Intelligent Information Management, **2010, 2, 487-497**
doi:10.4236/iim.2010.28059 Published Online August 2010 (http://www.SciRP.org/journal/iim)

Scientific Research

Analysis and Design of Distributed Pair Programming System*

Wanfeng Dou, Yifeng Wang, Sen Luo
School of Computer Science and Technology, Jiangsu Research Center of Information Security & Confidential Engineering, Nanjing Normal University, Nanjing, China
E-mail: *douwanfeng@njnu.edu.cn*
Received June 7, 2010; *revised July* 10, 2010; *accepted August* 11, 2010

Abstract

Pair Programming (PP) that has gained extensive focus within pedagogical and industrial environments is a programming practice in which two programmers use the same computer to work together on analysis, design, and programming of the same segment of code. Distributed Pair Programming (DPP) system is a programming system to aid two programmers, the driver and the navigator, to finish a common task such as analysis, design and programming on the same software from different locations. This paper first reviews the existing DPP tools and discusses the interaction and coordination mechanism in DPP process. By means of activity theory and language-action theory, some basic requirements of the DPP system are presented. Then, a design framework of such system and functions of each sub-system are deeply analyzed. Finally, a system prototype is implemented by plug-in style in Microsoft Visual Studio environment.

Keywords: Pair Programming, Distributed Pair Programming, Software Engineering, Extreme Programming

1. Introduction

In recent years, agile software methodologies have attracted increasing interest within pedagogical and industrial environments, with extreme programming being considered the most important of these agile methodologies [1]. In the agile manifesto, the authors state twelve general principles that all highlight the importance of flexibility and collaboration. One of these techniques, which are being adopted by software development group, is known as Pair Programming (PP), in which two developers work side by side, on the same computer, to collaboratively produce a design, an algorithm, a code, etc [2]. Taking these principles would imply a distributed application of agile methods, such as distributed extreme programming. Although some tools have been developed to better support distributed agile software development, there is still a need for additional research on tools and processes for distributed extreme programming, especially for solutions that extend the most obvious solution of providing a shared code editor. As the trend towards global software development continues, pair programming in which two developers are required to work in face-to-face interaction don't meet the need of global software development. This needs to create computer programs through pair programming practice where developers are located in different workstation but they collaborate simultaneously to solve the same problem. This approach is called Distributed Pair Programming (DPP). This paper focuses on reviewing the existing distributed pair programming systems, and presents system design and implementation. This paper has six sections. After this introduction, Section 2 gives a related work about DPP tools. Section 3 discusses analysis approach of DPP based on activity theory and language theory. The requirements of DPP tool are presented in Section 4. Section 5 describes the design and implementation of prototype system. Section 6 draws conclusions.

*This project was financed partly by the eleventh five-years plan from Jiangsu Education Science in 2009, No: D/2009/01/093.

2. Related Work

2.1. Pair Programming

Extreme programming, also known as XP, includes a set of principles and practices for the rapid development of high quality software. XP identifies 12 best practices of software development and takes them to an extreme. Pair programming originated in industry as a key component of the XP development methodology. As the name suggests, pair programming involves two programmers

working at the same computer to create code or analyze requirements and develop design and etc. This provides a mechanism for real-time problem solving and real-time quality control [2]. One programmer acts as the driver, who actively writes code or design document and has control of the keyboard and mouse. The other partner acts as the navigator, who helps plan as well as identify and prevent any syntactic or strategic deficiencies in code or design document, thinks of alternatives and asks questions [3]. The collaborator may exchange roles at any time during the pair programming session, or not at all.

The concepts underlying pair programming are not new [4,5], but pair programming has only recently attracted significant attention and interest within the software industry and academia. Several previous controlled experiments have concluded that pair programming has many benefits over solo programming [6]. Pair programming has significant improvements in code reviewing and various others measures of quality of the programs being developed including lower duration with only minor additional overhead in terms of a measure of cost or effort [4-5]. But, with respect to time taken and improvement of functional correctness of the software product compared with Solo programming showed no positive effects of pair programming [7]. The reasons are the difference in sample populations (e.g., students or professionals), study settings (e.g., amount of training in pair programming), lack of power (e.g., few subjects), and different ways of treating the development variables (e.g., how correctness was measured and whether measures of development times also included rework), and task complexity (e.g., simple dependent tasks, or complicated projects) [8-9].

Pair programming originated in industry as a key component of the extreme programming development methodology. As the name suggests, pair programming involves two programmers working at the same computer to create code or analyze requirements and develop design and etc. This provides a mechanism for real-time problem solving and real-time quality control [2]. One programmer acts as the driver, who actively writes code or design document and has control of the keyboard and mouse. The other partner acts as the navigator, who helps plan as well as identify and prevent any syntactic or strategic deficiencies in code or design document, thinks of alternatives, and asks questions. The collaborator may exchange roles at any time during the pair programming session, or not at all. Pair programming has been shown to be an effective pedagogical approach in teaching courses such as introductory computer science [10-11]. Undergraduate software engineering [12], and graduate object-oriented software development [13]. Studies have shown that pair programming creates an environment conducive to more advanced, active learning and social interaction, leading to students being less frustrated, more confident and more interested in IT [14], and also improve retention of women in computer science [15]. Pair programming encourages students to interact and cooperate with partners in their classes and laboratories, or development teams, thereby creating a more collaborative environment in which pairs discuss problems, brainstorm each other, and share knowledge. Pair programming also benefits the teaching staff. A pair of students can always analyze and discuss the low-level technical or procedural questions that typical burden the teaching staffs in the laboratory, hence there are fewer questions to be dealt with.

Distributed pair programming is a style of programming in which two programmers who are geographically-distributed and synchronously collaborating over the Internet work together on the same software artifact. Comparing with pair programming, DPP decreases the scheduling issues that arise for developers trying to schedule collocated pair programming. Making DPP technology available to students increases the likelihood that they will pair program. Trying distributed pair programming increases the likelihood that students will pair program remotely in the future. While DPP has been shown to be better than distributed non-pair programming, DPP is not perfect. The main reason is to require a better tool to support the DPP process.

2.2. Tools of Distributed Pair Programming

In pair programming environment, however, obstacles such as limited facilities, geographic separation, and scheduling often present challenges to collocated pair programming. DPP enables students or developers to collaborate from different locations to work on their programming projects remotely. One of the main trends in software development has been the globalization of the software industry. Motivating factors behind this trend include hiring qualified programmers in different cities and countries for software companies, placing development group closer to their client's location, creating quickly virtual development groups, and working continuously on critical projects by working on different time zones for groups [16].

Researchers have proposed several tools to better support distributed pair programming [17-22].These existing tools adopt either an application sharing approach to enhance an existing editor suite or provide customized tools that include various groupware features such shared awareness [17]. Customized groupware tools do not support all of the features needed by pair programming and thus limit partner's ability to successfully accomplish their work. On the other hand, application sharing solutions lack process support and thus met collaboration awareness.

2.2.1. Application Sharing Tools

JAZZ system is an example with an application approach [18]. It is an extension of eclipse that supports the XP and workflows in asynchronous interaction. JAZZ allows users to stay aware of co-workers and initiate chat sessions, and be invited to a synchronous pair programming session using an application sharing system. JAZZ implements a shared editing plug-in that provides a synchronous shared code editor based on operation transformation approach. But this plug-in is not integrated into the workflow of pair programming, and thus does not provide awareness and has no explicit switching of roles.

MILOS is another application sharing system [19]. IT provides awareness of co-present users and allows users to initiate pair programming sessions using application sharing like JAZZ. MILOS makes use of existing IDEs and integrates single-user development environments into pair programming settings. But, application sharing approach does not support flexible pairing such as one-to-many pairing way, and role switching. TUKAN is a special purpose groupware for all phrase of the XP process [20]. It provides a shared editor for manipulating code together and users can highlight important code using a remote selection. Moomba extends the awareness tools of TUKAN and support Java IDE where the users can use a shared java editor [21]. However, TUKAN and Moomba use ENVY environment and are built as a proprietary tool and thereby cannot provide the same domain specific tool support as it is present in modern IDEs. This is one of the reasons why they have not gain high popularity.

Eclipse is a popular and more open environment that allows closer coupling of the developing IDEs [4]. Coordination work can be integrated into Eclipse in the internal browser window or special-purpose planning plug-in. The Eclipse Communication Framework (ECF) aims at integration a collaboration infrastructure with the IDE. 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 [22]. Sangam use an event-driven design for this plug-in. There are three basic components in Sangam: event interceptor, message server, and event reproducer. 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. When the event reproducer receives a message and interacts with Eclipse to perform the driver's action. Saros plug-in supports driver-navigator interaction in Eclipse in a distributed pair programming session, and provides awareness on files that are opened at the driver's site [4]. Saros includes a shared editor that allows collaborative code creation, and remote selections that allow the navigator to point at relevant pieces of code. Xpairtise is an Eclipse plug-in that offers shared editing, project synchronization, shared program, test execution, user management, built-in chat communication, and a shared whiteboard [4].

RIPPLE is a plug-in for the popular Eclipse integrated development environment in which data on collaborative programming is collected. RIPPLE is designed for use in educational setting to facilitate various forms of collaborative programming problem solving including distributed pair programming and distributed one-to-one tutoring [23]. RIPPLE extends the architecture implemented in Sangam. Compilation and execution of code, as well as generation of console message, are performed directly by Eclipse. However, RIPPLE currently only supports Java programming because the event-driven behavior of it requires that language-specific messages be transmitted between users. The textual dialogue of RIPPLE is an instant-message-style chat program that supports enforced turn-taking in dialogue.

2.2.2. Customized Tools

COLLECE, developed using Java technology, is a groupware system to support synchronous distributed pair programming practices. COLLECE provides a set of tools including editor, session panel, coordination panels, and structured chat [24]. The editor provides a workspace in which the driver inserts or modifies the source code of the program that is being built. The session panel provides a simple awareness of partner that shows the photo and name of each pair. The coordination panels include three coordination tools that allow a collaboration protocol to be established: edition coordination panel, compilation coordination panel, and execution coordination panel. The structured chat is used to express conversational acts that are usually used during program coding, compilation and execution.

COPPER is a synchronous source code editor that allows two distributed software engineers to write a program using pair programming. Its functions include communication mechanisms, collaboration awareness, concurrency control, and a radar view of the documents, among others. COPPER system is based on the C/S architecture. It is composed of three subsystems: collaborative editor, user and document presence, and audio subsystems. The editor is further decomposed into the Editor module and the document server. The Editor module implements a turn-taking synchronous editor and the document server provides document storage, document editing access control, user authentication and permissions, and document presence extensions.

However, low display refresh rate can sometimes be confusing or something significant may be lost in the remote display. The trace of the mouse pointer is another problem if both developers use no same resolution for their monitors. Hence, next-generation tool is still analyzed and studied in terms of requirements of distributed pair programming.

3. Analysis and Interaction in DPP System

3.1. Analysis Based on Activity Theory in DPP System

Activity theory, as a social psychological theory on human self-regulation, is a well suited epistemological foundation for design. Activity theory was first used to design the user interface by Bodker. Later it has been extended and refined by numerous other authors. In particular, activity theory is used to understand cooperative work activities supported by computers [25,26]. Pair programming is a social activity involved two programmers, driver and navigator. This paper use activity theory as a theoretical basis for understanding the cooperative work activities in DPP.

Broadly defined, activity theory is a philosophical framework for studying different forms of human praxis as development processes, with both individual and social levels interlinked. Three of the key ideas of activity theory can be highlighted here: activities as basic unit of analysis, the historical development of activities and internal mediation with activities [25]. Activities—an individual can participate in several at the same time—have the following properties: 1) an activity has a material object; 2) an activity is a collective phenomenon; 3) an activity has an active subject, who understands the motive of the activity; 4) an activity exists in a material environment and transforms it; 5) an activity is a historically developing phenomenon; 6) contradiction is realized through conscious and purposeful actions by participants; 7) the relationships within an activity are culturally mediated.

Y. Engestrom has made an attempt to establish a structural model of the concept activity and culturally mediated relationships within it (**Figure 1**). This structure includes three components, namely subject, object and community, and forms three relationships: subject-object, subject-community and object-community.

This activity model contains three mutual relationships between subject, object and community: the relationship between subject and object is mediated by tools, that between subject and community is mediated by rules and that between object and community is mediated by the division of labor. Each of the mediating terms is historically formed and opens to further development. In this activity model, four subsystems are formed: production subsystem, communication subsystem, assignment subsystem and consumption subsystem.

The production subsystem is used by the subject (e.g., driver and navigator) to manipulate the object into outcome (e.g., analysis, design or programming for a code). In **Figure 1**, the production subsystem involves three components: subject, object and tool. In DPP, this subsystem is a shared editor that can support the synchronous editing, role switching, test execution and file sharing, etc.

Communication subsystem, in **Figure 1**, involves also three components: subject, community and rule. For instance, In DPP, the driver and navigator use this subsystem to communicate each other so as to solve the problems met during pair programming. The driver and navigator as a community should stand by rules. For example, a partner as a role of driver, another must be a navigator. They switch role at intervals. The communication subsystem that includes the relationship and interaction between subject and community should provide chat session, whiteboard and audio or video communication. The communication subsystem must be designed for users to easy discussion on problems and suggestions on their task and further focus on the shared code.

Assignment subsystem builds the relationship between object and community through establishment of the division of labor, that is to say, it assign activity according to social rules and expectation. In DPP, the pair with a driver role is responsible for writing the code using keyboard and mouse, and the other with a navigator role is responsible for reviewing the code written by the partner and gives some suggestions. During DPP, they should switch the role at intervals.

Consumption subsystem describes how subject and community to cooperate to act on object. Consumption process stands for the inherent contradictions of activity system. Although the goal of the production subsystem is to transform the object into outcome, it also consumes the energy and resources of subject and community. The consumption subsystem may plan arrangement and provide the resources for DPP.

In **Figure 1**, the emphasis of analysis of activity system is production subsystem. The production of object is leaded by the results or intention of activity system. For example, the activities of DPP lead to produce the code with high quality. Production subsystem is usually considered to be the most important subsystem. Hence, understanding the production subsystem will be a good start for design of DPP system.

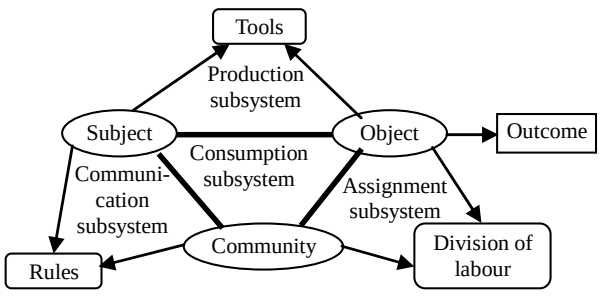

Figure 1. Basic structure of an activity.

3.2. Conversation Model of DPP

In a DPP system, two programmers, the driver and the navigator, work commonly on the same task such as a code, or a design, or an analysis by network and related tools. In order to the efficiency of their programming, the communication of pairs is important to effective cooperation for them. When the driver is editing, the navigator may observe the code or design remotely and at any moment to give suggestions about it, or think about optional solution or strategy. In other one, the driver may request acknowledgement of the pair to it during he/she writes the code. The conversation model is to describe the communicating process between the driver and the navigator so that we clarify how to communicate between them during pair programming. In follow section, we construct a conversation model of DPP by means of language-action theory.

In designing a DPP system for practical situations, we need to consciously focus on the context and application. The structure of the system determines the possibilities for action by the people who use it, and it is this action structure that is ultimately important. Design is ontological. That is what we are participating in the larger design of the organization and collection of practices in which it plays a role. In describing or constructing any system, we are guided by a perspective. This perspective determines the kinds of questions that will be raised and the kinds of solution that will be sought. One can consciously apply a perspective as a guide to design. It will not provide answers to all of the specific design questions, but serves to generate the question that will demand answers.

The language/action perspective is one of the relevant theoretical contributions that have appeared within cooperative work. Cooperative work is coordinated by the performance of language actions, in which the partner become mutually convinced to the performance of future actions and they make declarations creating social structures in which those acts are gathered and interpreted [27]. The language/action perspective has had a significant role with computer supported cooperative work. The PP or DPP is a cooperative activity with two actors, which can be modeled by language/action perspective.

The language/action perspective emphasis pragmatics, not the form of language, but what people do with it. The language/action has five fundamental illocutionary points —things you can do with an utterance [27]: 1) Assertive that commits the speaker to something being the case –to the truth of the expressed proposition; and 2) Directive that attempts to get the header to do something; and 3) Commission that commits the speaker to future course of action; and 4) Declaration that brings about the correspondence between the propositional content of the speech act and reality; and 5) Expressive that expresses a psychological state about a state of affairs.

The need of supporting DPP with suitable computer based tools implies the investigation of the deep aspects of cooperation and clarification. Cooperation clarification, to the extent that is made up of communication and negotiation, can be fully characterized under the assumption that the DPP can be viewed as a special linguistic action game between the driver and the navigator, constituted by asset of rules defining the conversations possible within it. The results of conceptual and experimental research motive the following answer: the driver and navigator spend their time taking commitments for future activities each other, coordinating the programming work, switching role according to the situation, explaining the problems they encounter during pair programming, reviewing the code. This needs to precisely develop conversation between the driver and navigator in order to take commitments for an effective negotiation and coordination of the activities.

A conversation between the driver and the navigator during a DPP process is a sequence of related utterances. The utterance within a conversation can be classified from the pragmatic point of view in some basic categories of speech acts on the basis of their illocutionary point namely, directives (e.g., Request, Acceptance or Rejecting of a promise), commission (e.g., Promise, Count-offer, Acceptance or Rejecting of a commitment, Declaring of commitment fulfillment). Each conversation involves two actors in the DPP: the driver and the navigator, and follows the pattern which defines the possible sequences of speech acts characterizing the specific type of conversation. In accordance with language/action theory, there are also three main types of conversation occurring in any PP. The first is the conversation for action, characterized by the definition of a commitment for doing an action. The driver in the PP can recognize, e.g., the conversations opened by a request, where the driver opening the conversation asks the partner for some activities; the conversation by a promise, where the navigator agrees and provides the support for its fulfillment. The second is the conversation for possibilities, where the pairs discuss a new possibility for the code, in terms of requirements, code structure, language and related knowledge these conversations, when successful and devoted to topics under the competence of the pair, end with a declaration explaining the concept and agreeing with the code. The third is the conversation for clarification, where the pairs cope with or anticipate breakdowns concerning interpretations of conditions of satisfaction for action. The conditions are always interpreted with request to an implicit shared background, but sharing is partial and needs to be negotiated. There is no sharp line between them, but they are accompanied by different moods.

The PP is characterized by a specific organizational rule, which define the roles of pair programming, role switching and compatibility of pairs. These rules can be

expressed in terms of conversation possibilities open to that role. For example, two roles are defined in pair programming mode, the driver and navigator. The driver is responsible for writing the code by the mouse and keyboard and the navigator can view and test the code, and think about the structure and some strategies. The navigator cannot enter any code but can point out the existing problems and request the discussion with the driver. If possible, the pairs can periodically switch role.

The conversation for action forms the central fabric of a DPP. In a simple conversation for action, the driver (A) in a pair programming makes a request to the navigator (B). The request is interpreted as having certain conditions of satisfaction, which characterize a future course of action by B. After the initial utterance (the request), B can accept (satisfaction for action); reject (end of the conversation); or counter-offer with alternative conditions. Each of these in turn has its own possible continuations (e.g., after a counter offer by B, A can accept, reject, or counter-offer again).

The meaning of a language/action exists in the interpretations of a driver and a navigator in a particular situation with their particular backgrounds. The request is an initial utterance, for the driver there are several kinds of request: 1) Help (request for collection of some materials or testing for the codes); and 2) Negotiation (request for clarification of some problems; and 3) Question (request for the design of programming).

Reducing the complexity of a work process and of communicative mode going on within it is that they need to be supported in copying with that complexity [28]. This means that any tools supporting practices of a conversation must broaden and not restrict the range of all kinds of possibilities of its participants. The relationship between conversation and commitments is not a one-to-one one. Making a commitment explicit is sometimes very useful, in particular when we must ensure that it will be completed satisfactorily. Considering a conversation as a sequence of communication events to which can be attached not only documents of any types but also any numbers of commitment negotiations. A DPP procedure includes a set of conversations. Each conversation with a commitment and a title includes a set of events. Each event is a structured message characterized by its completion time, content, associated code, attached documents, its sender and its receiver.

In a DPP procedure, there are a lot of conversation occurring between the driver and navigator. For example, the driver may request a help for some materials with the code from the navigator, or hope to discuss some uncertain programming problems. In some time the driver may request the navigator to test the code written by him/her. The navigator can point out the existing problems during reviewing the code.

A support system of commitment negotiation is required to help the user to understand the context where he/she is negotiating each other, as well as the state of the negotiation. The goal of this conversation model is to develop a theoretical framework for understanding communication within a DPP process.

Conversations are just sequences of communicative events involving two participants, driver and navigator in PP, where each participant is free to be as creative as he/she wants. Conversations can be supported by a system making accessible the sequence of records of the communicative events, together with the documents generated and/or exchanged and with the commitment negotiations steps which occurred during them. Within this model a commitment may be viewed as the respective negotiation steps performed within a conversation by the driver and navigator and by the documents that are attached to them. Any negotiation step of a commitment is characterized by its object, its time and its state. Commitment negotiations are therefore fully transparent to their actors within conversations without imposing any normative constraints upon them comparing to fully scheduleed model of conversation.

4. Requirements of DPP System

Distributed pair programming means that two developers synchronously collaborate on the same design or code from different locations. The results of experiments indicate that DPP is competitive with collocated pair programming in terms of software quality and development time [13], and it is feasible and effective to develop software code using distributed pair programming [29]. Considering the trend of globalization in software development we have aimed at finding out how programmers could effectively apply DPP technique with the use of appropriate groupware tools, and what would be the requirements of such tools. For this purpose we defined a set of requirements of distributed pair programming tool in terms of the analysis of the existing groupware tools and DPP tools [16,22-24], and features of pair programming. According to the technology of Computer Supported Cooperative Work (CSCW) we have identified the following requirements of the DPP tool.

4.1. Shared Editing Integrating Existing Editor

As a source code editor it should highlight keywords based on the programming language being used and not only provide conventional editing tools such as: Cut, Copy, Paste, Find, and Replace, but also the options of compilation and execution of the source code being edited and should notify the users of the error messages reported by the compiler. On the other hand, the existing editors with the integration of developing environment supporting a specific language have very powerful functionalities. Moreover, developers hope to use their fa-

miliar editor or integrating development environment to pair programming, and for some language, for example Java language has several editors or developing environment support editing and compiling source code such as Eclipse, JDK, JBuilder, Visual Café, and etc. It is required that collaborative pair programming tool can integrate these editors or developing environments. However, there are some problems to be solved when paring developers use different editors or developing environments. For example, interoperation is one of the main problems in information exchange between two different editors with the same language due to the differences of their format of editing commands and the parameter options of compiling and executing the source code.

4.2. Shared File Repository

The source code files and related documents being edited should be controlled at the repository. These files and documents should be shared among all members of the development team. Furthermore, configuration management tools are available to control the version change of code flies and documents. Mechanisms to request and obtain shared resources need to be provided so that developers invite their partner for pair programming.

In the DPP setting, users hope to share intermediate results by passing to one or more users. A shared file repository is provided for users to place and retrieve files. Users can browser files and pars on these files. The shared file repository allows users to organize the files in folders.

Pair programming tool should support text and audio or video-based communication so that the pairs discuss questions and selection of solutions or know the partner's sensibility and intention through these communicating tools.

4.3. Activity Indicator

Users need time to perform a task but only the results are shared among them. In the DPP setting, users need to be aware of other user's activities, which can use a peripheral place. The interface of the DPP also should support the presence of the role state of pairs and the function of role exchange.

4.4. Role Switch and Concurrent Control

When a navigator wants to own the role of driver and write the code the system should support to apply for and release the token. Once there is the occurrence of role exchange, the DPP tool should support the file locking to control the change of the code.

Concurrent operations to shared artifacts can lead to conflicting actions that confuse the interacting users and result in inconsistency on the artifacts, make interaction difficult. By means of a token and only let user holding the token modify or access the shared resources. In DPP setting, the user with the driver role can hold a token and allow modifying the code, and the user with a navigator role only browser the code written by the partner. Role switch can allow them switch the role each other and change the token holder.

4.5. DPP Communication Session

Pair programming process is a negotiation process for programming problems such as design strategy, code specification, and collaborative testing. Its goal is to improve code quality and increase programming efficiency. Hence, distributed pair programming tool should support free and natural problem negotiations with a set of communicative events associated a conversation.

For distributed interaction, communication between pairs poses an important role in DPP. There are all kinds of communication channels, such as text chat, whiteboard, remote selection, and audio or video channel. The text chat is a simplest communication style in which users can send short text messages and distribute these messages at the pair's site. The driver or navigator initiates a conversation at any time aiming at a code segment or a design. The conversation with a title is composed of events. Those events are mutually related to the same conversation with a sequence of occurring of them. Each event is represented a message format organized with complete time, content, sender, receiver, and optional code segment and attached documents. But the disadvantage of textual chat communication for a DPP is that the driver needs his or her hands to produce code. Normally, coding and talking goes hand in hand. Thus, the textual chat will not be the most important communication medium [4]. Whiteboard chat is similar to textual chat, but the only difference is that whiteboard uses graphical object to support their interaction. Whiteboard is usually used to discuss design problems of software. For example, pairs in DPP use UML (Unified Modeling Language) to finish the design and analysis of the software.

As an alternative or addition to the communication functionality an audio or video channel can be embedded in the DPP. An audio or video channel supports parallel communication and coding. But the disadvantages of these channels are that they will consume too much network bandwidth, not be stable enough, and establishing connections will not be quick and easy.

Remote selection shows remote user's selection to a local user. Make sure that other pair is aware of his or her partner who has selected the object or edited the code.

5. Design of DPP System

The goal of distributed pair programming system between heterogeneous code editors or developing environments is to enhance pair programming ability and cooperation capacity among partners. As a result of programmers daily use different kinds of single-user code editors or developing environments during their designing or programming task, the functions of existing distributed pair programming system is inferior to the one of the commercial or open-source single-user code editors or developing environments.

Figure 2 shows a framework of distributed pair programming system with the same or different code editors or environments, compatible to the specific program language, between driver and navigator. The system is implemented by the client/server architecture. The communication management module is responsible for transferring of operation information and event or message between the driver and navigator.

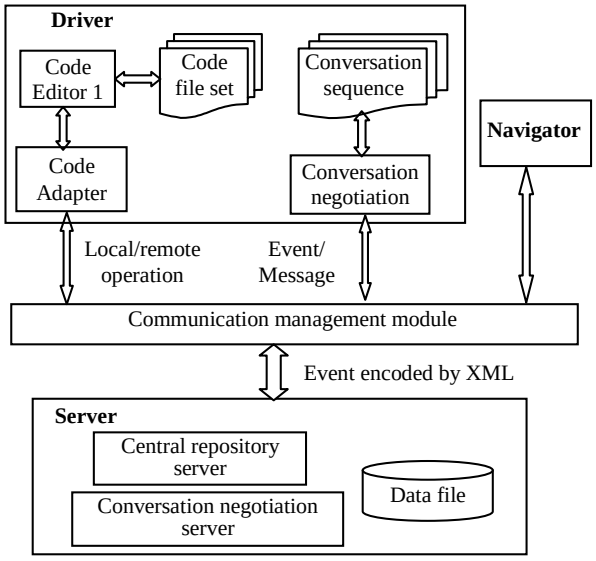

Figure 2. A framework of DPP system.

5.1. Collaborative Editing Subsystem

Moreover, the existing code editors or environments lack good compatibility with the commercial single-user systems, and its usability is poor. It is impossible for programmers to accept these systems to support their development task concurrently. In order to solve this problem, the collaboration transparency technology emerges as the times require. Collaboration transparency technology causes group of users to be possible of no revision to the single-user code editors or developing environments, allows them directly to use familiar single-user code editors or developing environments for distributed pair programming tasks, thus the research of collaboration transparency technology has a vital value.

In **Figure 2**, the driver can select any code editors or developing environments that support a specific program language. The Code Adapter component can capture any local operations from the driver, filter any inessential information, and recombine into useful operation information in a common or standard format. Similarly, the navigator can select the same or different editor with the driver. This is due to the like or experiments of the navigator. The Code Adapter component also transforms the operation information received by Information transfer component into suitable format according to the requirements of local editor, and executes it to the local editor.

In server site, the Central repository server is a resource repository. These resources include source code files, design documents, users, pair information. The design of Central repository operates on the client/server architecture. The clients reveal these resources, and the server is responsible for updating of them. In a one-to-many pair mode, the core programmer needs to know new changes to the code when he/she switches to previous partner.

5.2. Conversation Negotiation Subsystem

The conversation for negotiation subsystem is responsible for the initiating, maintenance, organization, and storage of conversation. The messages of conversation are transferred by specific format between the driver and navigator. Its role is to aid pairs to communication and negotiation for some coding or design problem. Each conversation corresponds to a commitment.

In DPP procedure, there are a lot of conversations occurring between the driver and navigator. For example, the driver may request a help for some materials with the code from the navigator, or hope to discuss some uncertain programming problems. In some time the driver may request the navigator to test the code written by him/her. On the other hand, the navigator may point out the existing problems by conversation negotiations during reviewing the code.

Each conversation is associated a sequence of message which is composed of title, time, source, destination, content, attached documents, associated code segments. **Figure 3** shows the model of conversation for the DPP process. The conversation negotiation server is responsible for recording all conversation information between the driver and the navigator so as to querying and indexing for later usage.

5.3. A DPP System Prototype

We have implemented a preliminary prototype system which adopts the client/server architecture. The system

can provide the basic functions of distributed pair programming in plug-in form integrated MS Visual Sudio environment. The XPPlugin based on client/server architecture consists of three subsystems: real-time communication, code synchronization and pairing management. In client site, communication and code synchronization are implemented in MS Visual studio plug-in style. In server site, all management tasks of DPP are finished by a solo program. The network communication between client and server is implemented by a XMPP (Extensible Messaging and Presence Protocol) which is an open instant-massage protocol based on XML (Extensible Markup Language). This prototype uses open source software, agsXMPP, under .net environment to support the interaction between pairs.

The client program exists in plug-in form which conforms to the specification of MS Visual Studio plug-in. **Figure 4** shows the window of our prototype system. The window consists of three sub-windows: code sharing and editing window, communication window and role switching and control window.

The system architecture, as showed in **Figure 5**, is divided into four layers:

- User interface layer provides the functionalities such as login in, text chat, code control and role switch. User interface is implemented by using LoginForm, chatControl, CodeMonitorcontrol class.
- Middle layer is decided by MS Visual Studio. Only using this layer, the XPplugin can support the tool window pane as a visual studio standard tool pane to be used freely. The goal of design is that DPP tools are allowed to be embedded in visual studio environment, thus increase the efficiency of the prototype system.
- Interaction layer implements interaction between the XPPlugin and internal data of visual studio, including XPPluginPachage and SccService class. XPPluginPackage inherits Package class to allow the whole program as a plug-in to be loaded into Visual studio environment. SccService implements the management of code encapsulated as a service which can be freely called by either internal of the program or other plug-in or programs of Visual studio.
- Network interface layer encapsulates a network communication class using a XMPP protocol to implement the interaction between client and server. Datahandler is an instance of such network communication class.

6. Conclusions

In this paper, we have reviewed the features of the exist-

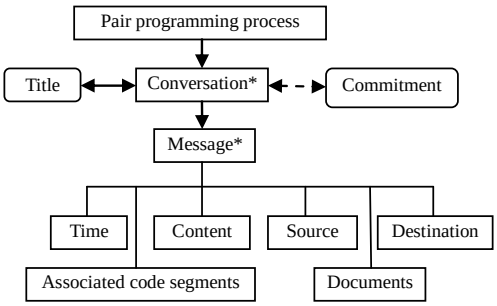

Figure 3. Model of conversation for the DPP process.

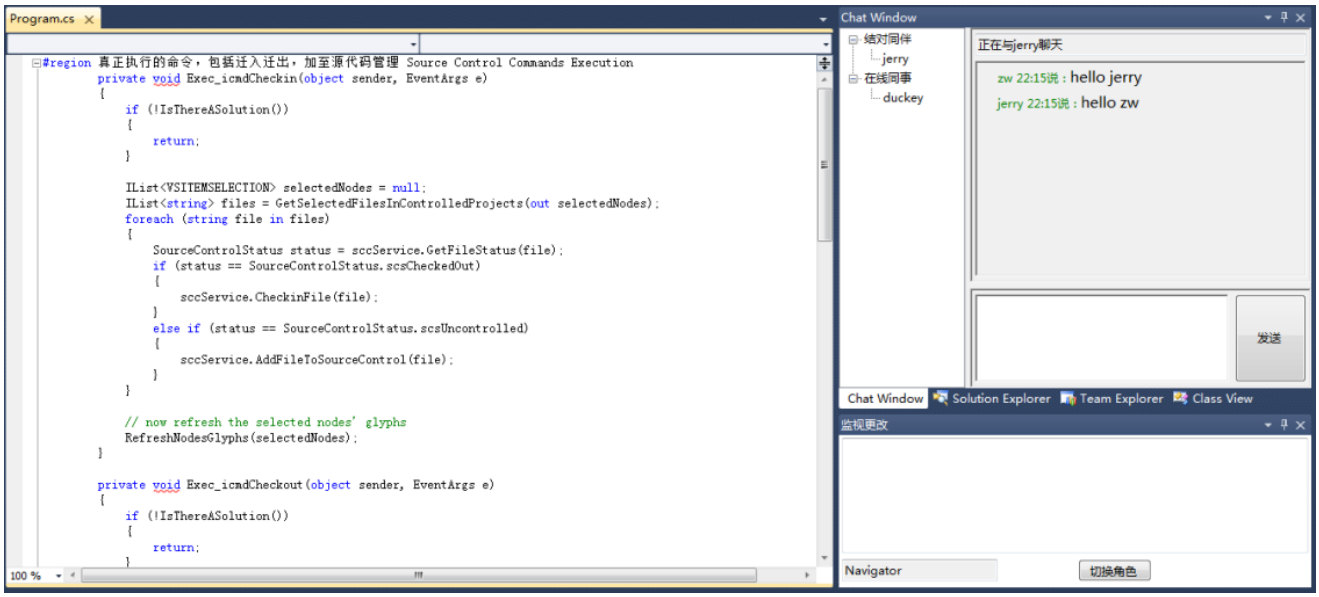

Figure 4. Main window of prototype system.

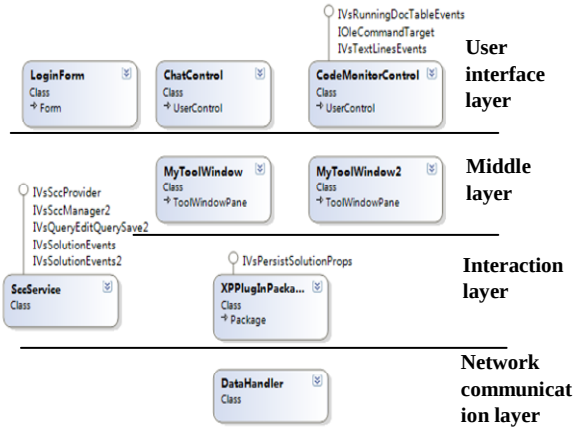

Figure 5. Layered structure of prototype system.

ing distributed pair programming tools, analyzed their advantages and disadvantages. An activity theory is introduced to analyze the process model of DPP and the related main subsystem. A conversation model with commitments is presented based on language/action perspective as a framework for understanding communication within DPP processes. We have analyzed the requirements of distributed pair programming system, presented four important aspects in designing distributed pair programming system: 1) interoperation between heterogeneous editors corresponding to the same language; 2) file sharing at the repository and awareness of pair programming information; 3) role switch and control, and 4) conversation pattern with negotiation. Finally, we have presented a framework supporting distributed pair programming with heterogeneous editors or developing environments. In the future, we will improve our current preliminary system with new collaborative tools to support communication with audio and video channels. We also hope to integrate the existing developing environment, such as J++, JBuilder, Visual Cafe, which is relative to Java language, into our system.

7. References

[1] R. Duque and C. Bravo, "Analyzing Work Productivity and Program Quality in Collaborative Programming," *The* 3*rd International Conference on Software Engineering Advances*, Sliema, 2008, pp. 270-276.

[2] D. Preston, "Using Collaborative Learning Research to Enhance Pair Programming Pedagogy," *ACM SIGITE Newsletter*, Vol. 3, No. 1, January 2006, pp. 16-21.

[3] L. Williams, R. Kessler, W. Cunningham and R. Jefferies, "Strengthening the Case for Pair Programming," *IEEE Software*, Vol. 17, No. 11, 2000, pp. 19-21.

[4] T. Schummer and S. Lukosch, "Understanding Tools and Practices for Distributed Pair Programming," *Journal of Universal Computer Sciences*, Vol. 15, No. 16, 2009, pp. 3101-3125.

[5] M. M. Muller, "Two Controlled Experiments Concerning the Comparison of Pair Programming to Peer Review," *Journal of Systems and Software*, Vol. 78, No. 2, 2005, pp. 166-179.

[6] L. Williams, R. R. Kessler, W. Cuningham and R. Jeffries, "Strengthening the Case for Pair Programming," *IEEE Software*, Vol. 17, No. 4, 2000, pp. 19-25.

[7] J. Ncwroclci and A. Wojciechowski, "Experimental Evaluation of Pair Programming," *Proceedings of European Software Control and Metrics Conference*, London, 2001.

[8] E. Arishoim, H. Gallis, T. Dyba and D. I. K. Sjoberg, "Evaluating Pair Programming with Respect to System Complexity and Programmer Expertise," *IEEE Transactions on Software Engineering*, Vol. 33, No. 2, 2007, pp. 65-86.

[9] H. Gallis, E. Arishoim and T. Dyba, "An Initial Framework for Research on Pair Programming," *Proceedings of the* 2003 *ACM-IEEE International Symposium on Empirical Software Engineering*, Toronto, 2003, pp. 132-142.

[10] C. McDowell, L. Werner, H. Bullock and J. Fernald, "The Effects of Pair Programming on Performance in an Introductory Programming Course," *Proceedings of the* 33*rd Technical Symposium on Computer Science Education*, Cincinnati, 2002, pp. 38-42.

[11] N. Nagappan, L. Williams, *et al.*, "Improving the CS1 Experience with Pair Programming," *Proceedings of the* 34*rd Technical Symposium on Computer Science Education*, Reno, 2003, pp. 359-362.

[12] L. Williams and R. L. Upchurch, "In Support of Student Pair Programming," *Proceedings of the* 32*nd Technical Symposium on Computer Science Education*, Charlotte, 2001, pp. 327-331.

[13] P. Baheti, E. Gehringer and D. Stotts, "Exploring the Efficacy of Distributed Pair Programming," *Proceedings of XP Universe*, Springer-Verlag, Chicago, 2002, pp. 208-220.

[14] L. Williams, D. M. Scott, L. Layman and K. Hussein, "Eleven Guidelines for Implementing Pair Programming in the Classroom," *Agile* 2008 *Conference*, Kopaonik, 2008, pp. 445-451.

[15] L. Werner, B. Hanks and C. McDowell, "Pair Programming Helps Female Computer Science Students," *ACM Journal of Education Resources in Computing*, Vol. 4, No. 1, 2004, pp. 1-8.

[16] H. Natsu, J. Favela, *et al*., "Distributed Pair Programming on the Web," *Proceedings of the* 4*th Mexican International Conference on Computer Science*, Los Alamitos, 2003, pp. 81-88.

[17] B. Hanks, "Tool Support for Distributed Pair Programming: An Empirical Study," *Proceedings of Conference Extreme Programming and Agile Methods*, Calgary, 2004, pp. 1-18.

[18] S. Hupfer, L. T. Cheng, S. Ross and J. Patterson, "Introduction Collaboration into an Application Development Environment," *Proceedings of the Computer Supported Cooperative Work*, ACM Press, New York, 2004, pp. 21-24.

[19] F. Maurer, "Supporting Distributed Extreme Programming," *Proceedings of Conference on Extreme Programming and Agile Methods*, Springer Verlag, London, 2002, pp. 13-22.

[20] T. Schummer and J. Schummer, "Support for Distributed Teams in Extreme Programming," In: G. Succi and M. Marchesi, Eds., *Extreme Programming Examined*, Addison Wesley, Boston, 2001, pp. 355-377.

[21] M. Reeves and J. Zhu, "Moomba: A Collaborative Environment for Supported Extreme Programming in Global Software Development," In: *Lecture Notes in Computer Science*: *Extreme Programming and Agile Process in Software Engineering*, Springer, London, 2004, pp. 38-50.

[22] C. W. Ho, S. Raha, E. Gehringer and L. William, "Sangam: A Distributed Pair Programming Plug-in for Eclipse," *Proceedings of the* 2004 *OOPSLA Workshop on Eclipse Technology Exchange*, New York, 2004, pp. 73-77.

[23] K. Elizabeth, A. Dwight, *et al*., "A Development Environment for Distributed Synchronous Collaborative Programming," *Proceedings of the* 13*th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, Madrid, 2008, pp. 158-162.

[24] R. Duque and C. Bravo, "Analyzing Work Productivity and Program Quality in Collaborative Programming," *The* 3*rd International Conference on Software Engineering Advances*, 2008, pp. 270-276.

[25] K. Kuutti, "The Concept of Activity as a Basic Unit for CSCW Research," In: L. J. Bannon, M. Robinson and K. Schmid, Eds., *Proceedings of the* 2*nd ECSCW*, Kluwer Academical, Amsterdam, 1991, pp. 249-264.

[26] K. Kuutti, "Identifying Potential CSCW Applications by Means of Activity Theory Concepts: A Case Example," *Proceedings of CSCW*, ACM Press, New York, 1992, pp. 233-240.

[27] T. Winograd, "A Language/Action Perspective on the Design of Cooperative Work," *Human Computer and Interaction*, Vol. 3, No. 30, 1998, pp. 203-220.

[28] G. D. Michelis and M. A. Grasso, "Situation Conversations within the Language/Action Perspective: The Milan Conversation Model," *Proceedings of the* 5*th Conference on CSCW*, Chapel hill, North Carolina, 1994, pp. 1-12.

[29] P. Baheti, L. Williams, *et al*., "Exploring Pair Programming in Distributed Object-Oriented Team Projects," *Proceedings of OOPSLA Educators Symposium*, Seattle, 2002, pp. 1-6.

