

9. CONCLUSION

In this paper we have introduced Saros, an Eclipse plug-in for simultaneous collaborative software development. We have highlighted that Saros is more than a tool for DPP,

[7]A complete list of compatible plug-ins is available at `https://www.inf.fu-berlin.de/w/SE/DPPCompatiblePlugins`.

[8]`http://subclipse.tigris.org/`

[9]`http://bespin.mozilla.com/`

[10]`http://docs.google.com`

because there exist many different scenarios for the use of Saros, such as for reviews or knowledge transfer (see section 5). The descriptions of the awareness features (see section 4) and the technical implementation (see section 3) show that Saros addresses the design principles for supporting collaboration identified by Cox and Greenberg [10] and should help to overcome some of the hurdles of developers being distributed. For example, Saros facilitates gesturing and deictic references via highlighting functionality and timely feedback of actions via the use of the Jupiter algorithm. Nevertheless, more empirical evaluation is needed to get detailed information about the usefulness of the awareness functionalities and to develop ideas for the improvement of Saros. Concerning this matter we have described the feedback module as one way to gather scientific data for future work. With the aid of this module Saros also provides a platform for scientific studies of the social practices associated with distributed software development, namely distributed pair, side-by-side and party programming. Finally, we have described our ongoing efforts to improve Saros and make it available in corporate and Open Source settings.

Acknowledgments.
Many thanks are extended to R. Djemili, B. Gustavs, O. Rieger, C. Jacob, M. Rintsch, S. Ziller, E. Rosen, L. Dohrmann, S. Szücs, T. Sóti, W. Polcwiartek, E. Starkmann, R. Kunze, S. Lau, H. Staib, O. Loga, and A. Kiwitt who have worked on Saros and the many Saros users who have given us feedback.

10. REFERENCES

[1] P. Baheti, E. Gehringer, and D. Stotts. Exploring the efficacy of distributed pair programming. In *Extreme Programming and Agile Methods – XP/Agile Universe 2002*, volume 2418 of *Lecture Notes in Computer Science*, pages 387–410. Springer, 2002.

[2] P. Baudisch, E. Cutrell, and G. Robertson. High-density cursor: a visualization technique that helps users keep track of fast-moving mouse cursors. In *INTERACT'03; IFIP TC13 International Conference on Human-Computer Interaction, 1st-5th September 2003, Zurich, Switzerland*, 2003.

[3] K. Beck. *Extreme Programming Explained: Embrace Change.* Addison-Wesley Professional, 1999.

[4] M. Bigler, S. Räss, and L. Zbinden. ACE—a collaborative editor. Project Report. `http://swiki-lifia.info.unlp.edu.ar/ContextAware/uploads/25/ACE_-_A_Collaborative_Editor_-_Report_Evaluation_Algorithms.pdf`. Last visited 2009-12-18, Apr. 2005.

[5] T. Bipp, A. Lepper, and D. Schmedding. Pair programming in software development teams - an empirical study of its benefits. *Information and Software Technology*, 50(3):231–240, 2008.

[6] S. Bryant, P. Romero, and B. du Boulay. Pair programming and the mysterious role of the navigator. *International Journal of Human-Computer Studies*, 2008.

[7] G. Canfora, A. Cimitile, and C. A. Visaggio. Lessons learned about distributed pair programming: what are the knowledge needs to address? In *Twelfth International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 314–319, Los Alamitos, CA, USA, 2003. IEEE Computer Society.

[8] J. Chong and T. Hurlbutt. The social dynamics of pair programming. In *ICSE07: Proceedings of the 29th Int'l Conf. on Software Engineering*, pages 354–363, Washington, DC, USA, 2007. IEEE Computer Society.

[9] A. Cockburn. *Crystal Clear: A Human-Powered Methodology for Small Teams.* Addison Wesley, 2004.

[10] D. Cox and S. Greenberg. Supporting collaborative interpretation in distributed groupware. In *CSCW '00: Proceedings of the 2000 ACM conference on Computer supported cooperative work*, pages 289–298, New York, NY, USA, 2000. ACM.

[11] P. Dewan, P. Agarwal, G. Shroff, and R. Hegde. Distributed side-by-side programming. In *CHASE '08: Proceedings of the 2008 international workshop on Cooperative and human aspects of software engineering*, pages 48–55, Los Alamitos, CA, USA, 2009. IEEE Computer Society.

[12] R. Djemili, C. Oezbek, and S. Salinger. Saros: Eine Eclipse-Erweiterung zur verteilten Paarprogrammierung. In *Software Engineering 2007 - Beiträge zu den Workshops*, Hamburg, Germany, Mar. 2007. Gesellschaft für Informatik.

[13] M. A. Domino, R. W. Collins, and A. R. Hevner. Controlled experimentation on adaptations of pair programming. *Information Technology and Management*, 8(4):297–312, 2007.

[14] P. Dourish and V. Bellotti. Awareness and coordination in shared workspaces. In *CSCW '92: Proceedings of the 1992 ACM conference on Computer-supported cooperative work*, pages 107–114, New York, NY, USA, 1992. ACM.

[15] R. English and C. M. Schweik. Identifying success and abandonment of Free/Libre and Open Source (FLOSS) commons: A classification of Sourceforge.net projects. *UPGRADE*, VIII(6):54–59, Dec. 2007.

[16] M. Fowler. Inversion of control containers and the dependency injection pattern. `http://www.martinfowler.com/articles/injection.html`, Jan. 2004.

[17] M. S. Frankel and S. Siang. Ethical and legal aspects of human subjects research on the internet. Published by AAAS online , June 1999.

[18] S. Freudenberg (née Bryant), P. Romero, and B. du Boulay. "Talking the talk": Is intermediate-level conversation the key to the pair programming success story? In *AGILE 2007*, pages 84–91, Washington, DC, USA, 2007. IEEE Computer Society.

[19] C. Gutwin and S. Greenberg. The effects of workspace awareness support on the usability of real-time distributed groupware. *ACM Trans. Comput.-Hum. Interact.*, 6(3):243–281, 1999.

[20] B. Hanks. Empirical evaluation of distributed pair programming. *International Journal of Human-Computer Studies*, 66(7):530–544, 2008. Collaborative and social aspects of software development.

[21] J. E. Hannay, T. Dybå, E. Arisholm, and D. I. Sjøberg. The effectiveness of pair programming: A meta-analysis. *Information and Software Technology*,

51(7):1110–1122, 2009.

[22] C.-W. Ho, S. Raha, E. Gehringer, and L. Williams. Sangam: a distributed pair programming plug-in for Eclipse. In *eclipse '04: Proceedings of the 2004 OOPSLA workshop on eclipse technology eXchange*, pages 73–77, New York, NY, USA, 2004. ACM.

[23] C. Jacob. Weiterentwicklung eines Werkzeuges zur verteilten, kollaborativen Softwareentwicklung. Diplomarbeit, Institut für Informatik, Freie Universität Berlin, Berlin, Apr. 2009.

[24] C. Jensen, S. D. Farnham, S. M. Drucker, and P. Kollock. The effect of communication modality on cooperation in online environments. In *Proceedings of the SIGCHI conference on human factors in computing systems*, volume 2, pages 470–477, 2000.

[25] P. M. Johnson, H. Kou, J. M. Agustin, Q. Zhang, A. Kagawa, and T. Yamashita. Practical automated process and product metric collection and analysis in a classroom setting: Lessons learned from hackystat. In *ISESE '04: Proceedings of the 2004 International Symposium on Empirical Software Engineering*, pages 136–144, Washington, DC, USA, 2004. IEEE Computer Society.

[26] M. Kersten and G. C. Murphy. Mylar: a degree-of-interest model for IDEs. In *AOSD '05: Proceedings of the 4th international conference on Aspect-oriented software development*, pages 159–168, New York, NY, USA, 2005. ACM.

[27] H. Natsu, J. Favela, A. L. Morán, D. Decouchant, and A. M. Martinez-Enriquez. Distributed pair programming on the web. In *Fourth Mexican International Conference on Computer Science (ENC'03)*, pages 81–88, Los Alamitos, CA, USA, 2003. IEEE Computer Society.

[28] K. Navoraphan, E. F. Gehringer, J. Culp, K. Gyllstrom, and D. Stotts. Next-generation dpp with sangam and facetop. In *eclipse '06: Proceedings of the 2006 OOPSLA workshop on eclipse technology eXchange*, pages 6–10, New York, NY, USA, 2006. ACM.

[29] J. R. Nawrocki, M. Jasiñski, L. Olek, and B. Lange. Pair programming vs. side-by-side programming. In *Software Process Improvement*, volume 3792 of *Lecture Notes in Computer Science*, pages 28–38. Springer, 2005.

[30] D. A. Nichols, P. Curtis, M. Dixon, and J. Lamping. High-latency, low-bandwidth windowing in the jupiter collaboration system. In *UIST '95: Proceedings of the 8th annual ACM symposium on User interface and software technology*, pages 111–120, New York, NY, USA, 1995. ACM.

[31] J. T. Nosek. The case for collaborative programming. *Communications of the ACM*, 41(3):105–108, 1998.

[32] L. Prechelt, U. Stärk, and S. Salinger. Types of cooperation episodes in Side-by-Side programming. In *Proc. 21st Annual Workshop of the Psychology of Programming Interest Group (PPIG '09)*, Limerick, July 2009. ppig.org.

[33] M. Reeves and J. Zhu. Moomba – A collaborative environment for supporting distributed Extreme Programming in global software development. In *Extreme Programming and Agile Processes in Software Engineering*, volume 3092/2004 of *Lecture Notes in Computer Science*, pages 38–50. Springer, Berlin / Heidelberg, May 2004.

[34] J. Rosenberg, R. Mahy, P. Matthews, and D. Wing. Session traversal utilities for NAT (STUN). Request for Comments 5389, Internet Engineering Task Force, Oct., 2008.

[35] P. Saint-Andre. Extensible messaging and presence protocol (XMPP): Core. Request for Comments 3920, Internet Engineering Task Force, Oct., 2004.

[36] P. Saint-Andre. Jingle: Jabber does multimedia. *IEEE MultiMedia*, 14(1):90–94, Jan. 2007.

[37] S. Salinger, L. Plonka, and L. Prechelt. A coding scheme development methodology using grounded theory for qualitative analysis of pair programming. *Human Technology: An Interdisciplinary Journal on Humans in ICT Environments*, 4:9–25, 2008.

[38] F. Schlesinger and S. Jekutsch. ElectroCodeoGram: An environment for studying programming. In *Workshop on Ethnographies of Code, Infolab21*, Lancaster University, UK, 30-31 March 2006.

[39] T. Schümmer and J. Schümmer. Support for distributed teams in eXtreme Programming. In *Extreme programming examined*, pages 355–377. Addison-Wesley, Boston, MA, USA, 2001.

[40] T. Schümmer and S. Lukosch. Supporting the social practices of distributed pair programming. In *Groupware: Design, Implementation, and Use*, volume 5411 of *Lecture Notes in Computer Science*, pages 83–98. Springer, 2008.

[41] J. Stark. Peer reviews as a quality management technique in Open-Source software development projects. In *ECSQ '02: Proceedings of the 7th International Conference on Software Quality*, pages 340–350, London, UK, 2002. Springer-Verlag.

[42] D. Stotts, L. Williams, N. Nagappan, P. Baheti, D. Jen, and A. Jackson. Virtual teaming: Experiments and experiences with Distributed Pair Programming. In *Extreme Programming and Agile Methods — XP/Agile Universe 2003*, volume 2753/2003 of *Lecture Notes in Computer Science*, pages 129–141. Springer, Berlin / Heidelberg, Sept. 2003.

[43] C. Sun. Undo as concurrent inverse in group editors. *ACM Trans. Comput.-Hum. Interact.*, 9(4):309–361, Dec. 2002.

[44] C. Sun and C. Ellis. Operational transformation in real-time group editors: issues, algorithms, and achievements. In *CSCW '98: Proceedings of the 1998 ACM conference on Computer supported cooperative work*, pages 59–68, New York, NY, USA, 1998. ACM.

[45] L. Williams, R. R. Kessler, W. Cunningham, and R. Jeffries. Strengthening the case for pair programming. *IEEE Software*, 17(4):19–25, 2000.

[46] D. Winkler and S. Biffl. Evaluierung von Werkzeugen für Distributed Pair Programming: Eine Fallstudie. In *In Proceedings of the 2009 Conference on Software & Systems Engineering Essentials*, Berlin, Germany, May 2009.

[47] S. Ziller. Behandlung von Nebenläufigkeitsaspekten in einem Werkzeug zur Verteilten Paarprogrammierung. Diplomarbeit, Institut für Informatik, Freie Universität Berlin, Berlin, Oct. 2009.