A Dual-mode Exerciser for a Collaborative Computing Environment

Chien-Ming Wang, Shyh-Fong Hong, Shun-Te Wang, Hsi-Min Chen
Institute of Information Science Academia Sinica
NanKang, Taipei, Taiwan
Email:{cmwang, fong, sdwang,seeme}@iis.sinica.edu.tw

Abstract

Computer-supported cooperative work (CSCW) supports groups with communication and coordination during the execution of their activities. It allows physically dispersed teams to engage in a common task by providing an interface to a shared workspace. A variety of synchronous applications are playing a major role in distance education, joint program development, cooperative publishing, etc. As these applications are usually platform-dependent, groupware programmers have to develop new applications for each groupware platform. In this paper, we present a collaborative application developing environment called CollabRunJava, which allows the groupware developers to concentrate only on application-specific details. CollabRunJava supports 2 modes for developing applications. In the instant-develop mode, user can immediately execute and test the classes, which are also developed by our system. In the dynamic modify-observe mode, user can observe application behavior, debug applications and change the running applications codes without re-execution. Our system can be further enhanced by using plug-in components with the above mechanisms. For example, with the aid of visualization components, our system can help users explore program behavior. This also makes it helpful for distant learning and program testing.

1. Introduction

A collaborative computing environment allows remote users to collaborate to solve problems, share information, develop programs and process distance learning [1][2][5][6]. However, developing synchronous groupware is a difficult and time-consuming tasks [3][4][7]. In addition to application-specific details, shared data have to be organized, communication between different sites has to be managed and user interfaces that can handle events from multiple users have to be developed. To relieve the application

developer from re-designing standard modules, collaborative platforms are widely available with building blocks for standard solutions. They provide a communication infrastructure and a runtime system. Using a collaborative platform, the application developer can concentrate on application-specific details and use collaborative services from a standard library.

Our system, called CollabRunJava, is not only a collaborative platform for developing and executing collaborative applications; it also shares single-user Java applications for synchronous collaborations. Programmers don't need to re-design existing Java applications for new groupware applications. We also provide a replicated resource sharing mechanism to maintain the developed project resource consistent.

In a conventional program developing environment, testing, design time and execution time are exclusive. Programmers always get trapped in a cycle of coding, debugging, application execution, error occurrence, and re-debugging. This process is inconvenient and time consuming. In addition, the setup of the libraries and components required to deploy and execute an application is a complex operation. Another disadvantage is that a conventional program cannot keep existing states and continue execution after a modification of the program is made. Consequently, an application's states, such as variable values and loaded classes, always need to be reset when the program is modified and re-executed.

To deal with these problems, CollabRunJava not only keeps the features of a conventional developing environment but also provides a collaborative dual-mode exerciser. In the instant-develop mode, users can develop applications collaboratively and rapidly within an interactive programming environment. The developed applications can be tested and executed directly in this mode. In the dynamic modify-observe mode, we can change the codes of executing application without re-executing it. With the above features, our system is a good program development and testing environment, it is also a good E-learning platform in collaborative environment.

2. The ShareTone Collaborative Platform

CollabRunJava was developed as a collaborative application on a CSCW project, called ShareTone [19]. This is a Java-based collaborative computing platform that facilitates information exchange and collaboration among participants over the Internet. Its main objective is to build a collaboratory that enables remote users with expertise in specific areas of a scientific field to collaborate with one another. Information that is pertinent to each user's specialty can be shared in order to solve a particular problem.

ShareTone also facilitates the remote execution of software objects for on-line demonstration and distance learning. Figure 1 shows the three layers of the system:

- The application layer contains collaborative applications provided by the system, such as CollabRunJava, or those developed by users. To achieve collaboration, each application uses a collaboration toolkit, called Collabench, to share the application states among users.
- The communication layer is responsible for collaborative communication, including security management and the collaborative message service. CollabMS is a Java messaging API for collaborative and peer-to-peer communication. Collabench is built on the top of CollabMS and enhances the automation and customization of collaborative applications. These two components enable CollabRunJava to work as a collaborative computing environment.
- The service layer provides the necessary services during collaboration, such as registry service, collaborative persistent service (CollabPS), and collaborative file service (CollabFS). CollabFS is a service based on CollabMS for collaborative file sharing. By using this service, all files can be stored on the collaborative file server. Any file update events are broadcast synchronously to all clients in the same working group, and all clients retrieve the updated files from the server to the local disk automatically.

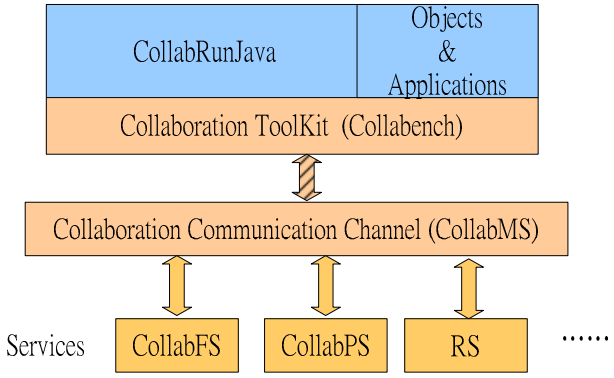

Figure 1: The ShareTone collaboration platform

3. System Design

Our system provides a collaborative development tool and runtime platform. Users can develop applications standalone or collaboratively. All basic developing tools, such as editor, debugger and compiler, are provided. A Java interpreter is also provided, so

users can immediately test and operate the Java classes they developed. This feature makes the program design more interactive.

3.1. Dual-Mode Exerciser

CollabRunJava has two modes: The instant-develop mode and the dynamic modify-observe mode. In the instant-develop mode, users can develop applications rapidly with useful tools. User can also operate objects directly through an embedded interpreter. The dynamic modify-observe mode is used when applications that are currently being designed or executed, need to be debugged or modified on-the-fly. Tracing or observing the status of process execution can also be done in this mode.

As shown in Figure 2, all the activities in both modes are collaborative through the collaboration communication channel. Users in the same working group can join the processes of application development and share the same class and object space.

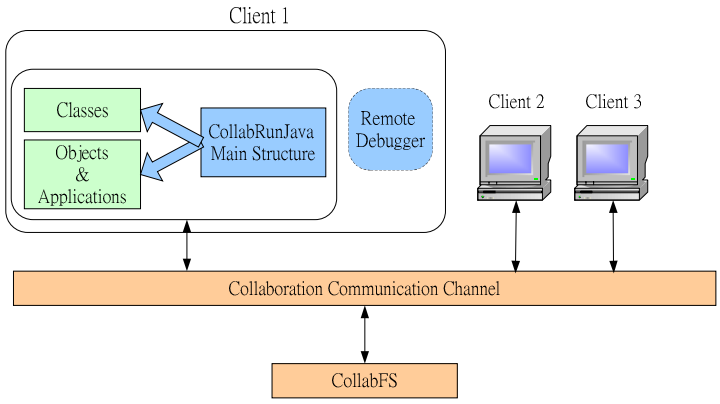

Figure 2: A dual mode exerciser on a collaborative computing environment

The Instant-Develop Mode

After a user logs in to a collaborative working group, CollabRunJava is ready and its basic mode, instant-develop mode, starts to operate. In this mode, an interpreter, a compiler and a project editor compose CollabRunJava. Programmers can use CollabRunJava to write source codes, compile these codes into Java class and perform other developing activities. Through the collaboration package, all activities, including file creation and update, Java interpreting etc. will be synchronous among all clients in the same working group.

The Java language interpreter, called RunJava, is responsible for Java interpreting and class loading. Like the Java class loading mechanism, RunJava only loads the basic classes when it is initialized. It will only load the required classes in the class path, or developed by the users on-line, to save the memory space. RunJava supports dynamic class loading, thus it provides more flexibility to maintain class library at runtime. It's easy to check or operate the classes immediately after development by interpret testing codes in RunJava. If problems occur in these classes, programmer can switch to the dynamic modify-observe mode to debug, modify and re-compile these classes without re-execute the applications. This improves the performance of developing applications.

Since CollabRunJava is a collaborative environment, all operations are synchronous, including editing, code interpretation, and file access. The interpretation and edit operations will be sent to the collaboration server first, and broadcast to all clients. Each client will process these operations after receiving them from the collaboration server.

To keep all project resources, such as source files and libraries, consistent among all users, we use CollabFS to maintain the above resources. When a project is created, all necessary resources will be uploaded to the file server by the creator. The other participants will download these resources at the beginning when they join the project. When the resources are modified, the collaboration server will be modified and update the corresponding files. Once the server updates the project resources, it will notify other clients to update their files from CollabFS. For the less limitation, the locking mechanism are not used to the project resources, all users can access and modify a file at the same time without limitation. The file will be overridden when more than one user save it. If it happens, user will receive a notification and can choose one of the following actions: (1) Save their current file as a new file and update the latest version from the CollabFS server. (2) Update the latest version without save. (3) Ignore the notification.

The Dynamic Modify-Observe Mode

In a conventional developing environment, the execution of applications must be terminated in order to modify the application codes. It's also necessary to switch from the execution mode to the debugging mode to observe the process of application execution, and observe or set the values of variables declared in applications. In addition, the program status must to be reset when switching between the execution mode and the debugging mode.

The dynamic modify-observe mode overcomes the above disadvantage by utilizing a remote debugger. With the remote debugging connection to the running applications,

which is executed through CollabRunJava, the programmer can debug the applications and observe the program behavior. By using the “hot swap” feature [21] of the remote debugger, the application codes can be changed without re-execute it.

With the above features CollabRunJava provides a unique function called "annotation code insertion". Annotation codes are temporary codes that can be inserted into source codes to enhance the original program, without change the original source codes. After inserting the annotation codes, CollabRunJava recompiles the source codes, and reload the modified classes automatically. This feature is very useful when it is necessary to do extra work without changing the source codes, such as geometric algorithm visualization or e-learning. Geometric algorithm visualization will be addressed later in this paper.

Some limitations should be noted: (1) the modification of methods have active stack frames, those active frames continue to run the byte codes of the previous method. The redefined method will be used on new invocations only. (2) The redefined class function does not cause any initialization, In other words, redefining a class does not cause its initializers to be run. The values of preexisting static variables will remain as they were prior to the call. However, completely uninitialized (new) static variables will be assigned their default value. (3) Not all target virtual machines support this operation, we suggest to use Sun Java Virtual Machine 1.4.1 version or above.

3.2. The Architecture of CollabRunJava

Figure 3 shows the architecture of CollabRunJava. As shown in the figure, CollabRunJava is composed of several plug-ins. For reducing code duplication and makes components easier to maintain, all plug-ins are designed by following the Model-View-Control (MVC) architecture. The user interfaces of the plug-ins are replaceable. Developers can, therefore, design a graphic user interface specific to their needs. We provide a simple collaborative project editor. It allows users to edit source files, which are set share-enabled, collaboratively. The compiler is used to compile source codes and produce the Java classes. Currently, it uses the Sun Java compiler 1.4.1 version.

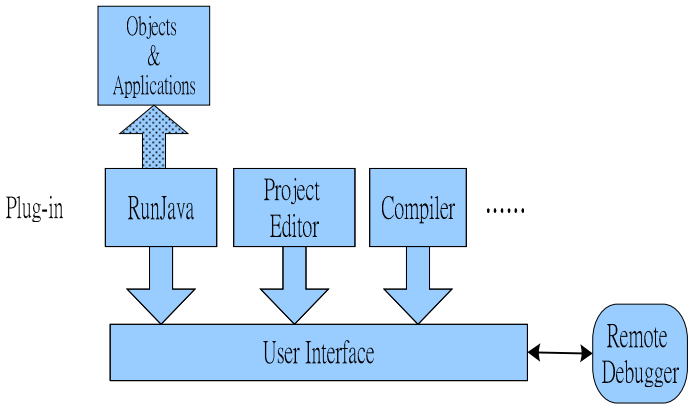

Figure 3: The architecture of CollabRunJava

There are 2 major components in our system. The first one is RunJava, an interpreter of Java programming language. It is the core of this system. Because RunJava is embedded, the compiled classes and objects written in java language can be operated easily and can also obtain an immediately response. It can also achieve the goal of interactive programming and reduce the difficulties of system developing. In RunJava, several useful functions are supported, including a class browser, an interpretation scripts recorder, dynamic class loading and on-line class path setup. The class browser will display all the classes loaded into the memory, together with their details such as the methods and fields of each class. The instance list in class browser shows the object instances that are declared as the selected class. When one instance on the list is selected, the values of the fields belonging to this instance will be shown. The interpretation scripts recorder will record (capture) the interpretation scripts when it is enabled. These scripts can be saved as files and retrieved for interpretation via the replay function. Class loading is an annoying problem for many new programmers using Java. RunJava can load the libraries dynamically and set the class path at runtime. This eases the complexity of class path setup.

Another major component is the enhanced remote debugger. We use JSwat [20] as our remote debugger. By using the enhanced remote debugger, CollabRunJava can replace the classes at runtime, and suspend a running application dynamically. It can also add extended codes to the running application without modifying the source codes, and observe and change the status of the running application. The remote debugger provides greater flexibility in the development of applications and simplifies the debugging procedure.

To extend the functionality of CollabRunJava, software components written in the specification of JavaBeans, can be plugged into the system. For example, a collaborative geometric drawing bean, called GeoDrawing is plugged in by default. It can read the geometric data and visualize the outcome of geometric algorithms. In the current version, the plug-in components must be embedded by the ShareTone software component upload mechanism before CollabRunJava starts running. A dynamic plug-in mechanism will be developed in the future.

4. Design Choices and Collaboration Issues

While developing CollabRunJava, many design choices and issues had to be investigated. In this section, we describe important design choices and research issues in the design of the collaborative dual-mode exerciser.

4.1. Communication between two Processes

As described above, CollabRunJava has to start two processes to provide functionality to the dynamic modify-observe mode. The main process contains CollabRunJava, user defined classes and objects, applications and collaboration package. It is responsible for code editing, compiling, interpreting, object and class operation and collaboration among all clients. The debugger process is slimmer, as it only contains a remote debugger, which makes a remote debugging connection to the main process when the dynamic modify-observe mode is activated.

The first issue is how to arrange an interface with all the components of CollabRunJava. There are five elements in the system: CollabRunJava, remote debugger, UI of CollabRunJava, UI of the remote debugger, applications/objects initialized by RunJava. The following are possible choices of system architecture.

Compose all elements in one process, each process has its own user interface or Integrate 2 user interface in the debugger process. In the First case, the whole system will be suspended by itself. The second case is inconvenient and non-user-friendly. In the third case, we need to put collaboration service on debugger process and main process, it makes a large amount of collaboration and interpretation communications. Finally, we choose the last case, which integrates two user interfaces in the main process. The most common user activities are editing, file accessing, compiling and interpretation. A lot of communication between two processes can be eliminated, if the integrated user interface

is in main process. Also, only one collaboration service is needed in this case. CollabRunJava can share this service with the applications and objects initialized by RunJava, as shown in Figure 4.

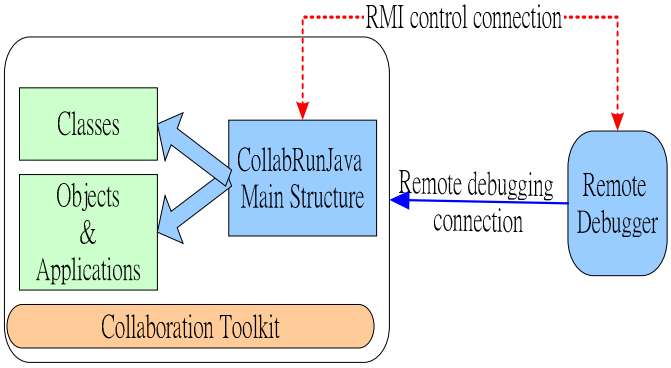

Figure 4: Communication between Two processes

In the original design, a socket was used as the communication mechanism, and a debugger controlling protocol was designed to allow the user to control the remote debugger from the main process. However, the debugger controlling protocol was becoming too complex, especially when more functions were added in CollabRunJava. The one way communication mechanism was not sufficient. We therefore choose Java RMI mechanism as the new communication mechanism. By using RMI, we can easily invoke remote object methods instead of designing complex communication protocol. It is also convenient to make multiple connections and achieve the goal of duplex communication.

4.2. The Suspend Policy

In CollabRunJava, the whole developing environment and the executing applications are in the same process. However, if users want to suspend the executing applications to observe or modify them, the remote debugger will not only suspend the applications, but also the entire process. Consequently, the entire CollabRunJava will be halt and no more operation can be performed.

To avoid this problem, RunJava is started as a sub thread of CollabRunJava, so all applications initialized by RunJava are also execute as a sub thread. We also modified the

suspend policy of the remote debugger from SUSPEND_ALL to SUSPEND_EVENT_THREAD. With these changes, the remote debugger will only suspend the specified application, not the whole system.

4.3. The Issues of Late Comers

The collaboration system, must not only keep all online users to acting synchronously, it must also need to let the late-comers with the same status as the existing users. Although the collaboration service of ShareTone achieves the former, the late-comer issue is more complex.

In CollabRunJava, for two users have the same status, the RunJava context must have the same value. The RunJava context has two major elements: the class table and the variable table. The class table records the class name loaded into memory and the corresponding class instance. The variable table records the objects and variables initialized, or declared in RunJava. There are two main mechanisms that provide the later-comer with the same RunJava context as the existing users. They are the replay mechanism and the serialization mechanism.

CollabRunJava will save the interactive operation and interpretation commands in the history command table. When a new user joins a working group, the collaboration service will sends the history command table to the new user. After interpreting all commands in the history command table, this new user continues to request and process new operations, which are created during the late comer restoration, from the collaboration server. When the operation queue in the collaboration server is empty, the new user begins to listen the broadcast events from the collaboration server, and act the same operations with the other users. Thus all users have the same RunJava context and are playing the replicated events in the same time.

But the above “replay” mechanism will take a long time to restore the current state if the amount of history command is large. Our system also provides serialization mechanism to over comer the problem. CollabRunJava will serialize the class table and variable table and send them to the collaboration service when synchronous information request is received. After the new user de-serializes the tables, the RunJava context is the same as the other users. But this mechanism has some limitations as Java Serialization mechanism [18] does. Some classes or objects, such as Thread class, cannot be serialized correctly.

As both synchronous mechanisms are provided by CollabRunJava, users can decide manually which one they prefer. We hope to find better and automatic solutions in the future.

4.4. Multi-user Control of the Debugger

Although CollabRunJava is a collaborative developing environment and users can develop and execute applications collaboratively, CollabRunJava doesn't support the collaboration mechanism for all functions in dynamic modify-observe mode. When one user switches to this mode, the remote debugger in this client will be notified and start to process debugging. Because the memory references of clients are different, it will confuse other users when the debugging operates individually on each client. Therefore, we adopt the centralized sharing approach here and use the virtual collaboration mechanism to simulate collaborative debugging in the dynamic modify-observe mode. Only one remote debugger will be notified to make a remote debugging connection to main process in this client. This client is called debugging executor.

In the collaboration communication mechanism, the user who created, or joined the working group first, will be assigned to provide the synchronous information to the late-comer. When this user leaves this group, the collaboration mechanism will randomly assign another user to provide the synchronous information. CollabRunJava also uses this mechanism to assign the debugging executor.

Because the main process contains the collaboration communication mechanism, operations on the main process of all users are still synchronous. All users can operate the remote debugger of the debugging executor through the collaboration communication mechanism, but the context of variables and memory reference etc. are the values on the debugging executor. In other words, only debugging executor is debugged. The application on the other clients is suspended until the debugging process is finished. But the other clients can observe and operate the debugging process on the debugging executor.

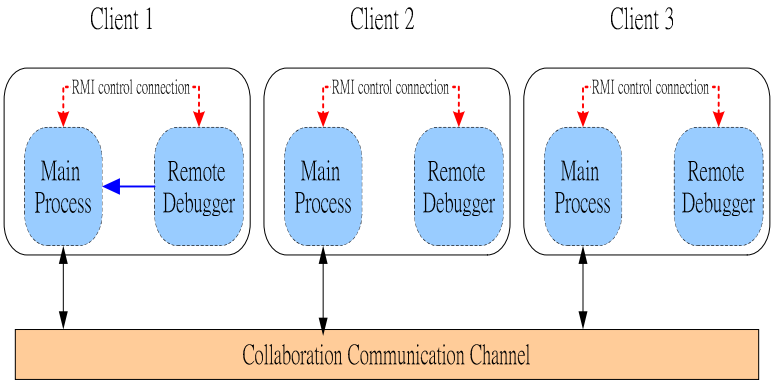

Figure 5: Multi-user Control of Remote Debugger

4.5. The Asynchronous Interpretation Problem

Since only one remote debugger will connect to the main process in the same client, all debugging actions only work at this client. Consequently, if users want to suspend an application by setting up break points, only the application executed on the client of the debugging executor will be suspend. As the application executed on other clients won't be suspended, application activities among users in the working group will be asynchronous.

To avoid this problem, CollabRunJava keeps a suspended class table to record classes which contain active break points. It also checks the interpretation commands and the commands sent by the collaboration communication channel. If these commands relate to the classes recorded in the suspended class table, they will be saved in a command queue. The commands in the queue will be processed when the related classes continue to execute. In the current situation, we can only examine whether the commands relate to the classes in only one level deep. Related commands cannot always be intercepted correctly. This is a major issue to be solved in the future.

4.6. Visualization Component Interface

We provide a GeoDrawing bean by default plug-in component to drawing the geometric algorithms and let users to dynamic modify, debug algorithms as section 3 described. In order to develop customize visualization components with the above functionalities, the visualization components must implement an interface, called VisualRuntimeInterface, thus these beans can interact with CollabRunJava. The interface defines the behavior of insert/delete annotation codes, insert/delete monitor variables, and add variable monitor listeners. The more details will be described in next section.

```
VisualBeanRuntimeInterface{
  //methods of annotation codes
getAnnotateCodeTable();
  setAnnotateCodeTable(Hashtable codeTable);
  addAnnotateCodes(String className,String [] codes,int line);
  deleteAnnotateCodes(String className, int startLine, int endline)
  modifyAnnotateCodes(String className,String [] codes,int line);
//methods of variable monitoring
getMonitorVariable();
  setMonitorVariableTable(Hashtable monitorTable);
  addMonitorVariable(String className, String varName,int line);
  deleteMonitorVariable(String className, String varName,int line);
  addVariableListener(InspectorListener listener);
  startMonitor();
}
```

5. Case Study – CollabRunJava and GeoDrawing

In computational geometry, the visualization and animation tool is very important to understand, present and debug algorithms [8] [9]. Many systems have tried to satisfy the needs of computational geometry [10][11][12][13]. Different approaches are used in algorithmic visualization. An important factor is how the visualization code connects with the algorithmic implementation to create the visualization and animation [14].

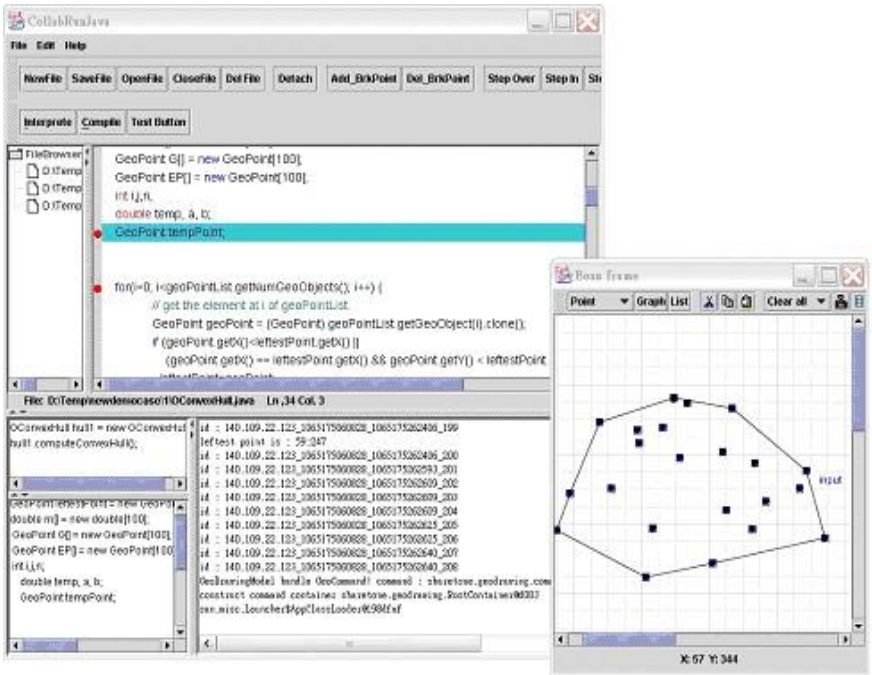

Figure 6: A snapshot of CollabRunJava and GeoDrawing

GeoDrawing, a collaborative geometric drawing JavaBeans which is a component of the ShareTone project, can read the geometric data and visualize the geometric algorithm. Combined with CollabRunJava, our system helps users to dynamically annotate their visualization code in the algorithmic implementation in the running time. Therefore, the geometric algorithm is independent of the visualization codes. Figure 6 is the snapshot of GeoDrawing embedded in CollabRunJava.

Unlike most visualization tools, our system not only display the final state of geometric graph, the intermediate steps of drawing the algorithmic visualization can also be observed through setting break points, or annotating pause codes to the visualization codes. This feature is called incremental visualization. For example, Figure 7 shows 4 steps while drawing the convex hull. Each step is suspended by the breakpoint and the annotated visualization codes update the display.

The geometric visualization may need more interaction with the users [15]. By using the annotation codes and interpretation, users can modify some geometric data interactively and see the result produced by the algorithm immediately. This feature is called interactive visualization. In figure 8, (a) a convex hull was drawn and then (b) we add a new point outside the convex hull. (c) After an interpretation of "re-compute" command, the new convex hull is drawn. Of course, the bean also has collaborative capability, which enables users in different regions to view and manipulate the geometric data collaboratively.

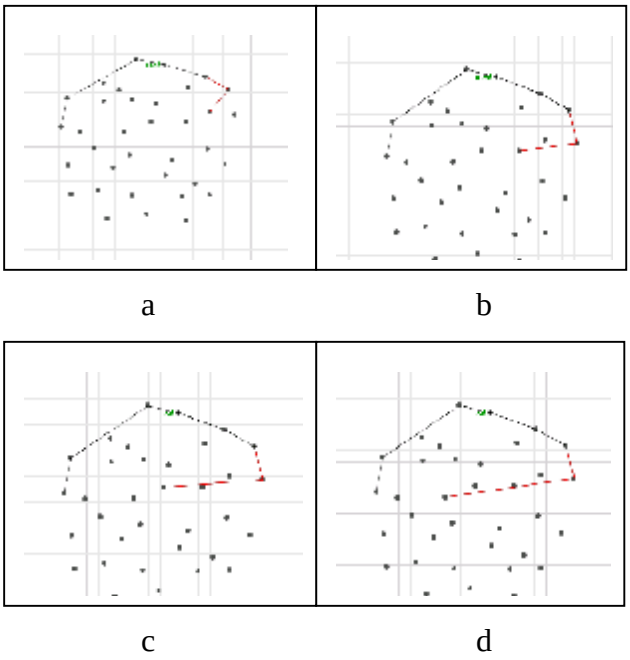

Figure 7: Incremental Visualization: Display the intermediate result of drawing convex hull step by step

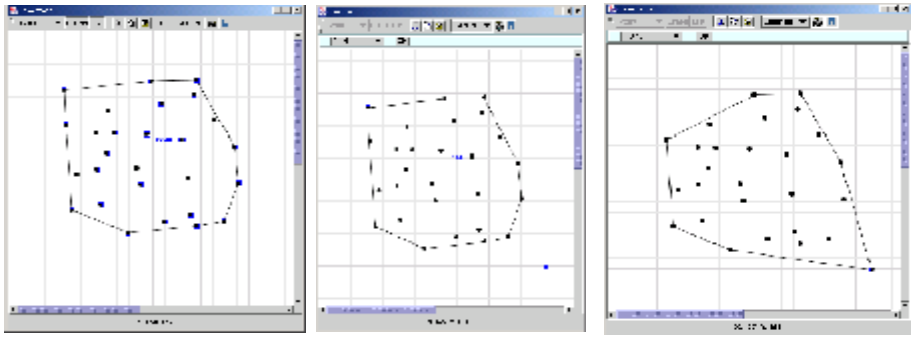

Figure 8: Interactive Visualization: Add a new point on and then display the result immediately.

6. Related Works

Several other software systems and toolkits that enable or support collaboration have been developed, or are evolving. For example, TeamWave is a groupware system that combines synchronous groupware technologies, such as shared white-boards, chat rooms, and customizable groupware applets, with a persistent work environment. However, TeamWave is a relatively closed system that is based on the GroupKit groupware toolkit [11], and only subsystems and applications developed specifically with GroupKit can be used with TeamWave.

Most development systems only focus on the single-user environment, and only a few of these systems have a facility for interactive programming. For example, WingIDE[12] is a development environment for Python language. Like our system, it provides a graphic debugger, a code browser, and a source editor. However, the runtime and design-time are separated, and the user needs to re-execute the applications after re-designing them. The system lacks interactive programming ability. DrJava [13] is a lightweight development environment for writing Java programs. It provides basic features, including an editor, an interpreter and a debugger, for program design.

The above systems only develop or debug standalone programs. In contrast, our system provides an open collaborative development environment that helps users work collaboratively and develop collaborative products easily. It is also a collaborative application platform and all applications that follow the ShareTone collaboration protocol can be executed by it. Finally, because of an embedded, interpreter, users can test and operate the classes and applications immediately when they develop applications with our system.

7. Conclusion and Future Works

The dual-mode exerciser for a collaborative computing environment facilitates the efficient development of applications collaboratively and share the single-user Java applications for synchronous. With its interactive programming ability, it helps users refine the programs, test classes, find algorithm bugs, and observe the process of program execution more easily, as the example in the case study shows. With the replicated resource sharing mechanism, implemented by CollabFS, we can maintain the consistent of project resources.

Our work on CollabRunJava continues in several directions with multiple collaboration teams. Ongoing research includes automating the choice policy between the replay and the serialization mechanism for late-comers, and refining the concurrency-control algorithms in asynchronous interpretation problems. Further plug-in components of CollabRunJava will be developed to support more functions. For example, the component, which is called Inspector, allows users to monitor class fields and even local variables. Whenever these fields and variables are accessed or modified, the events and specific instances will be fired to the corresponding objects. This function is very useful on dynamic evaluation, especially on the auto-visualization for the geometric algorithm.

Acknowledgements

The authors wish to thank the Web-Based Collaboratory Research Group at IIS with the very useful feedbacks, comments and supports. This work is supported in part by National Science Council under contract Nos. NSC92-2213-E-001-15 and NSC93-2213-E-001-008.

References

[1] C.A. Ellis, S.J. Gibbs, and G.L. Rein, "Groupware: Some Issues and Experiences," Comm. ACM, vol. 34, no. 1, Jan. 1991, pp. 39-58.

[2] T. Rodden, "A Survey of CSCW Systems," Interacting with Computers, vol. 3, no. 3, Dec. 1991, pp. 319-353.

[3] M. Roseman and S. Greenberg, "Building Real-Time Groupware with GroupKit, A Groupware Toolkit," ACM Trans. Computer-Human Interaction, Vol.3 No. 1, Mar. 1996, pp. 66-106

[4] Kaplam S.M., Tolone W.J., Carrol A.M., Bogia D.P., Bignoli C.: "Supporting collaborative software development with conversation builder", Proceedings of *ACM SIGSOFT* '92: Fifth Symposium on Software Development Environments, Washing D.C., pp. 11-20, 1992.

[5] J. Altmann, R. Weinreich, "An Environment for Cooperative Software Development Realization and Implications", Proceedings of the 31st Hawaii International Conference on System Sciences (HICSS-31), Collaboration Systems and Technology, Big Island of Hawaii, USA, January 6 - 9, 1998, IEEE Computer Society Press 1998

[6] Gaeta, M.; Ritrovato, P., "Generalised environment for process management in cooperative software engineering", Computer Software and Applications Conference, 2002. COMPSAC 2002. Proceedings. 26th Annual International , 26-29 Aug. 2002 Pages:1049 – 1053

[7] L. Osterweil "Software Processes are Software too", in Proceedings of the 9th. International Conference on Software Engineering, ACM Press, New York, N.Y., pp.2-13, 1987.

[8] A. Hausner, D. P. Dobkin, "Making Geometry Visible: an Introduction to the Animation of Geometric Algorithms", Computer Science Department, Princeton University, 1996.

[9] Andreas Kerren, John T. Stasko, "Algorithm Animation - Introduction", In Proceedings of Dagstuhl Seminar on SoftwareVisualization, 2001.

[10] A. Tal and D. P. Dobkin, "Visualization of geometric algorithms", IEEE Trans. on Visualization and Computer Graphics, 1 (1995) 194-204.

[11] Alejo Hausner and David P. Dobkin, "Gawain: Visualizing Geometric Algorithms with Web-Based Animation", In Symposium on Computational Geometry, pages 411-412, 1999.

[12] Ch.A. Hipke and S. Schuierer, "VEGA: A user centered approach to the distributed visualization of geometric algorithms", In Proceedings of the 7-th International Conference in Central Europe on Computer Graphics, Visualization and Interactive Digital Media (WSCG'99), pages 110--117, 1999.

[13] D. T. Lee, Chin-Fang Shen, and Dennis S. Sheu, "Geosheet: A Distributed Visualization Tool for Geometric Algorithms", International Journal of Computational Geometry and Applications, pages 119-156, 1998

[14] Camil Demetrescu, Irene Finocchi, John T. Stasko, "Specifying Algorithm Visualizations: Interesting Events or State Mapping?", In Proceedings of Dagstuhl Seminar on SoftwareVisualization, 2001.

[15] Matthias sken and Stefan Naher, "GeoWin A Generic Tool for Interactive Visualization of Geometric Algorithms", In Proceedings of Dagstuhl Seminar on SoftwareVisualization, 2001.

[16] Archaeopteryx Software Inc., http://wingide.com/wingide

[17] Java Programming Languages Team, Rice University, http://drjava.sourceforge.net/

[18] Sun Microsystems, Java Object Serialization Specification, 2003

[19] ShareTone, available at http://www.sharetone.org
[20] JSwat, A graphical Java debugger, http://www.bluemarsh.com/java/jswat/

[21] M. Dmitriev, “Toward Flexible and Safe Technology for Runtime Evolution of Java Language Applications”, Published in the Proceedings of the Workshop on Engineering Complex Object-Oriented Systems for Evolution, in association with OOPSLA 2001 International Conference, Tampa Bay, Florida, USA, October 14-18, 2001.