A Dual-Mode Exerciser for a Collaborative Computing Environment

Chien-Min Wang, Shyh-Fong Hong, Shun-Te Wang, Hsi-Min Chen
Institute of Information Science, Academia Sinica
Taiwan, R.O.C.
{cmwang, fong, wangsd, seeme}@iis.sinica.edu.tw

Abstract

Computer-supported cooperative work (CSCW) supports groups with communication and coordination functionality 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 development environment called CollabRunJava, which allows the groupware developers to concentrate only on application-specific details. CollabRunJava supports two modes for developing applications. In the instant-develop mode, user can immediately execute and test the classes. In the runtime-evolution mode, user can observe an application's behavior, debug applications and change the codes of running applications without re-execution. Our system can be further enhanced using plug-in components with the above mechanisms. For example, with the aid of visualization components, our system can help users explore a program's behavior. In addition, all these activities can be performed in a collaborative way. This makes it also helpful for distant learning and program testing.

Keywords: CSCW, collaborative computing, debugger, Java interpreter, software testing.

1 Introduction

A collaborative computing environment allows remote users to collaborate to solve problems, share information, develop programs and enable distance learning [1][2][5][6]. However, developing synchronous groupware may be a difficult and time-consuming task [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, but it also shares single-user Java applications for synchronous collaborations. Programmers do not 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 development environment, design time, testing 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 development environment but also provides a collaborative dual-mode exerciser. In the instant-develop mode, users can develop collaborative or standalone applications rapidly within an interactive programming environment. The developed applications can be tested and executed directly in this mode. In the runtime-evolution mode, we can change the codes of an executing application without re-executing it. All the developing processes in both modes are synchronous. With the above features, our system is a suitable program development and testing environment. It is also a good e-learning platform in a collaborative environment.

2 The ShareTone Collaborative Platform

CollabRunJava is developed as a collaborative application on a CSCW project, called ShareTone [19]. It 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:

(1) 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.

(2) 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.

(3) 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 broadcasted synchronously to all clients in the same working group, and all clients retrieve the updated files from the server to the local disk automatically.

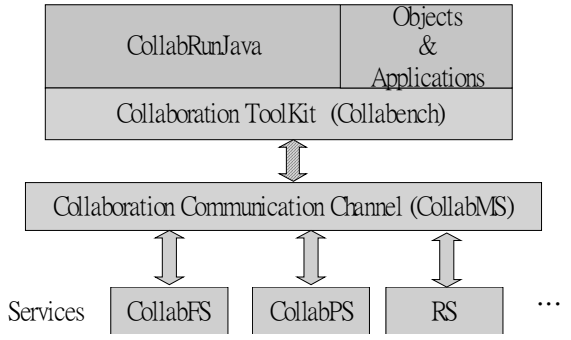

Figure 1 The ShareTone collaboration platform

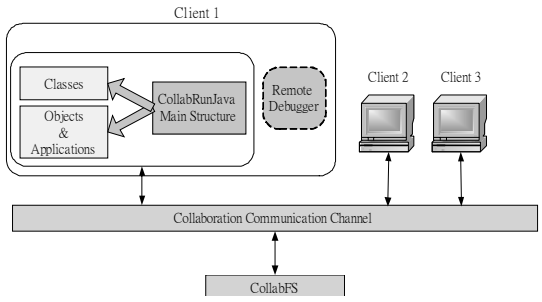

Figure 2 A dual mode exerciser on a collaborative computing environment

3 System Design

Our system provides a collaborative development tool and a runtime platform. Users can develop applications standalone or collaboratively. All basic development 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 runtime-evolution mode. In the instant-develop mode, users can develop applications rapidly with useful tools. User can also manipulate objects directly through an embedded interpreter. The runtime-evolution 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.

3.1.1 The Instant-Development Mode

After a user joins 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 the main part of CollabRunJava. Programmers can use CollabRunJava to write source codes, compile these codes into Java classes and perform other developing activities.

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 those 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 validate or manipulate the classes immediately after development by

interpreting testing codes in RunJava. If problems occur in these classes, programmer can switch to the runtime evolution mode to debug, modify and re-compile these classes without re-execute the applications. This improves the performance of application development.

Since CollabRunJava is a collaborative environment, all operations are synchronous, including editing, code interpretation, and file access. The interpretation and editing operations will be sent to the collaboration server first, and broadcasted 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 these resources are modified, the collaboration server will modify and update the corresponding files. Once the server updates the project resources, it will notify the other clients to update their local files synchronously. To enhance flexibility and performance, the locking mechanism is not used on 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.

3.1.2 The Runtime-Evolution 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 runtime-evolution 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-executing 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 functionality of the original program, without changing 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. Examples of Geometric algorithm visualization will be addressed later in this paper.

However, some limitations of this feature should be noted. (1) If the modified methods are in 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. On the contract, 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 later.

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. To reduce code duplication and makes components easier to maintain, all plug-ins are designed according to the Model-View-Control (MVC) architecture. The user interfaces of the plug-ins are customized. Developers can, therefore, design a graphic user interface specific to their needs. We provide a simple collaborative project editor. It allows users to edit shared source files, which are set share-enabled, collaboratively. The compiler is used to compile source codes and generate the Java classes. Currently, it uses the Sun Java compiler 1.4.1 version.

There are two 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 development. In RunJava, several useful functions

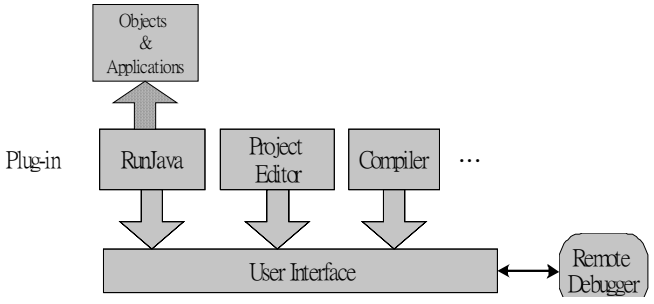

Figure 3 The architecture of CollabRunJava

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 runtime-evolution 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 runtime-evolution mode is activated.

In our design, we decide to integrate user interfaces in the main process. The reason behind this decision is because most common user activities are editing, file accessing, compiling and interpretation. A lot of communication between processes can be eliminated, if the integrated user interface is in the 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.

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 development environment and the executing applications are in the same process. However, if users want to suspend the executing applications for observation or modification, the remote debugger will suspend not only 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

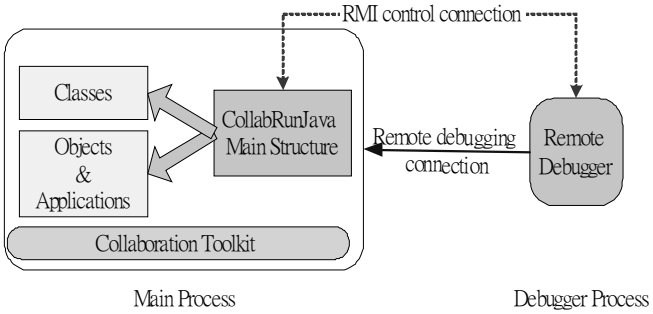

Figure 4 Communication between two processes

these changes, the remote debugger will only suspend the specified application, not the whole system.

4.3 The Issue of Late Comers

In addition to keep all online users acting synchronously, a collaborative system must also allow the late-comers with the same status as the existing users. Although the collaboration service of ShareTone achieves the former, the late-comer issue needs more efforts.

In CollabRunJava, the RunJava contexts of users in the same group 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 operations 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.

The above "replay" mechanism may take a long time to restore the current state if the amount of history command is large. Our system also provides a serialization mechanism to overcome this 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. However, 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, it doesn't support the collaboration mechanism for all functions in runtime-evolution 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 if the debugging process id performed individually on each client. Therefore, we adopt the centralized sharing approach here and use the virtual collaboration mechanism to simulate collaborative debugging in the runtime-evolution mode. Only one remote debugger will be notified to make a remote debugging connection to the main process in this client as shown in Figure 5. 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 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. The other

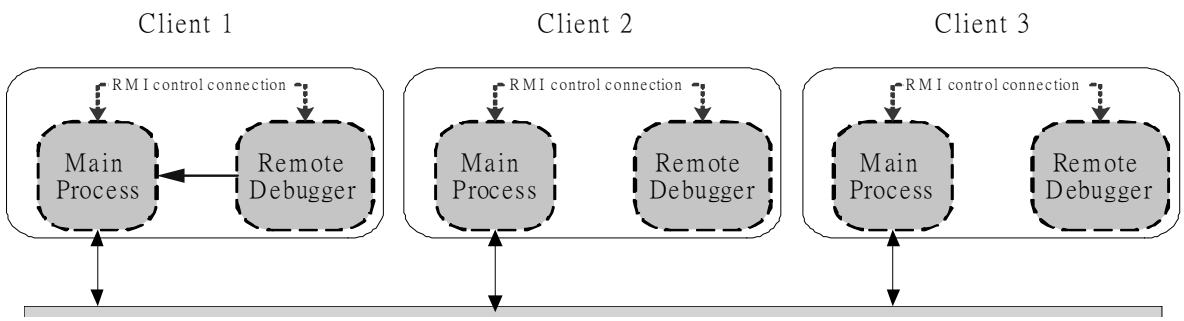

Figure 5 Multi-user control of remote debugger

clients can observe and operate the debugging process on the debugging executor.

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 that 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 default plug-in component, called the GeoDrawing bean, to draw geometric algorithms and let users dynamically modify and debug algorithms as described in Section 3. In order to develop customized 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 annotation code insertion/deletion, monitor variable insertion/deletion, and variable monitor listener addition. The details will be described in next section.

```
VisualBeanRuntimeInterface{
//methods of bean manipulation
setGeoData(GeoData data)
getGeoObject(String lable)
 deleteGeoObject(String lable)
 addGeoDataChangeListener(GeoListener)
 redraw()
 compute()
//methods of variable monitoring
  getMonitorVariableTable();
 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].

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

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.

6 Related Work

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[16] and DrJava [17] are development environments for Python and Java language, respectively. Both of them provide a file browser, a source editor and a graphic debugger. In WingIDE, the runtime and design-time are separated, and

the user needs to re-execute the applications after re-designing them. This system lacks the interactive programming ability. Because DrJava has an interpreter embedded, it has interactive programming ability. But it still need to re-execute the application after modify its codes. Both of the WingIDE and DrJava are designed to develop programs standalone.

7 Conclusion and Future Work

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

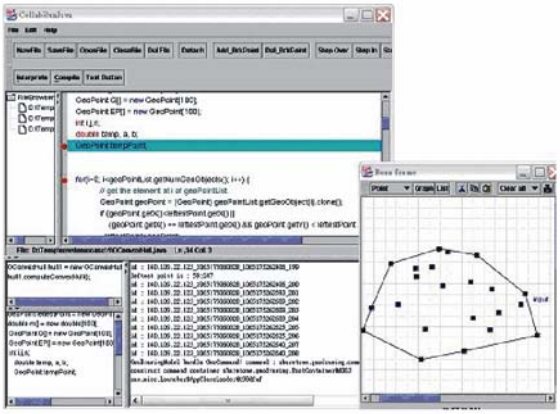

Figure 6 A snapshot of CollabRunJava and GeoDrawing

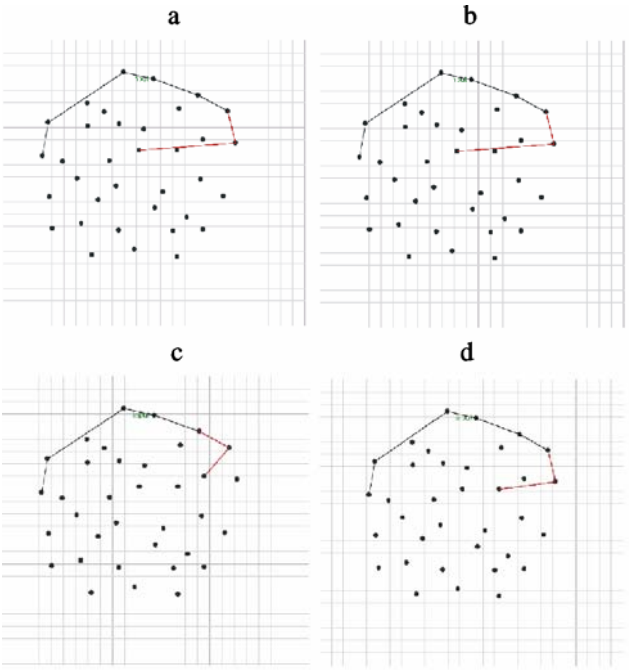

Figure 7 Incremental Visualization: Display the intermediate result of drawing convex hull step by step

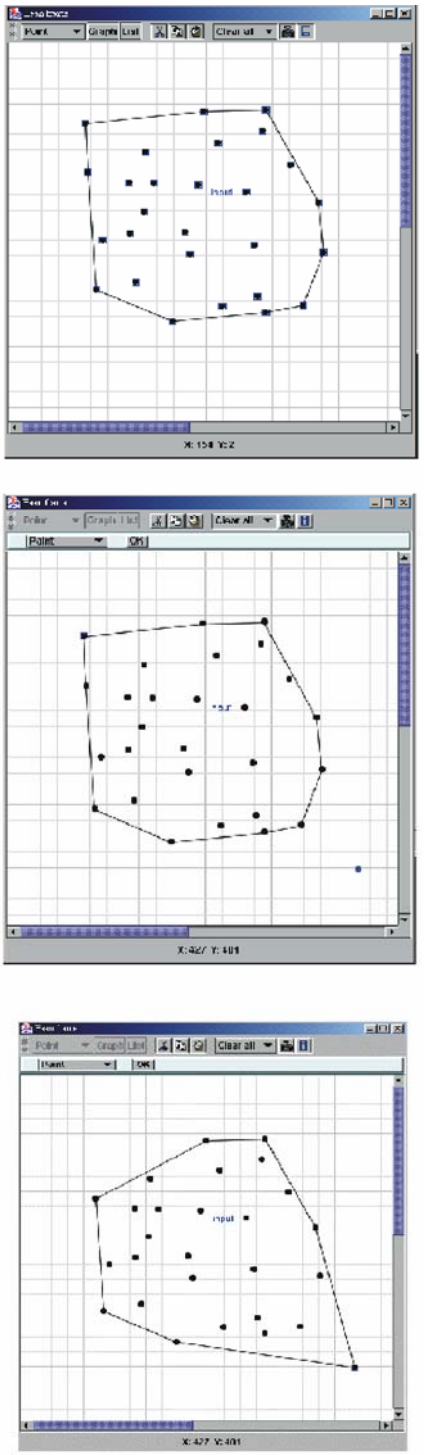

Figure 8 Interactive Visualization: Add a new point on and display the re-computed result immediately

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 anonymous referees for their insightful comments on an earlier version of this paper. Their suggestions have greatly enhanced the quality of this paper. This work was supported in part by National Science Council under Contract Nos. NSC92-2213-E-001-015 and NSC93-2213-E-001-008.

References

[1] C. A. Ellis, S. J. Gibbs and G. L. Rein, Groupware: Some Issues and Experiences, *Communications of the 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. on Computer-Human Interaction*, Vol. 3, No. 1, March 1996, pp. 66-106.

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

[5] J. Altmann, R. Weinreich, An Environment for Cooperative Software Development Realization and Implications, In *Proc. of the 31st Hawaii International Conference on System Sciences, Collaboration Systems and Technology (HICSS-31)*, Big Island, Hawaii, USA, Jan. 6-9, 1998.

[6] M. Gaeta and P. Ritrovato, Generalised environment for process management in cooperative software engineering, In *Proc. of the 26th Annual International Computer Software and Applications Conference (COMPSAC 2002)*, Aug. 26-29, 2002, pp. 1049-1053.

[7] L. Osterweil, Software Processes are Software tool, In Proc. *of the 9th. International Conference on Software Engineering*, New York, USA, 1987, pp. 2-13.

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

[9] A. Kerren and J. T. Stasko, Algorithm Animation - Introduction, In *Proc. of Dagstuhl Seminar on Software Visualization*, 2001.

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

[11] A. Hausner and D. P. Dobkin, Gawain: Visualizing geometric algorithms with web-based animation, In *Proc. of Symposium on Computational Geometry*, 1999, pp. 411-412.

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

[13] D. T. Lee, C. -F. Shen and D. S. Sheu, Geosheet: A Distributed Visualization Tool for Geometric Algorithms, *International Journal of Computational Geometry and Applications*, 1998, pp. 119-156.

[14] C. Demetrescu, I. Finocchi and J. T. Stasko, Specifying Algorithm Visualizations: Interesting Events or State Mapping? In *Proc. of Dagstuhl Seminar on Software Visualization*, 2001.

[15] M. sken and Stefan Naher, GeoWin A Generic Tool for Interactive Visualization of Geometric Algorithms, In *Proc. of Dagstuhl Seminar on Software Visualization*, 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, In *Proc. of the Workshop on Engineering Complex Object-Oriented Systems for Evolution*, Tampa Bay, Florida, USA, Oct. 14-18, 2001.

Biographies

Chien-Min Wang received a B.S. and a Ph.D. in electrical engineering from National Taiwan University in 1987 and 1991, respectively. Since then he joined the Institute of Information Science as an assistant research fellow. In January 1996, he became an associate research fellow of the institute. His major research interest includes parallel compilers, parallel algorithms, parallel computer architectures, and object-oriented technology. Dr. Wang is a member of IEEE Computer Society.

Shyh-Fong Hong received a B.E. degree and a M.S. degree in computer science and information engineering from National Chung Cheng University, Chia-Yi, Taiwan, in 1996 and 2000, respectively. He has been a software engineer at Institute of Information Science, Academia Sinica, Taiwan, since 2000. His main areas of interest include Java system and mobile communication protocols.

Shun-Te Wang received his B.E. degree in mechanical engineering from National Taiwan University of Science and Technology in 1996. He received the M.S. degree in engineering science from National Cheng Kung University, Taiwan, in 1999. He has been a software engineer at Institute of Information Science, Academia Sinica, Taiwan, since September 1999. Currently he is a doctoral candidate in the Department of Electronic Engineering at National Taiwan University of Science and Technology. His main areas of interest include pervasive computing, sensor networks, ad hoc networks, wireless networks, mobile computing, communication protocols, component software, software warehouse, authoring system, and Java collaborative computing.

Hsi-Min Chen received his M.S. degree in computer science and information engineering from National Central University, Taoyuan, Taiwan, in 2000. He has been a software engineer at Institute of Information Science, Academia Sinica, Taiwan, since 2001. He is also a Ph.D. student in the Department of Computer Science and Information Engineering at National Central University. His major research interest includes software engineering, computer supported cooperative work, service-oriented software technology, and grid computing.