Experiences in Implementing a Java Collaborative IDE over the Internet

CHIEN-MIN WANG, SHUN-TE WANG, SHYH-FONG HONG and HSI-MIN CHEN
Institute of Information Science, Academia Sinica
Nankang 115, Taipei, Taiwan, R.O.C.
{cmwang, wangsd, fong, seeme}@iis.sinica.edu.tw

Abstract

A collaborative computing environment supports groups of people as they cooperate to achieve their goals. This report presents a system architecture to support the application areas in collaborative authoring and editing. An experimental platform, called the i-Code system, has been developed. It is a Java collaborative integrated development environment (IDE) tool designed to help collaboratively build Java applications for collaborative computing. Not only the IDE itself, but also the Java applications, are collaborative. Furthermore, in a computer-supported cooperative work (CSCW) environment, event broadcasting is a common operation in an object-oriented software system to solve many issues. In a multi-user case in particular, due to scalability, such operations are expected to be more quickly and reliably executed. In this report, we provide a study of issues related to event broadcasting in a Java collaborative computing environment over the Internet. We address how these issues happen and propose solutions for them.

Keywords: CSCW, Java, event broadcasting, collaborative computing, i-Code

1 Introduction

Computer-Supported Collaborative Work (CSCW) is the study of how people work together using the enabling technologies of computer networking and associated hardware, software, services and techniques. Much CSCW research has focused on developing new theories and technologies for applications of real-time teamwork [1]. With the advance of computer and communication technologies, communication over the Internet has become commonplace and collaboration via the Internet has recently gained popularity. To support groups of people as they cooperate to achieve their goals, the idea of a collaboratory seems to be a reasonable objective. The aim of a collaboratory is to enable remote users with expertise in specific areas to collaborate with one another to solve a particular problem by viewing the shared data that is pertinent to each user's specialty [2].

A collaborative computing environment involves a suite of software systems, communications protocols and tools that enable collaborative and computer-based cooperative work. To implement a collaboratory which refers to an integrated, tool-oriented computing and communication system that supports collaboration, some technical problems must be solved. For example, if graphics output is involved, all sorts of display devices have to be supported. Since Internet is readily accessible by many people in the world, building a collaboratory on the

Internet seems to be a most reasonable solution. The Java programming language developed by Sun Microsystems, which is considered platform-independent, is a suitable choice to implement such a collaboratory [3][4][5].

Some Java tools are currently available to broadcast presentations such as audio, video, electronic slides, etc., over the network to a distributed audience at their desktops. Some of the tools focus on heavy-weight applications such as videoconferencing. They are suitable for providing telepresence when a high level of interaction is needed. In the application areas of collaborative authoring and editing, tools are also available to support people, so they feel as though they are working together. One of the key factors in the success of these tools is an authoring system for distance learning and demonstration. Figure 1 shows a possible application of collaborative authoring and editing, which allows professionals to develop teaching modules to help users learn computational geometry algorithms in multimedia form. Combined with the support of collaboration, it allows professionals all over the world to work together via a common graphical user interface [6].

Some existing software, such as NetBeans and Eclipse, are popular freely available Java IDEs that offer a very flexible and extensible platform where a growing number of collaborative applications are being offered [17][18]. However, in the field of teamwork software development, most of current available plug-ins for them are devoted to finding and sharing defects of any files with a team, or providing methods of management of code annotations, storing XML based file, which can be shared with the team by a concurrent versions system (CVS). In addition, there are some plug-ins focusing on visual UML modeling and CASE tools with teamwork support [19][20]. As we know, current existing IDE tools and their plug-ins cannot support collaborative graphical user interface (GUI) design process.

In this report, we discuss ways of applying the Java technology to support the collaborative activity of groups that are distributed across different locations. From the programmer's perspective, the IDE is where coding takes place and is the home of many different development tools. Since coding can be a team effort, we can add collaborative capabilities to the IDE toolset alongside the editor, compiler, and debugger. Hence, we have developed the i-Code system, a Java IDE tool designed to help collaboratively build Java applications for collaborative computing. Note that not only the IDE itself, but also the Java applications, are collaborative. As a development testbed, the i-Code system currently does not have its plug-in version for certain universal tool platforms like NetBeans or Eclipse. We will contribute our efforts to the free software community in the near future.

In addition, because of the nature of Java programming language, some problems may occur when we implement tools for collaborative authoring and editing. This report also discusses the problems we may encounter, and possible solutions we can adopt, when a collaborative Java integrated development tool is constructed. In a CSCW environment, event broadcasting is a common operation used to solve many problems in an object-oriented software system. In a multi-user case, in particular, such an operation has to be quickly and reliably executed, because of scalability. Some common issues, e.g., event loss, replay, and event duplicates may occur because of different implementations. In this report, we present a study of the issues on event broadcasting in a Java collaborative computing environment. We also show how these issues occur and propose solutions for them.

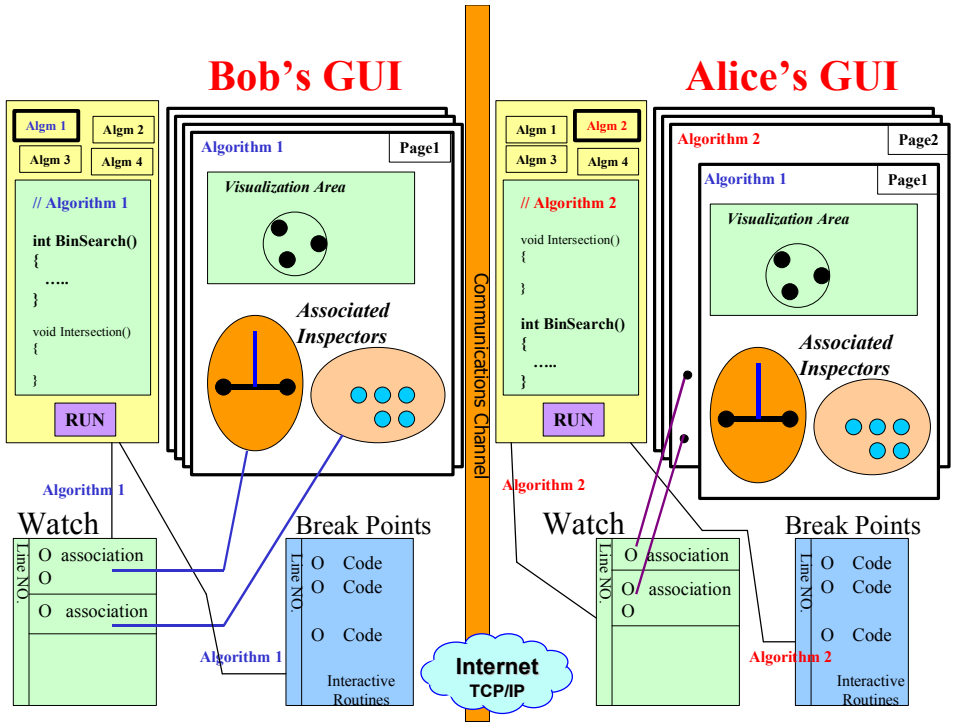

Figure 1. An application in collaborative authoring and editing.

2 The i-Code System

The i-Code system is an integrated, tool-oriented computing and communication system. It enables remote users with expertise in specific areas to collaborate with one another, viewing the shared data that is pertinent to each user's specialty, in order to solve a particular problem. The i-Code system allows remote parties to gain access to scientific resources such as software libraries, as well as expensive and physically large equipment that would not otherwise be accessible. It can help professionals to develop teaching modules that contain information in multimedia form and that can be demonstrated on-line.

As shown in Figure 2, the i-Code system contains the following components: a) A graphical user interface named GUI-Builder that allows the interactive development of collaborative applications. b) An interpreter named RunJava for the on-line manipulation and execution of executable Java source codes. c) A centralized tool called Communicator for message broadcasting on the server. d) Some JavaBeans used for visual programming.

2.1 The GUI-Builder

The GUI Builder is an integrated development tool used to quickly and easily develop Java applications for collaborative computing. It enables users to collaboratively create graphical user interfaces and saves a lot of time compared to hand-coding for GUI development and maintenance tasks.

The GUI Builder allows collaborative users to utilize or just try out their own newly created beans in a user-friendly manner. Users provide their own JavaBeans, and then the GUI Builder allows them to drop their beans onto a composition area, resize and move beans around, edit the exported properties of a bean, connect a bean event source to an event handler method, connect together bound properties on different beans, save and restore sets of beans, make applications from beans, get an introspection report on a bean, add new beans from JAR files, and toggle execution environment between runtime and design mode

The GUI Builder contains all the basic components found in a typical window, such as a text label, a scrollbar and a radio button. All components are implemented as JavaBeans. In the GUI Builder, these JavaBeans have attributes that users can modify collaboratively. Figure 3 shows a snapshot of the GUI Builder.

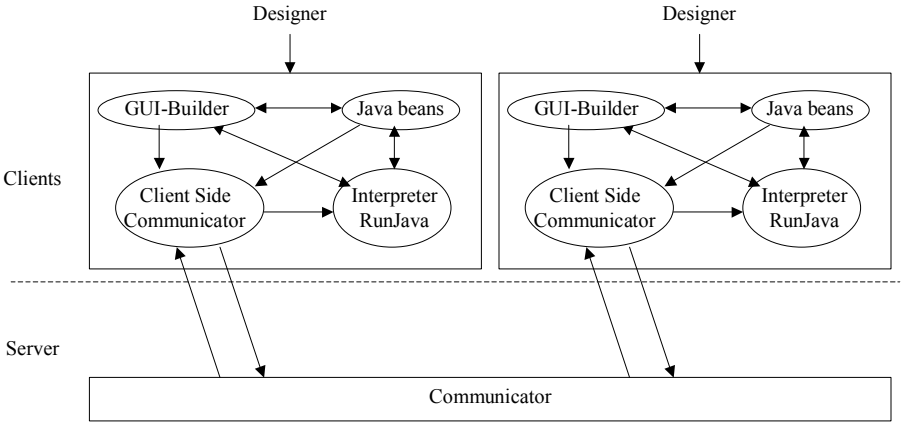

Figure 2. The i-Code system architecture.

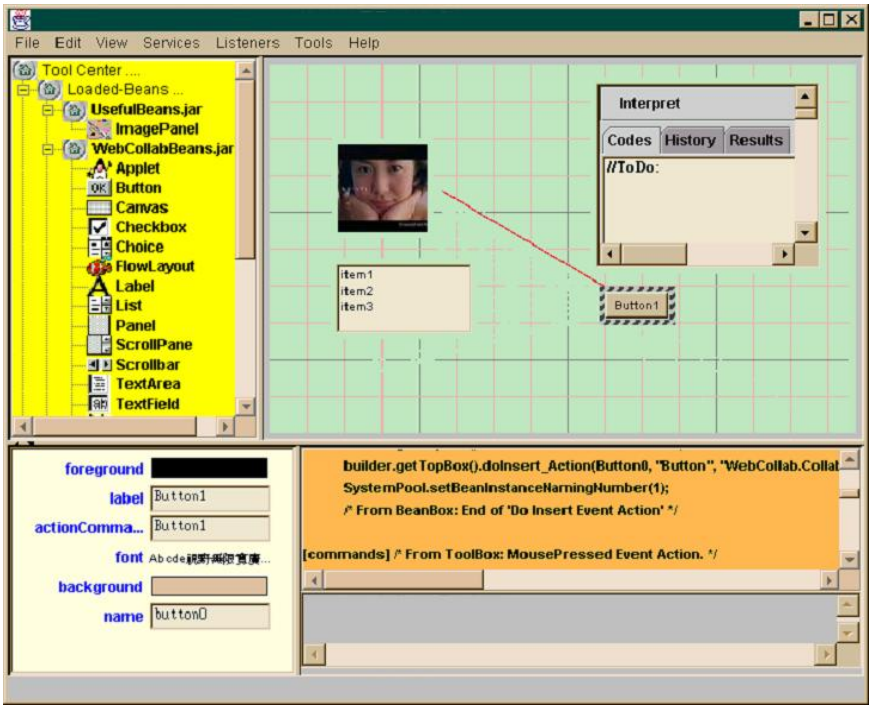

Figure 3. A snapshot of the GUI Builder.

2.2 The RunJava Interpreter

The RunJava is a Java source code interpreter written in pure Java. It is embedded in the i-Code System to provide extensive scripting for collaborative applications, and uses the Java reflection API to provide runtime execution of Java statements and expressions. The RunJava is similar to the DynamicJava interpreter [21]. However, the RunJava has more capabilities than DynamicJava.

The RunJava is responsible for class loading. It loads the basic classes when itself is initialized. It will later load the required classes in the class path of JavaBeans, or those classes developed by the on-line users, to save the memory space.

The RunJava also supports dynamic class loading, so it is easy to check or operate the classes immediately. If problems occur in these classes, programmer can switch to the debugging mode, modify, re-compile and reload these classes without re-execute the whole application (We do not report the collaborative debugging mechanism in this article because it is complicated and could be another topic for study and discussion.)

. The RunJava interpreter allows users to create object instances, invoke methods and examine the contents of objects interactively at run-time, as shown in Figure 4. In addition, it has the ability to capture user operations in a log file and to replay these captured operations later. It also has the ability to store object contents in a data file and compare the contents of two objects. These features are very useful for testing and debugging.

The RunJava interpreter is enhanced to accommodate late-comers to a collaborative activity. Because the i-Code system employs a replicated distribution architecture, the application of each participant maintains a copy of the shared state. For example, in a GUI-building application, this state may comprise the number of JavaBean instances and the properties of those instances. Without information about the current shared state, it is not possible to participate in a session.

Late-comers may obtain the current shared state with the aid of the Java serialization mechanism. Serialization involves saving the current state of an object to a stream, and restoring an equivalent object from that stream. However, because not all Java classes are designed to be serializable, it is not easy to serialize all the current shared states. To deal with this problem, although there is a little performance penalty for using a source code interpreter, the RunJava interpreter can ask for a log file that keeps track of some Java source code of user operations, using the History Replay service provided by the Communicator, and replay the log file to reconstruct the current shared states and then participate in current session. This can be regarded as one kind of a version control system that helps users collaboratively work on the same project.

Figure 5 shows the graphical user interface of the Communicator for user logon. When a late-comer provides his/her name and password, and then chooses to join a current project, the Communicator will send a copy of the current log file to the RunJava interpreter. After the RunJava interpreter replays the log file, the current shared state is then reconstructed for late-comers to catch up with the working project.

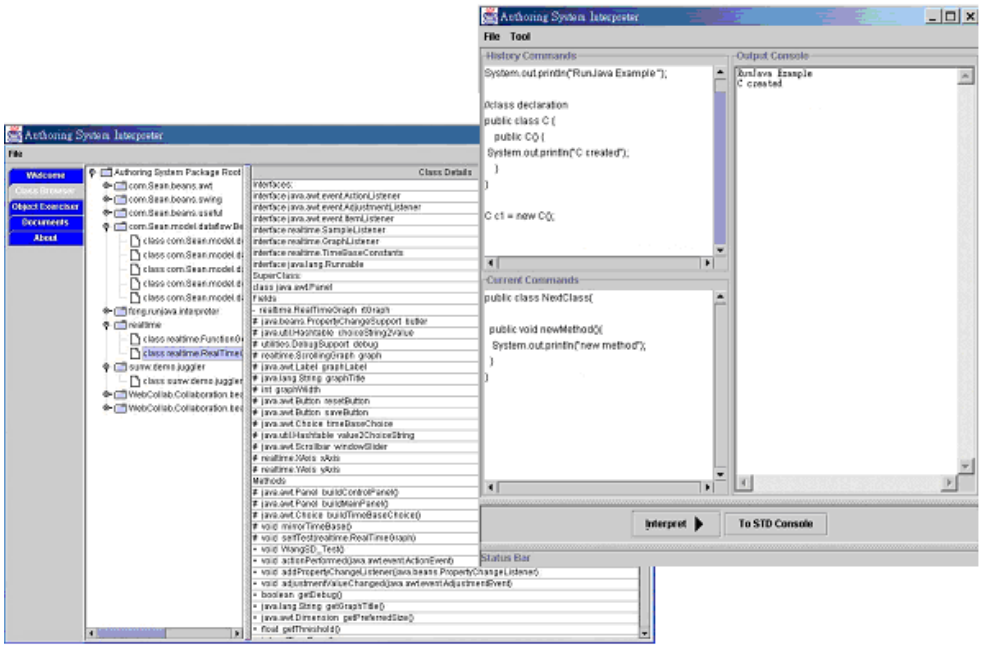

Figure 4. The RunJava Interpreter.

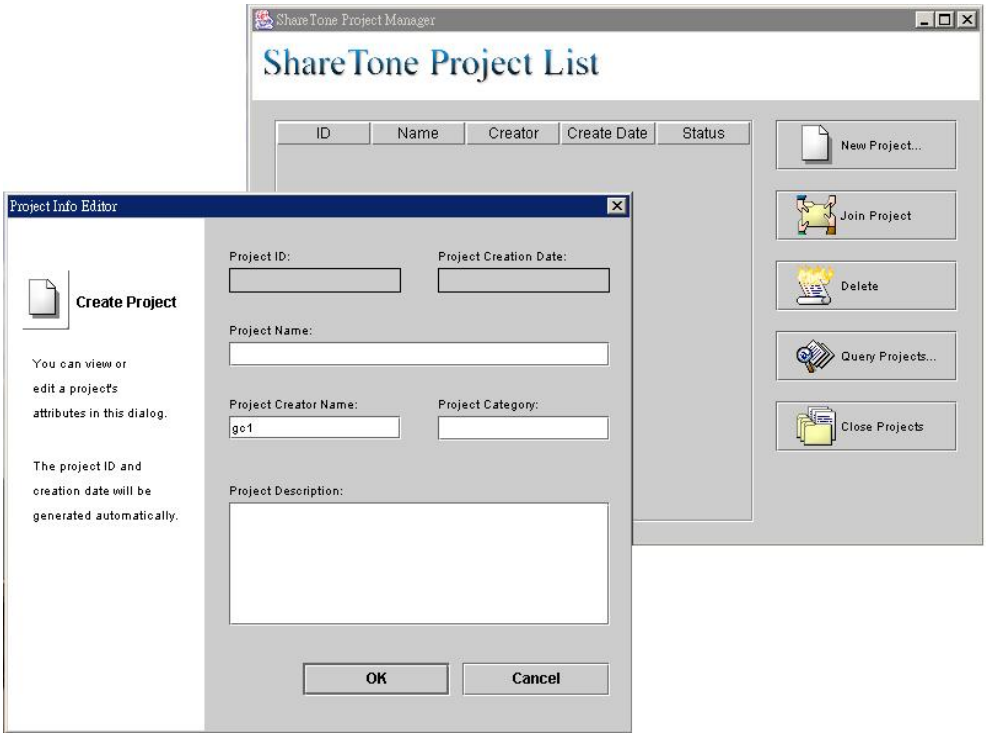

Figure 5. Project logon.

2.3 The Communicator

The Communicator has two main functions. One is to support collaboration, by communicating with some Java object on the clients. Messages from a Java object will be broadcast to all the other Java objects in the same communication channel. The other function is to provide network transparency. It receives user commands through the network and interacts with the software components to complete the on-line manipulation and execution of the library of software. It also records the contents of communications.

The GUI-Builder and RunJava interpreter are manipulated as client side applications, whereas the Communicator acts as a server to coordinate the communications among the GUI-Builders. The Communicator is also implemented using Java so that it can be ported to any platforms and run everywhere.

In a collaborative environment, distributed clients who manipulate the GUI-Builder can cooperate to develop Java applications together with the help of the Communicator. The developed applications assembled by the GUI-Builder also have collaborative ability when they later connect to the Communicator.

We can regard the Communicator as a mediator that receives collaborative objects sent from one client and broadcasts them to others. In addition, the Communicator also furnishes the following additional services: history replay, design-phase project management, application-phase project management, and the remote compilation of Java source codes.

3 Collaboration Schemes

The idea of providing the real-time shared use of legacy single-user applications is usually called *collaboration transparency*. To enable the collaborative use of existing single-user JavaBeans, the support of collaboration transparency is required. In this way, programmers may not need to design other collaborative versions of JavaBeans that can only work in certain collaborative infrastructures [7][8][9]. One key to the success of collaboration transparency is the event broadcasting function provided by the JavaBeans container in the i-Code system.

Because the i-Code system is designed to be a builder tool with visual programming environment, the JavaBeans reusable component architecture hence needs to be supported. From the viewpoint of the ability for collaboration, the JavaBeans can be classified as *collaborative JavaBeans* and *Noncollaborative JavaBeans* (standard JavaBeans). In the i-Code system, three collaborative schemes are provided.

3.1 Collaboration of GUI-Builders

First, a collaboration scheme for the GUI-Builder of the i-Code system must be established. As shown in Figure 6, the GUI-Builder itself is a collaborative JavaBean, which has knowledge about the underlying communication mechanism. It can directly send some String-Commands to the Client-Side Communicator to asynchronously notify other corresponding components that belong to the same working group to keep their states the same. For example, if a JavaBean instance in the GUI-Builder is deleted, the GUI-Builder then sends a “delete” command to the Client-Side Communicator. The Server-Side Communicator will then broadcast the command to

all the other corresponding i-Code clients. The “delete” command is then executed by the interpreter to keep their GUI states the same.

Distributing JavaBean events (e.g. MouseClicked) across Java virtual machines is also involved in this collaboration scheme. For the GUI-Builder, a default Listener class is provided, which implements many Java event listener interfaces. It captures the user interaction and passes the information along to the String-Command Generator, which will then return some corresponding String-Commands to the Listener object. Note that the Listener object doesn’t call back to the GUI-Builder, but it simply sends the String-Commands to the Client-Side Communicator to broadcast them. After the other i-Code clients receive these String-Commands, they execute them via the Interpreter to reproduce the user interaction or events.

3.2 Collaboration among noncollaborative JavaBeans

For noncollaborative JavaBeans, another collaboration scheme must be provided. Because noncollaborative JavaBeans are designed to be portable to containers, they have no knowledge about the underlying communication mechanism provided by the i-Code system. Therefore, the interactions among these kinds of JavaBeans cannot be transformed automatically to some String-Commands to produce the effect on collaboration. For example, in Figure 7, for example, let Bean1 be a Button and Bean2 be a Label. If any user of the i-Code system presses the Button, the Label and every other corresponding Label must be informed to perform a specified action such as *setText("pressed")*. Note that in this case, the *MouseClicked* event on the Button is not necessarily broadcasted, but the specified action on the Label must be specified and broadcasted to achieve collaboration.

In this collaboration scheme, a simple Listener class is implemented. The Listener object registers itself to every instance of noncollaborative JavaBeans when the instance is constructed. If the Listener object receives an event, it then generates some skeleton Java listener codes and invokes a collaborative editing process (called the String-Command Generation process) for users to edit and specify the essential actions to be done on other noncollaborative JavaBeans. After collaborative editing, the file of listener codes is compiled to a Java class file and returned to the Listener object. The Listener object then produces some String-Commands and broadcasts the class file and these String-Commands. The Interpreter will then create an instance of the class file and bind the bean instances by executing the String-Commands.

3.3 Collaboration among collaborative JavaBeans

For collaborative JavaBeans, a collaboration scheme similar to that of the GUI-Builder is provided, as shown in Figure 8. A collaborative JavaBean (Co-Bean for short) is designed to have the knowledge about the underlying communication mechanism. If necessary, it will directly send some String-Commands to the Client-Side Communicator to asynchronously notify every corresponding Co-Bean object that belongs to the same working group to keep their states the same. For example, if the property of a Co-Bean object in the GUI-Builder is changed, the Co-Bean object sends a "propertyChange" command to the Client-Side Communicator. The Server-Side Communicator will then broadcast the command to all the other corresponding i-

Code clients. The "propertyChange" command is then executed by the interpreter to make the properties of all Co-Bean objects the same.

To distribute Co-Bean events (e.g. PropertyChange) across Java virtual machines, for a Co-Bean, a Listener class and a String-Command Generator class must be provided when the Co-Bean is designed. The Listener class can implement some Java event listener interfaces. It captures the user interaction and passes the information to the specified String-Command Generator, which will return some corresponding String-Commands to the Listener object. Note that the Listener object doesn't call back to the Co-Bean, but it simply sends the String-Commands to the Client-Side Communicator for broadcasting. After the other i-Code clients receive these String-Commands, they execute them via the Interpreter to reproduce the user interaction, or events on the Co-Bean.

In the next section, we describe the design issues that occur when we implement the above three collaboration schemes.

3.4 Converting JavaBeans into Collaborative JavaBeans

Although the i-Code System provides collaboration transparency to enable event broadcasting for noncollaborative JavaBeans, it cannot always satisfy certain special requirements. Sometimes, the bean has to perform event broadcasting itself to support collaboration.

In this subsection, we describe the API specification for converting any JavaBeans into collaborative JavaBeans that can be utilized by the clients of the i-Code System. As an example, Figure 9 shows the class diagram for a collaborative TextArea. The concrete class ServiceProvider provided by the i-Code system implements the underlying broadcasting function and communication mechanism that can be called by the TextArea JavaBean. In addition, the class TextArea must implement three interfaces: BeanRole, BroadcasterGate, and RootBean.

The interface BeanRole provides methods to enable or disable the collaboration function. The interface RootBean provides a method to re-read all the properties and update the property editors with other properties that have changed in the GUI-Builder. The interface BroadcasterGate provides methods for the TextArea JavaBean to interact with the GUI-Builder.

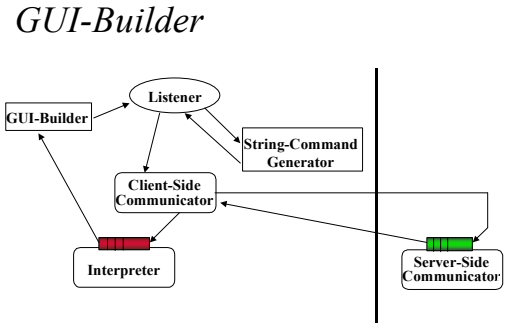

Figure 6. Collaboration of GUI-Builder.

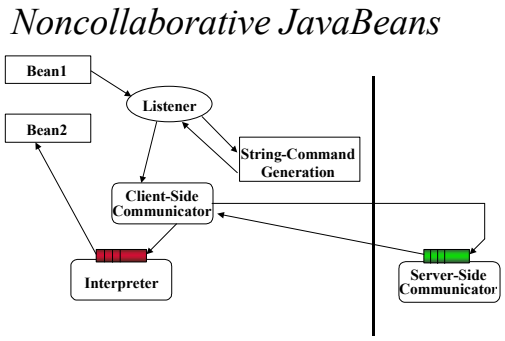

Figure 7. Collaboration among noncollaborative JavaBeans.

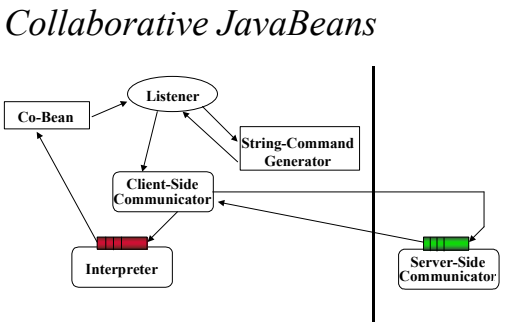

Figure 8. Collaboration among collaborative JavaBeans.

4 Issues on Event Broadcasting

In an object-oriented programming language such as Java, communication between the user and the program occurs entirely in the form of events. The Java Virtual Machine (JVM) obtains raw event data from the operating system and creates event objects for the program to process further. The Java 2 Platform uses the delegation model to deal with events. The delegation model works like the observer-observable design pattern, in which the observable is the component that generates events, and the observer is the object that has registered to receive events. Java 2 handles events that are generated in a graphical component by delegating to objects that have registered an interest in that type of event. This model works well in a single JVM. However, it cannot work directly in a collaborative system.

The development of collaborative tools demands the creation of suitable paradigms and related run-time infrastructures [10]. In an event-based paradigm, collaborative components interact by generating and consuming events [11][12][13]. The occurrence of an event in a certain component is asynchronously notified to any other corresponding component that belongs to the same working group, as shown in Figure 10.

Usually we can write Java network I/O code, or use the Java Message Service (JMS) API, to facilitate this communication. However, for collaborative computing, both of these approaches are error prone and time consuming.

4.1 The issue of event loss

To enable the collaborative use of existing single-user JavaBeans, the support of collaboration transparency is required. One key to the success of collaboration transparency is the event broadcasting function. Because communication must be reliable in collaborative computing, event broadcasting is usually implemented using Java TCP APIs. This provides a reliable, point-to-point communication channel that client-server applications use to communicate with each other, i.e. no data can be dropped because it must arrive on the client side in the same order in which the server sent it. If an event is sent using a TCP connection, the event loss issue will not appear during transmission from the programmer's point of view. However, an event may be lost either before, or after, broadcasting.

First, event loss may happen when converting low-level events to high-level events. Because the operating system and user interface only send low-level events, each component's peer first listens for low-level events like key presses or a combination of low-level events. The component then consumes those events and fires new high-level events such as ActionEvents, TextEvents, ItemEvents, or AdjustmentEvents. For example, when the user clicks the mouse on a button, then releases it, the button receives two or three separate, low-level, mouse events, e.g. one for mouse down, one for mouse up, and one for mouse drag if the user moves the mouse while the button is pressed. The button then fires just one high level ActionEvent. Usually, for simple JavaBeans like Buttons, the broadcasting of high-level events is sufficient for collaboration, since the high-level semantic events encapsulate the meaning of a user interface component. However, for some specific JavaBeans, the low-level events are important and cannot be ignored. A simple drawing utility such as a whiteboard, for instance, is sensitive to low-level events. Due to the frequent arrival of the low-level events, the practical implementation of the event broadcasting function usually aggregates those low-level events and broadcasts them later to avoid a waste of bandwidth. However, the design of low-level events aggregation is usually application-specific and event loss may occur if the design is poor.

Event loss may occur after events are broadcasted. This happens in a multi-threaded environment. Because the event-handling code usually executes in a single thread, when the next broadcasted events are sent to their event listeners, each event handler will finish the current execution before the next one. This means that the code in event handlers should be executed very quickly, otherwise, the application's perceived performance will be poor. If in a collaborative application, a certain event must be processed within a particular period of time, then the out-of-date broadcasted events may be dropped by event handlers. As the performance of every user's computer is different, some users will see the result of the broadcasted events, while others may not.

Due to the frequent arrival of certain events, the practical implementation of the event broadcasting function can aggregate many events and broadcast them later, if the broadcasting of high-level events is sufficient for collaboration. To avoid both the event loss issue and the waste of bandwidth, the Balking design pattern can be used [14]. Table 1 shows an implementation for the collaboration of ScrollPane. The ScrollPaneListener uses a thread to collect the AdjustmentEvents. When there is no further event arrival within 0.5 seconds, the final states of the ScrollPane (e.g. the Adjustables) are broadcasted.

If the code in event handlers is executed slowly, the application's perceived performance will be poor. To deal with problem, the event handlers can create a thread to execute the work associated with the newly arrived event, if the events are independent in sequence.

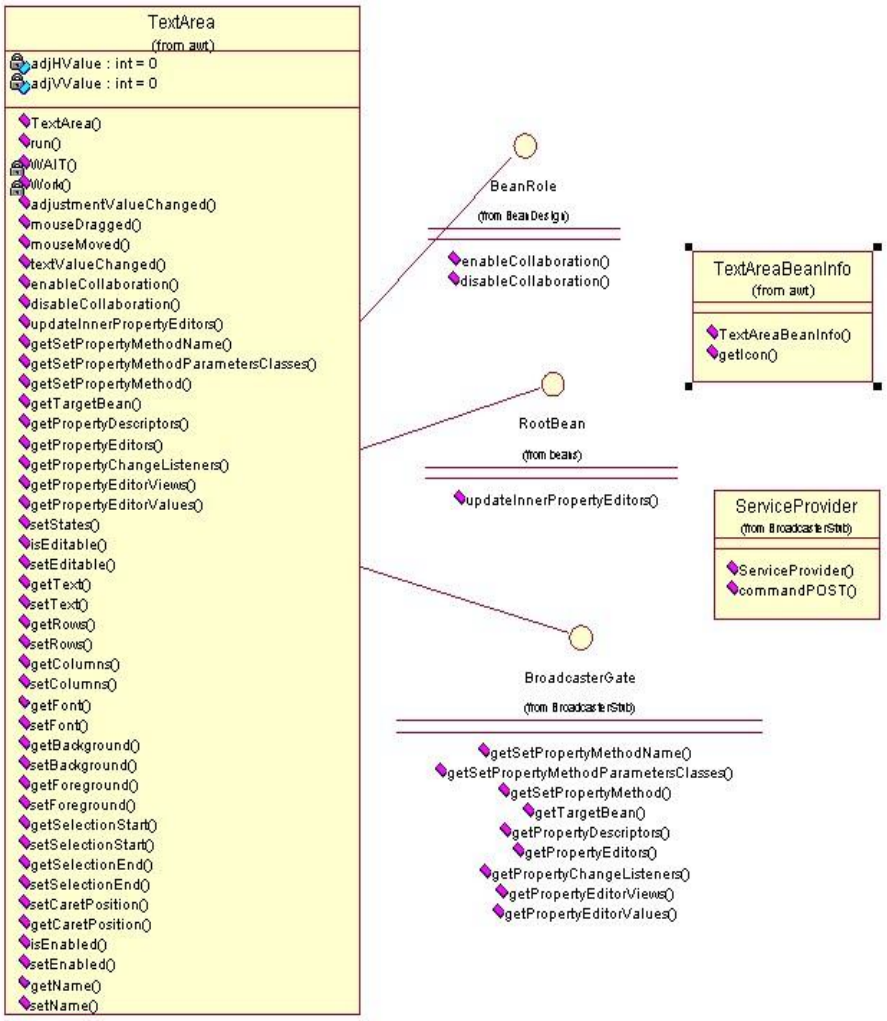

Figure 9. Class diagram for a collaborative TextArea.

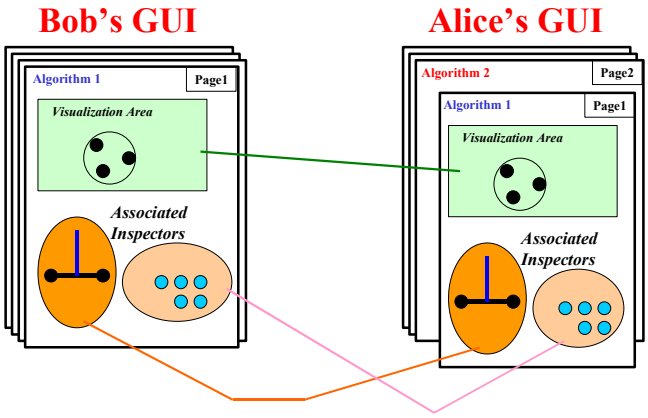

Figure 10. Asynchronous notification to the other corresponding components.

4.2 The issue of replay

The replay issue occurs when a component's peer handles the low-level events immediately and then fires new high-level events. For example, a TextField shows the typed characters immediately after the keys on the keyboard are pressed, but it broadcasts the TextEvents later. Once the broadcasted TextEvents loop back to the source of the high-level events, those characters will be processed and printed again. This replay may not be harmful if the following event order issue does not appear. However, if the JavaBean is a Counter object, the replay must be ignored to prevent itself from being counted twice.

For a graphical JavaBean, the event-handling and drawing methods are usually expected to be executed quickly. That is, a JavaBean can repaint itself immediately after a specified event occurs to avoid an unresponsive situation. In this case, the low-level and the user-defined high-level events must be processed first, and the replay of loop-back events must be ignored.

To prevent the event source from replaying a loop-back high-level event, the event source can broadcast an event with an identification number (i.e. a local timestamp) embedded. The event handlers can later check if the event with the identification number has been actioned. The event source will not replay a loop-back high-level event, if the identification number was issued by itself. This solution works, but its shortcoming is that the event handlers must check every time an event arrives.

4.3 The issue of event order

A floor control mechanism allows users of networked applications to share resources without access conflicts. Temporary permission is granted dynamically to certain users to guarantee mutually exclusive access.

The event order issue is related to the replay issue if no floor control mechanism is introduced to the system. For example, if two users are typing in a collaborative TextField, the TextEvents from both are broadcasted. In this case, the effect of the TextEvent that comes first may not be seen or is just held for a very short time, because the effect of the later TextEvent will override the former.

Note that the issue of event order may result in a long delay and can be a problem if a slow event-listener exists, because event-handling and drawing methods may be executed in the same thread. Perhaps a user can tolerate a delay for a short period, but if it occurs twice due to a former overridden event, it is usually not acceptable.

Floor control is the technique that deals with the issue of event order. It coordinates concurrent use of shared resources and data in a collaborative computing environment. Methods of floor control have been investigated in previous research [15][16]. These methods also can be applied in the design of a Java collaborative computing environment.

4.4 The issue of event duplicates

When a collaborative container of JavaBeans receives a broadcasted event, and then rebroadcasts it, endless looping events may occur. This is referred as the event duplicate issue.

To support collaboration transparency, a collaborative container of JavaBeans must implement many listener interfaces of JavaBeans and register itself to be one of the observers of those JavaBean objects. If the observer (i.e. the collaborative container) of high-level events does not know whether or not a high-level event comes from the network, but simply broadcasts the event in its event-handling method, the event duplicate issue exists.

The event handlers must know whether or not a high-level event comes from the network in order not to rebroadcast the loop-back event. Moreover, to avoid endless looping events, the event-handling method must temporarily remove some listeners that will produce another event broadcast.

As shown in Table 1, to avoid another event broadcast, if the system broadcasting service calls the method *setStates* after it receives an event from the network. The *setStates* method must first call the *disableCollaboration* method and finally call the *enableCollaboration* method to temporarily remove the AdjustmentListener object.

Table 1. An example of listener for event aggregation

```
class ScrollPaneListener
implements Runnable, AdjustmentListener{
  private Thread thread = null;
  // Constructor
  public ScrollPaneListener(){
    this.thread = new Thread(this);
    this.thread.start();
    this.enableCollaboration();
  }
  private synchronized void Broadcast(){….}
  public void enableCollaboration(){….}
  public void disableCollaboration(){….}

  // implement
  // java.awt.event.AdjustmentListener
  public synchronized void adjustmentValueChanged
            (AdjustmentEvent e) {
    // DO: event aggregation to avoid
    //    the waste of bandwidth
    notify();
  }

  public void setStates(Object[] tas){
    // avoid endless looping event
    this.disableCollaboration();
    // DO: set the states of ScrollPane
    //    from the broadcasted event
    this.enableCollaboration();
  }
```

```
  // Balking Algorithm
  public void run(){
    while(true)
        this.WAIT();
  }
   // Using Balking design pattern here
  private synchronized void WAIT(){
    long millis1 = 0L, millis2 = 0L;

    try {
      wait();
    } catch (InterruptedException ex){ }

    while (true) {
      try {
       millis1 =
              System.currentTimeMillis();
       wait(500L);
       millis2 =
              System.currentTimeMillis() - millis1;
       if (millis2 >= 500L)
         break;
      }catch (InterruptedException ex){}
    }
        // avoid endless looping event
    this.disableCollaboration();
    this.Broadcast();
    this.enableCollaboration();
  }
}
```

4.5 The issue of event conflict

The event conflict issue is similar to the event loss issue. It occurs when some events are dependent and are expected to be processed in a specific order. Moreover, if it is necessary to determine if a precondition is satisfied before processing a specific event, or taking an action, the event conflict issue may also occur.

Because Java event-handling methods are executed in a single thread, it is not certain that the event-handling for the former event will be completed before the later event. For example, a *Paste* action must wait for the completion of a *Copy* action, or it must check if a *Copy* action is ever performed. To avoid the NullPointerException, a *Paste* action will do nothing if the precondition, i.e. the existence of the object instance to be pasted, is "false".

The main difference between the event conflict issue and the event loss issue is that the former occurs when a sequence of event actions are dependent and must be processed in a specific order before some preconditions come into existence.

This issue can be solved by using a state check immediately after an event arrives. However, the programmer must describe and implement the task actions to be performed on event messages. Here we define a task as a unit of work that forms one logical step within a meaningful process. Each task must have a precondition that defines when the actions of a task can be executed. When the precondition of a task evaluates to "true" during the processing of an event message, the task is complete and the actions are dispatched. Although this solution works, it may degrade the system performance, due to status checks on event arrivals.

5 Conclusion

In this report, we have studied how a collaborative Java integrated development tool can be constructed to support the application areas in collaborative authoring and editing on Internet. Some problems we may encounter and possible solutions we can adopt were discussed in the i-Code system, a Java integrated development tool used to collaboratively build Java applications for collaborative computing. We have also presented issues of event broadcasting in the design of Java collaborative computing environments that support collaboration transparency. From our experience of Java collaborative computing, the issues of event loss, replay, event order, event duplicates, and event conflict appear frequently. However, they can be avoided by way of careful design in implementation.

Acknowledgments

The authors wish to acknowledge the very useful feedback provided during discussions with the Web-Based Collaboratory Research Group at IIS, Academia Sinica. 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] Kevin L. Mills, Introduction to the Electronic Symposium on Computer-Supported Cooperative Work, *ACM Computing Surveys (CSUR)*, Vol. 31, No. 2, June 1999, pp. 105-121.

[2] M. C. Dorneich, A system design framework-driven implementation of a learning collaboratory, *IEEE Transactions on Systems, Man and Cybernetics,* 32(2): 200-213, March 2002.

[3] H. Abdel-Wahab, B. Kvande, O. Kim and J. P. Favreau, An Internet collaborative environment for sharing Java applications, In *Proc. of the Sixth IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems*, Oct. 29-31, 1997, pp.112-117.

[4] B. Ionescu, J. Binder and D.Ionescu, A distributed architecture for collaborative applications, In *Proc. IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, Aug. 22-24, 1999, pp.525-529.

[5] The Java web site. http://www.java.sun.com

[6] SHARETONE project. http://webcollab.iis.sinica.edu.tw/SHARETONE/

[7] R. Iqbal, A. James and R. Gatward, A framework for integration of CSCW, In *Proc. of the 7th International Conference on Computer Supported Cooperative Work in Design*, Sept. 25-27, 2002, pp.43-48.

[8] M. Mansouri-Samani and M. Sloman, A Configurable Event Service for Distributed Systems, In *Proc. of the Third International Conference on Configurable Distributed Systems*, May 6-8, 1996, pp. 210-217.

[9] Marsic and B. Dorohonceanu, An Application Framework for Synchronous Collaboration using Java Beans, In *Proc. of the Hawai'i International Conference On System Sciences*, January 5-8, 1999, Maui, Hawaii.

[10] W. Sun, B. S. Lee and C. K. Yeo, JMS: a flexible collaborative environment, In *Proc. Of Internet Workshop (IWS 99),* Feb. 18-20, 1999, Osaka Japan, pp.195-202.

[11] G. Cugola, E. D. Nitto, and A. Fuggetta, Exploiting an event-based infrastructure to develop complex distributed systems, In *Proc. of the 1998 (20th) International Conference on Software Engineering*, April 19-25, 1998, Kyoto, Japan, pp. 261-270.

[12] G. Bricconi, E. Tracanella, and E. D. Nitto, Issues in Analyzing the Behavior of Event Dispatching Systems, In *Proc. of the Tenth International Workshop on Software Specification and Design (IWSSD'00)*, 2000, pp. 95-103.

[13] K. Richter and R. Ernst, Event Model Interfaces for Heterogeneous System Analysis, In *Proc. of the 2002 Design, Automation and Test in Europe Conference and Exhibition (DATE'02)*, pp. 506-513.

[14] M. Grand, *Patterns in Java Volume 1: A Catalog of Reusable Design Patterns Illustrated with UML*, 2nd Edition, John Wiley & Sons, 2002, ISBN: 0471227293.

[15] J. Boyd, Floor Control Policies in Multi-User Applications, In *Proc. of INTERACT '93 and CHI '93 Conference Companion on Human Factors and Computing Systems*, April 24-29, 1993, Amsterdam, The Netherlands, pp. 107-108.

[16] M. Chen, Achieving Effective Floor Control with a Low-Bandwidth Gesture-Sensitive Videoconferencing System, In *Proc. of the 10th ACM International Conference on Multimedia*, December 1-6, 2002, Juan-les-Pins, France, pp. 476-483.

[17] NetBeans.org.

http://www.netbeans.org/
[18] Eclipse.org Main Page.
http://www.eclipse.org/
[19] NetBeans Plugin Catalogue. http://www.netbeans.org/catalogue/index.html
[20] Eclipse plugin resource center and marketplace for Eclipse and Plugin Ecosystem.
http://www.eclipseplugincentral.com/
[21] DynamicJava.
http://koala.ilog.fr/djava/