

# **JACIE — an authoring language for WWW-based collaborative applications**

Abdul S. Haji-Ismail, Min Chen and Phil W. Grant

Department of Computer Science, University of Wales Swansea,  
Singleton Park, Swansea SA2 8PP, United Kingdom

csabdul@swansea.ac.uk, m.chen@swansea.ac.uk, p.w.grant@swansea.ac.uk

## **Abstract**

With continuous acceptance of WWW as a de facto standard for human-computer interaction and human-human communication, it is desirable to develop collaborative applications under the framework of WWW. In this paper, we present a new script language called JACIE, which is designed to support rapid implementation of a wide range of multimedia collaborative applications. In particular, we highlight the necessity to support the management of interaction and communication activities in collaborative applications, and describe how JACIE facilitates such support through the concepts of channels and interaction protocols. JACIE also features a template-based programming style, a single program for both client and server, and platform-independence by using Java as the target language.

## **Keywords**

script language; authoring tool; collaborative applications; Java

## **1. Introduction**

As a world-wide electronic medium, the World Wide Web (WWW) has become a de facto standard for human-computer interaction and human-human communication. Despite the fact that the primary use of WWW has been to distribute multimedia information, a number of applications have demonstrated that it is equally feasible and effective for WWW to facilitate collaborative activities [Woo94, Fri94].

However, most existing collaborative applications were not developed under the framework of WWW, and the development of new applications is retarded by expensive development costs, lengthy development period, and shortage of expert knowledge and programming skills. Although there are many powerful general purpose development environments (such as Microsoft Visual J++, Borland JBuilder, IBM VisualAge, Supercede Java and Symantec Visual Cafe), to develop collaborative applications with such an environment, it is still essential for a developer to have sufficient knowledge of network programming (such as client/server computing, socket programming, Java Remote Method Invocations or Common Object Request Broker Architecture) [Sun98a, Sun98b]. The resurgence of the popularity of script languages such as Perl, VBScript and JavaScript further highlighted the importance of programming efficiency in the development of Internet-based applications [Kha97].

In recent years, efforts have also been made to develop special purpose development environments for collaborative applications. They include DIVE [Car93, Hag96], AC3D [CSEG98], SOL [CSEG98], Yarn Web [Woo94], VRML-extension [Bro97], JCE [Abd96], Habanero [NCSA98], TANGO [Bec98], Eventware [Eve98] and Hesse [Pra98]. Some such as DIVE are platform dependent. Many target at a relatively small class of applications. For instance, DIVE, AC3D and Broll's VRML-extension are designed mainly for 3D virtual environments, Yarn Web for WWW-based meetings, Habanero and SOL for object sharing, and JCE for application sharing. All of them are in the form of software frameworks or embedded software libraries that remove programming burden only in a limited way.

In this paper, we present a new script language called JACIE (Java-based Authoring language for Collaborative Interactive Environments). JACIE has been designed to support rapid implementation of a wide range of network-based collaborative applications. In particular, it facilitates the management of interaction and communication through simple communication primitives such as channels and interaction protocols, hence hiding much network programming from programmers. It also features a template-based programming style, a single program for both client and server, and platform-independence by using Java as the target language.

In Section 2, we will give an overview of collaborative applications and their system requirements. In Section 3, the design principles of JACIE and its main functional features will be described. This is followed by a discussion in Section 4, focusing on the management of interaction and communication in JACIE. In Section 5, we will describe the software architecture on which the implementation is based. We will conclude this paper with some observations and our plan for further development.

## **2. Collaborative Applications**

The fundamental objective of most collaborative applications is to facilitate collaborative activities. The most important elements of such an application are its multimedia contents, users' interaction with computers, and their communications with other users. Although the past decade witnessed enormous efforts in the development of virtual environments and embodiment techniques [Gre95, Gre96], it is generally recognized that the effort for making the virtual world look real must not undermine the effectiveness of the elements of multimedia, interaction and communication.

The desire to support collaboration over the Internet has led to a number of substantial projects. One of the most acknowledged research projects is DIVE [Car93], an Internet-based multi-user system that allows remote participants to meet and interact with each other in a virtual 3D space. Other major European projects, including COVEN [Nor98], DEVRL [Sla96], COMIC, VirtuOsi [Ben94], MASSIVE [Gre95] and MASSIVE-2 [Gre96]. Many of them have focused on 3D virtual environments, while others have attempted to address a wide range of issues related to virtual environments, such as awareness, scalability and human factors. The development of these projects has been mostly carried out by highly skilled programmers with a common aim to drive the technology to its limits.

Parallel to these developments, there are also a number of relatively small projects targeted at specific application areas. Such a collaborative application typically provides a virtual environment for users who are geographically separated but wish to carry out some practical

work together, such as visualizing some data [Ran95, Dis95, Duc98, Bro98], having a meeting [Sum98, Gas94, Gin95], editing a document, giving a distance learning course [Chr95, Sho95], etc.

Most of these collaborative applications fall into the following six groups:

- (a) collaborative work environments (for conducting collaborative work such as engineering design, visualization, documentation, and so on);
- (b) meetings, seminars and conferences over the Internet;
- (c) simulation of face-to-face contacts where visual quality is critical (such as recruitment interviews, medical diagnoses and remote surgical operations);
- (d) distance learning environments (for providing course materials, holding a tutorial, carrying out a team project, and conducting an examination);
- (e) networked computer games;
- (f) leisure and entertainment (including 3D navigation and virtual embodiment).

Modern data communication technology enables users in a collaborative environment to interact through a variety of communication media. As summarized in Table 1, different groups of applications often focus on a different set of media. The first four media types are reasonably well supported by general purpose systems such as Microsoft NetMeeting [Sum98], or application software of a distributed nature such as CAVE [Dis95]. In contrast, implementation of *shared interactive canvas* are generally application-specific and usually require considerable knowledge of system and network programming. This is reflected by the fact that applications in groups (a), (d) and (e) are not very well supported by the development tools that are currently available. Although the implementation of 3D virtual space require a range of modeling and programming skills in addition to powerful graphics hardware, Table 1 also indicates that there is limited practical interest in 3D virtual space except for applications in group (e).

**Table 1: The role of communication media in different collaborative applications.**

media	(a)	(b)	(c)	(d)	(e)	(f)
text-based online chat	✓✓✓	✓	✓✓	✓✓✓	✓	✓
voice conferencing	✓✓	✓✓✓	✓✓✓	✓✓	✓	✓
video conferencing	✓	✓✓✓	✓✓✓	✓✓	✓	✓
shared applications	✓✓✓	✓✓✓	✓✓	✓✓✓	✓	✓
shared interactive canvas	✓✓✓	✓	✓	✓✓✓	✓✓✓	✓✓
3D virtual space and embodiment	✓	✓	✓	✓	✓✓	✓✓✓

✓✓✓ : necessary

✓✓ : desirable

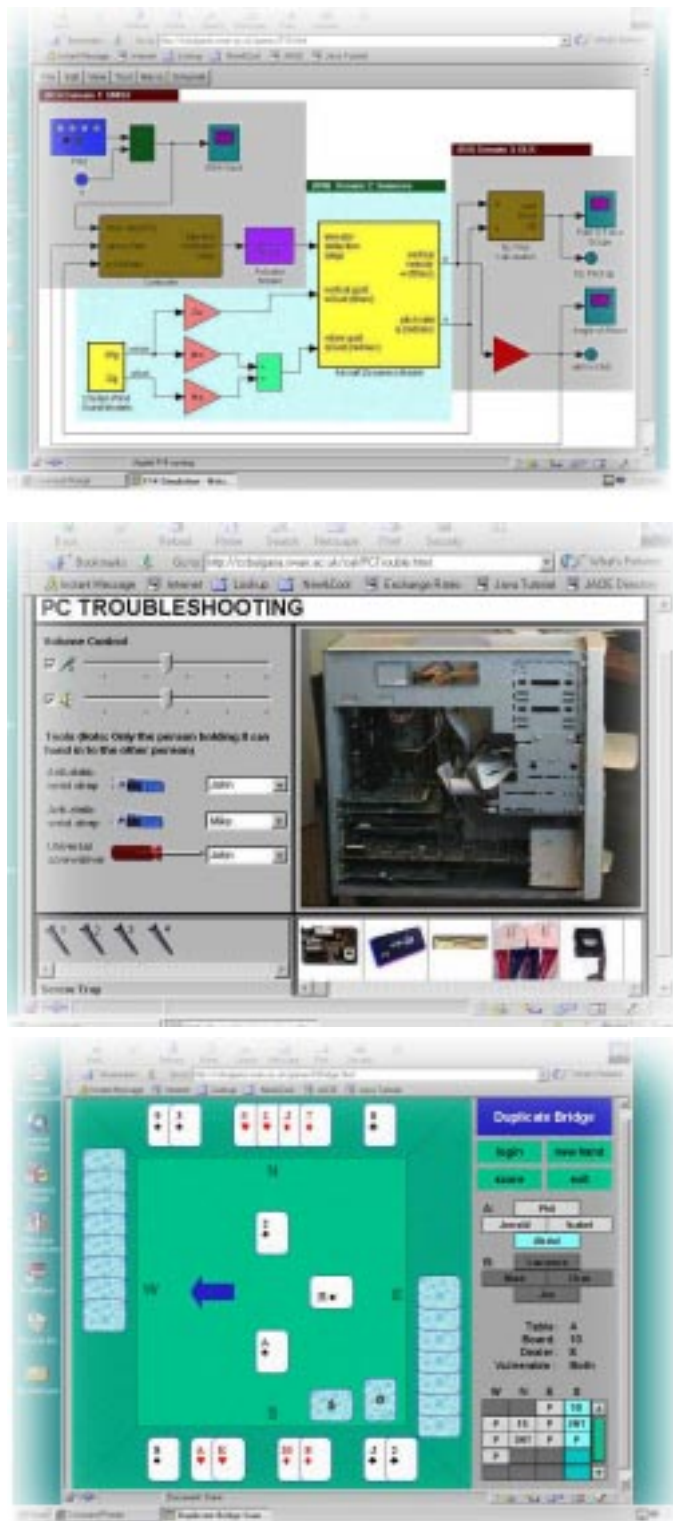
✓ : occasionally useful

Consider three typical collaborative applications in groups (a), (d) and (e) respectively. As shown in Figure 1, they are a CAD environment for designing and simulating a dynamic system, a teamwork environment for a PC maintenance course, and a networked bridge game. The main difficulty involved in developing such applications is not the complexity in implementing the graphical realism, but that of multimedia interactions which cannot be adequately handled by the traditional event-driven programming.

Event-driven programming is one of the most fundamental techniques for processing information created by a user interactively. With this technique, a user's inputs are mapped

onto typed events (e.g. menu selection, window resize, etc.), which are buffered in an event queue. Each event is then dispatched, in the order of its arrival time, to a specific handler according to its type. The logic flow of the application is constructed based on the assumption that the order of events in the queue is the same as that of the user's inputs. However, this assumption is no longer valid in a collaborative environment, where events that influence a graphical display come from remote users as well as the local user. The ordering of events cannot be determined simply based on the arrival time at the local event queue. Because of the concurrency available in such an environment and the delay in network communication, a user may easily generate some inputs that would become invalid upon their arrival at a remote site. Event validation thereby becomes a critical step in processing events. Furthermore, the task to ensure the consistency between a local graphical display and all remote displays is not trivial. There are also many complex implementation issues, such as global clock vs. local clock and distributed queuing vs. centralised queuing [Cou94].

Furthermore, the management of these events differs depending on individual applications. For instance, the collaborative CAD environment may divide a diagram into several access domains, each of which can be modified by only one user at a time. On the other hand, the environment has to maintain consistency among all visual displays for the diagram and its simulation results. In contrast, the teamwork environment may be designed to develop good communication skills and leadership quality of its users through collaborative activities. All users may be allowed to access every part of the computer, and it is up to the users to coordinate their activities effectively and safely. The bridge game is governed by a set of rules, and all actions must follow a predefined order and be restricted to their corresponding access domains. To implement code for handling correctly events generated by interaction



**Figure 1. Examples of collaborative applications.**

and communication, a programmer would normally have to program at the system level and deal with low level communication protocols. It is this difficulty that provides the initial motivation for the development of JACIE.

### 3. JACIE: Design Principles and Main Features

#### 3.1 Overview

JACIE (Java-based Authoring language for Collaborative Interactive Environments) is a script language, and it has been designed to support rapid implementation of a wide range of network-based collaborative applications. In particular, it targets a collection of applications for which the existing programming tools would incur expensive development costs. The key design principles of JACIE are therefore *special purpose* and *programming efficiency*. The main functional features of JACIE, which reflects its design principles are summarized in the following two sections.

#### 3.2 Special Purpose

- *Typical applications* — These include WWW-based groupware (e.g. a distributed timetabling system), courseware (in particular teamwork-based courseware) and games (e.g. board games and card games) These applications commonly feature real-time collaborative activities, shared working canvas, controlled access domain and structured communication. It is often very difficult for applications in these areas to generate much commercial profit, and thereby difficult to attract resources to develop them in the first place. It is intended for JACIE to provide a software development tool that enables such applications to be developed at a very low development cost, within a short development period, and by not-necessarily-experienced programmers.
- *Images and carefully-selected graphics functions* — From Figure 1, we can observe that images and simple drawings are likely to dominate the graphics requirements of these three typical applications. JACIE assumes that the graphics in most of these applications involve mainly images and simple 2D graphics drawings, since the use of complex 3D graphics modeling in such applications would not be cost-effective in terms of expertise and effort. In fact the very same assumption is made by most hypermedia authoring tools except for VRML that was designed for 3D graphics. This assumption allows JACIE to reduce the complexity of its display functions, a desirable feature for any script language.

In order to facilitate effective canvas management and device-independent display, JACIE places its emphasis on a set of grid-based operations to support the display of, and interaction with, images and graphics primitives including line, text and rectangles.

- **Structured communications** — On the other hand, the majority of such applications would require a structured communication, that is, unlike our daily face-to-face conversation, the order of communication activities is a critical part of an application. Different applications may require different communication protocols. Some may be centrally controlled by a control program (server) or a specific user (master client), while others may distribute the control to users (clients). Some may impose strict order over any communication activities, while others may allow arbitrary communication to take place.

Protocol control is the weakness of most existing programming languages and development tools. The implementation of the protocols is often not a trivial task during

the development of an application involving structured communication. It is intended that JACIE addresses this problem by providing a set of built-in protocols that can be easily utilized by JACIE programmers.

### 3.3 Programming Efficiency

- **Script language** — At the initial design stage of JACIE, several considerations were taken into account as to whether JACIE should be a script language or a software library. The decision was made based on the following factors:
  - (a) A software library will not solve entirely the problems faced by the users and developers of the application areas concerned. The knowledge of the programming language, where a software library would be embedded, will still hold the key to the effort and cost required for developing such applications.
  - (b) The recent development in WWW-based technologies has demonstrated the effectiveness and popularity of script languages. In most cases, the development of a script language involves a software library in the target language to support the compilation of common functions. It is easy for a script language to provide a software library as a side product, but not vice versa.
- **“Single program” for server and clients** — With almost all WWW-based programming languages and software development tools, separate programs have to be written for server and client. To ease the programming effort, JACIE uses a single program to specify both server and client. It is the compiler’s job to generate server and client programs to run on different computers.
- **Channels** — In an application, communications may take place in a number of different forms of media. JACIE introduces the concept of *channels* for specifying these media. The built-in channels in JACIE include *canvas*, *message*, *chat*, *voice*, *video* and *whiteboard*. This enables the developer of an application to select a set of channels appropriate to its requirements and hardware constraints. By providing a set of high-level communication primitives, JACIE successfully hides from its programmers the complication in dealing with low level communication mechanism and network programming.
- **Interaction Protocols** — An *interaction protocol* defines the rules that govern the means of interactions between users in an collaborative environment, and is used to coordinate the input from users and the display on a channel. (The term “interaction protocol” is used specifically to avoid the confusion with low-level communication protocols). In a single user environment the order of incoming events is consistent with that of the user’s inputs. As discussed in Section 2, this assumption is no longer valid in a networked environment. In order to ease the difficulties in validating events, JACIE provides a collection of built-in protocols for managing a variety of the interactions and communications activities in collaborative applications. By associating a protocol to a certain group of activities programmed in JACIE, programmers no longer need to deal with the order of incoming events directly.
- **Session control** — JACIE employs a template-based programming style for its main program body. It divides each program into a set of standard components, as shown in Figure 2. A syntax-directed editor could provide this template for the user.

<pre> JACIE {    applet name X;   appletlauncher imagebutton "X.gif";   configuration {...}   messages {...}    client implementation {     declaration {...}     on canvas {...}     on session start {...}     on session {...}     on session end {...}   }    server implementation {     declaration {...}     on server start {...}     on session start {...}     on session {...}     on session end {...}     on server end {...}   } } </pre>	<table border="1"> <tr><td><b>SYSTEM CONFIGURATION</b></td></tr> <tr><td>Define program name.</td></tr> <tr><td>Define launcher (optional).</td></tr> <tr><td>Specify networking parameters, channels, protocols, etc.</td></tr> <tr><td>Declare attributes of messages.</td></tr> <tr><td> </td></tr> <tr><td><b>CLIENT BODY</b></td></tr> <tr><td>Declare constants, variables and methods.</td></tr> <tr><td>Initialize canvas such as drawing a background image.</td></tr> <tr><td>Perform processes upon established connection.</td></tr> <tr><td>Main client session control.</td></tr> <tr><td>Perform processes upon session termination.</td></tr> <tr><td> </td></tr> <tr><td><b>SERVER BODY</b></td></tr> <tr><td>Declaration of constants, variables and methods.</td></tr> <tr><td>Initialize server processes.</td></tr> <tr><td>Perform processes upon each user's connection.</td></tr> <tr><td>Main server session control.</td></tr> <tr><td>Perform processes upon user's session termination.</td></tr> <tr><td>Housekeeping processes upon server termination.</td></tr> <tr><td> </td></tr> <tr><td> </td></tr> <tr><td> </td></tr> </table>	<b>SYSTEM CONFIGURATION</b>	Define program name.	Define launcher (optional).	Specify networking parameters, channels, protocols, etc.	Declare attributes of messages.		<b>CLIENT BODY</b>	Declare constants, variables and methods.	Initialize canvas such as drawing a background image.	Perform processes upon established connection.	Main client session control.	Perform processes upon session termination.		<b>SERVER BODY</b>	Declaration of constants, variables and methods.	Initialize server processes.	Perform processes upon each user's connection.	Main server session control.	Perform processes upon user's session termination.	Housekeeping processes upon server termination.			
<b>SYSTEM CONFIGURATION</b>																								
Define program name.																								
Define launcher (optional).																								
Specify networking parameters, channels, protocols, etc.																								
Declare attributes of messages.																								
<b>CLIENT BODY</b>																								
Declare constants, variables and methods.																								
Initialize canvas such as drawing a background image.																								
Perform processes upon established connection.																								
Main client session control.																								
Perform processes upon session termination.																								
<b>SERVER BODY</b>																								
Declaration of constants, variables and methods.																								
Initialize server processes.																								
Perform processes upon each user's connection.																								
Main server session control.																								
Perform processes upon user's session termination.																								
Housekeeping processes upon server termination.																								

**Figure 2. A set of standard JACIE components.**

The most interaction and communication control will be in the “on session {...}” components in both server and client bodies. The programming of the rest of the components is considered to be straightforward.

- **Interfacing with Java** — No doubt, some applications will require functions, such as complicated graphics operations, which JACIE cannot supply. The target language Java naturally becomes convenient in such cases. JACIE allows the inclusion of Java code as part of a JACIE program (Figure 3). This enables experienced programmers to utilize Java for the implementation of complex code segments, for example, complicated graphics, numerical computation, or logic control. All simple variables in JACIE can be modified in Java and have a prefix “JACIE\_”. More complicated variables such as channels can also be accessed in Java in a predefined manner, typically by calling appropriate methods defined in the corresponding Java class.

<pre> ... x[] = {100, 200, 250, 50, 100}; y[] = {50, 50, 200, 200, 50}; Java {   Public void paint (Graphics g) {     g.setColor(Color.blue);     g.fillPolygon(JACIE_x, JACIE_y, 5);   } ... </pre>	<table border="1"> <tr><td><b>JACIE code</b></td></tr> <tr><td>JACIE assignments.</td></tr> <tr><td> </td></tr> <tr><td><b>Java code</b></td></tr> <tr><td>calling methods in a Java class.</td></tr> <tr><td> </td></tr> <tr><td><b>JACIE code</b></td></tr> </table>	<b>JACIE code</b>	JACIE assignments.		<b>Java code</b>	calling methods in a Java class.		<b>JACIE code</b>
<b>JACIE code</b>								
JACIE assignments.								
<b>Java code</b>								
calling methods in a Java class.								
<b>JACIE code</b>								

**Figure 3. A Java code segment in a JACIE program.**

## 4. JACIE: Interactions and Communications

### 4.1 Built-in Channels

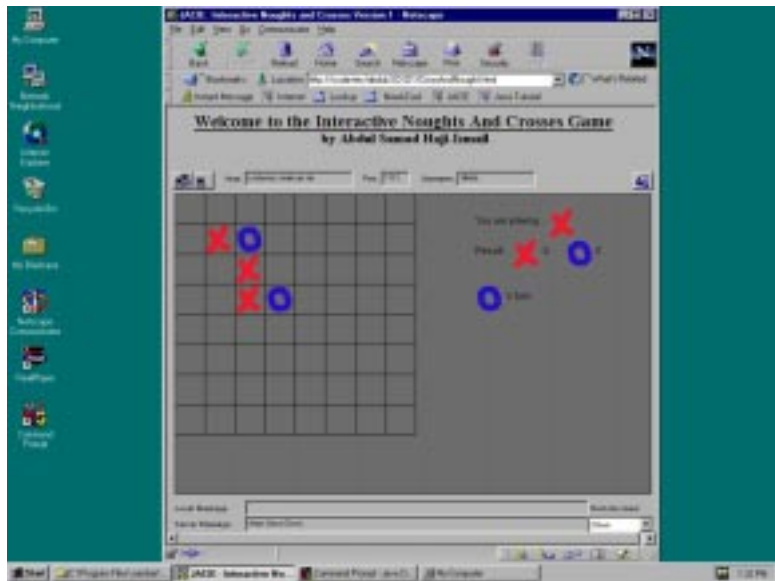
As mentioned in Section 3, there are six built-in channels in JACIE, namely *canvas*, *message*, *chat*, *voice*, *video* and *whiteboard*.

- **Canvas channel** — It is a shared workspace for collaborative activities, and in most cases it is the main focus of a collaborative application. It allows the display, and modification, of images and graphics drawings according to a pre-defined protocol. A canvas is initialized by an “`on canvas {...}`” component (Figure 2), which normally draws a background that remains unchanged throughout a session. Display of images and graphics primitives takes place within the “`on session {...}`” component of a client in an event-driven manner. More than one canvas channel can be activated in an application.
- **Message channel** — It is mainly used for displaying messages from the server and other clients. Multiple message channels are often used to form a more complex channel or display. Each message channel is associated with a number of attributes, such as type and length, which are declared in “`messages {...}`” component (Figure 2). One of these attributes specifies whether or not a message can be *captured* by the programmer. In the program body of a client or a server, an event handler can be defined for any message that can be captured. The arrival of such a message will activate the event handler which allows the message to be processed in a manner specific to the application. A message channel can be either visible or invisible, and the latter is normally for communicating control messages. There is a set of pre-defined message channels in JACIE.
- **Chat channel** — It is used for online text-based communications. In implementation, a chat channel is composed of several text-based message channels, one for each client. Messages in a chat channel cannot be captured online by a user-definable event handler.
- **Voice channel** — It is used for online voice-based communications. Voice information from different clients are combined at the server which then broadcasts to all clients. A client may exercise volume controls locally over the outgoing voice as well as the combined incoming voice. Appropriate interaction protocols may be used to provide turn control to a voice channel. However, no voice information can be captured by a user-definable event handler.
- **Video channel** — It is used for online video-based communications. No video information can be captured by a user-definable event handler. A voice channel is normally used in conjunction with one or more video channels.
- **Whiteboard channel** — It is used for arbitrary drawings on a dedicated canvas.

A collaborative application may open more than one channel at a time, and may customize the function of each channel by specifying its communication parameters and interaction protocols as detailed in 4.3.

## 4.2 A Study on Interactions

An *interaction protocol* defines the rules that govern the means of interactions among users in a collaborative environment. Different applications may require different protocols. In order to identify the interaction protocols that may possibly be required by a collaborative application, a collection of networked *noughts and crosses* (tic-tac-toe) games were implemented in Java prior to the development of JACIE. All games are designed to run across the Internet, and users on different computers may interact through WWW browsers (Figure 4). Each game implements a different rule that simulates a possible interaction protocol, and Table 3 lists the main features of a selection of these games.



**Figure 4. A WWW-based noughts and crosses game.**

**Table 3. Different versions of noughts and crosses.**

Name	Players	Board	Turn Control	Cell Control	Winning
<b>Traditional Game</b>	2 players	3x3 cells	Each player places his/her symbols in turn.	One symbol in an empty cell each time.	The first one obtains 3 of his/her symbols in a line wins the game.
<b>Generalized Game</b>	2 players	8x8 cells	Each player places his/her symbols in turn.	One symbol in an empty cell each time.	The first one obtains 3 of his/her symbols in a line wins the game.
<b>Speed Fight</b>	2 players	8x8 cells	Players place their symbols as fast as they can. No turn control.	One symbol in an empty cell.	The first one obtains 5 of his/her symbols in a line wins the game
<b>Vicious Fight</b>	2 players	8x8 cells	Player place their symbols in any cells as fast as they can. No turn control.	An existing symbol can be replaced.	The first one obtains 8 of his/her symbols in a line wins the game.
<b>Gentlemen's Fight</b>	2 players	8x8 cells	Each player signals the other player if he would like to play, and cannot place his symbols until permission is received from the other player (or the server).	One symbol in an empty cell each time.	The first one obtains 5 of his symbols in a line wins the game.
<b>Dictator's Entertainment</b>	2 players	8x8 cells	Each player must receive a signal from the dictator (the server) before placing a symbol, the dictator randomly selects a player each time using a random number generator.	One symbol in an empty cell each time.	The first one obtains 5 of his/her symbols in a line wins the game.
<b>Group Game</b>	2 groups, 2 players each group	8x8 cells	Each player places his/her symbols in turn.	One symbol in an empty cell each time.	The first group obtains 3 his/her symbols in a line wins the game.

There are other versions of noughts and crosses games, involving grouped or ungrouped multiple players. Most of the games require structured interactions, as the order of events is a critical part of a game.

### 4.3 Built-in Interaction Protocols

There are five basic built-in interaction protocols as detailed in Table 4, where we utilize the technical terms used for specifying medium access control methods in low level network communication [Hal95].

**Table 4. Protocols for multimedia communications.**

Protocol	Description	Example
unstructured	Information is passed onto the other end of the channel without turn control, unprocessed, and displayed in an appropriate medium and in its corresponding raw form.	white-board
contention	Information is passed onto the other end of the channel without turn control. It can be processed, and if required mapped onto an appropriate display form.	the speed fight version of noughts-and-crosses game
token-pass	In a simple token-pass protocol, a control token is circulated among clients in the round-robin manner. A client may activate one communication activity after receiving a token, and must forward the token immediately after. Complicated protocols may involve multiple tokens, time-out tokens and hierarchical tokens (i.e., token among groups and token among group members).	traditional noughts-and-crosses game
reservation	A request must be first made by a client (not necessarily by a user and can be coded as an action after an activity/event). The client may activate a series of communication activities after receiving the control.	the gentlemen version of noughts-and-crosses game
centralised	Events are generated by a server (or a master client via a server) to activate communication activities of clients in a controlled manner.	The dictator version of noughts-and-crosses game

Recall our discussion in 4.1, a channel may be customized by defining its traffic mode parameter and interaction protocol. The traffic mode divides communication activities into two major categories, namely *simplex* and *duplex*. Half-duplex traffic can be achieved through structured interaction protocols, that is, contention, token-pass, reservation and centralised. The communication modes and interaction protocols that are available to each built-in channel are listed in Table 5. Those traffic modes in italic have to be defined through the duplex mode in conjunction with appropriate interaction protocols or other means (such as volume control for voice). Although a message channel can be shared by a client and a server, duplex traffics are disabled in JACIE's implementation as messages should not be inter-mingled.

**Table 5. Traffic modes and interaction protocols available to each channel.**

Channel	Traffic Mode	Interaction Protocol
<b>canvas</b>	<i>simplex</i> , <i>half-duplex</i> or duplex	unstructured, contention, token-pass, reservation, centralised
<b>message</b>	<i>simplex</i> or <i>half-duplex</i>	unstructured, contention, token-pass, reservation, centralised
<b>chat</b>	duplex ( <i>simplex message channels</i> )	unstructured
<b>voice</b>	<i>simplex</i> , <i>half-duplex</i> or duplex	unstructured, contention, token-pass, reservation, centralised
<b>video</b>	<i>simplex</i>	unstructured
<b>white-board</b>	<i>simplex</i> or duplex	unstructured

## 5. JACIE: Software Architecture

JACIE utilize WWW as the main vehicle to deliver a collaborative application. Its general functionality, including its language components, session control, image and graphics functions, event management and interaction protocols, are mapped onto Java programs. Some of the JACIE channels (including message, chat and canvas) are implemented through a set of purposely designed Java classes, while the others (including voice, video and whiteboard) are built upon Microsoft NetMeeting [Sum98]. Figure 5 illustrates the software architecture of JACIE.

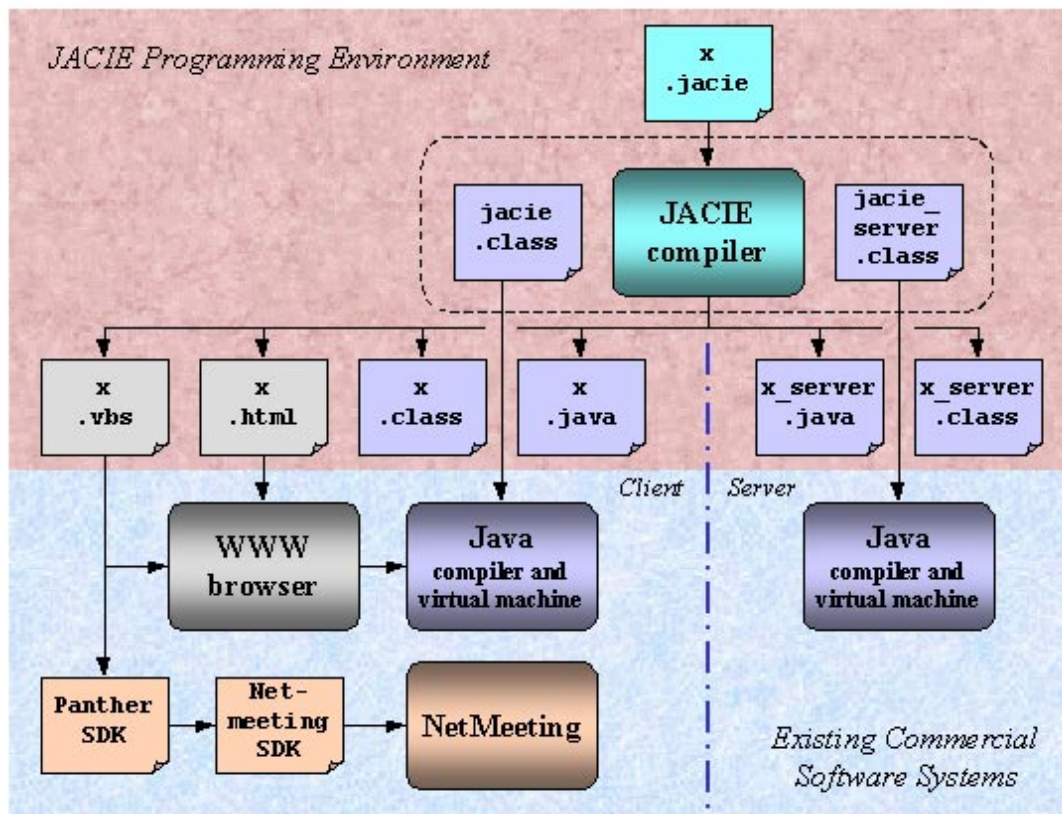


Figure 5. Software Architecture of JACIE.

Consider an arbitrary collaborative applications written as a JACIE program, for instance, called `x.jacie`. The JACIE compiler generates the followings:

- `x.html` — a simple HTML file for launching the application;
- `x.java` — a Java applet that defines the main structure of the client program, and is to be compiled into Java bytecode for running on the Java Virtual Machine;
- `x_*.class` — a set of application specific classes in Java for supporting client operations in the individual components of `x.jacie`;
- `x_server.java` — a Java program that defines the main structure of the server program, and is to be compiled into Java bytecode for running on the Java Virtual Machine;
- `x_server_*.class` — a set of application specific classes in Java for supporting server operations in the individual components of `x.jacie`;

- `x.vbs` — a program which is written in VBScript for supporting voice, video and whiteboard channels through NetMeeting, and which is generated only if there are such channels defined in `x.jacie`.

With JACIE, there are also two sets of generic classes predefined in Java. They are not generated during JACIE compilation, and instead they are automatically integrated into the client applet and server programs respectively during Java compilation. The client applet can also be launched on its own using Java applet viewer.

There is a significant amount of coding efficiency achieved through JACIE. We carried out a comparison between a JACIE “hello” program (with a canvas channel and two message channels) and the equivalent Java programs [Haj98]. The size of the JACIE program is 4% of that of the Java programs in terms of the number of lines, and 2.4% in terms of the numbers of characters (without space). The size of the actual code (in Java) compiled from such a JACIE program is even larger than the hand-crafted Java code used in this comparison.

Figure 6 shows a collaborative scrabble game written in JACIE, which allows groups of players to play the game with a hierarchical token-based interaction protocol. The application uses a canvas channel with two built-in visible message channels for the main game board, and two separate chat channels, one for public announcement and the other for private communications within a group.

The example display in Figure 6 shows a game being played 2 groups with 2 players in each group. Each player is given 4 tiles randomly. Each group will get its turn in a round-robin fashion controlled by a *group token* (level one token) Through a private chat channel, players of the same group may inform each other the tiles in their hands and discuss their game strategy. The turn control within each group is managed by a *player token* (level two token) with a time-out function. The time-limit assigned to the player token discourages lengthy or

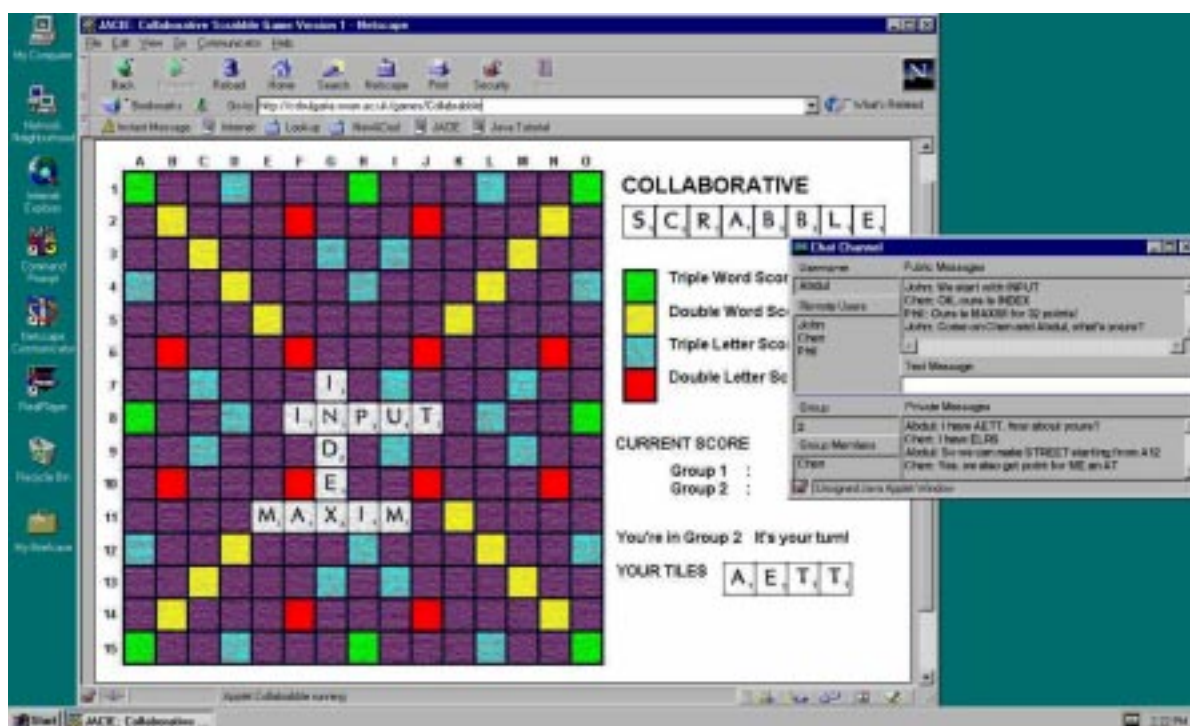


Figure 6. An example JACIE application

ineffective group discussions.

## 6. Conclusions

We have described a novel script language, JACIE, that is specifically designed for developing multimedia collaborative applications. By focusing on the management of interactions and communications, JACIE offers a practical solution to the programming difficulties involved in the implementation of these applications. Unlike most other development environments in the field, JACIE programmers need to script a single program for a collaborative application based on a well-defined template. Low-level networking programming and event management are hidden from the programmers. By building the main functionality of JACIE on the top of Java, we have achieved platform-independence to a certain extent. We believe that JACIE is well-suited for a wide range of collaborative applications under the framework of WWW.

Further work being undertaken on JACIE includes:

- (a) refining the JACIE grammar by utilizing some formal approaches, and improving the reliability and efficiency of JACIE compiler;
- (b) investigating the feasibility of using a standard approach such as Active-X for interfacing “foreign” software systems;
- (c) using JACIE to develop some practical applications of a considerable scale, including a collaborative training system.

## Acknowledgment

Authors wish to thank our colleague Mark Kiddell for his valuable technical assistance in areas of video and voice conferencing.

## References

- [Abd96] Abdel-Wahab et. al. “Using Java for multimedia collaborative applications”, *Proc. 3<sup>rd</sup> International Workshop on Protocols for Multimedia Systems (PROMS’96)*, October 1996, Madrid.
- [Bec97] L. Beca, F. G. Cheng, T. G. C., Jurga, Olszewski, M. K., Podgorny, and K. Walczak, “Web technologies for collaborative visualization and simulation”, *Proc. the 8<sup>th</sup> SIAM Conference On Parallel Processing*, Minneapolis, March 1997.
- [Ben94] S. D. Benford, J. Bowers, S. Gray, T. R. Rodden, M. Rygol, and V. Stanger, “The VirtuOsi project”, *Proc. VR’94 (London Virtual Reality Expo 1994)*, Feb. 1994, Meckler.
- [Bro97] W. Broll, “Distributed virtual reality for everyone - a framework for networked VR on the Internet”, *Proc. of the IEEE Virtual Reality Annual International Symposium 1997 (VRAIS’97)*, Albuquerque, NM, March 1-5, 1997, IEEE Computer Society Press.
- [Car93] C. Carlson and O. Hagsand, “DIVE: a platform for multi-user virtual environments”, *Computers and Graphics*, 17(6), 1993.
- [Chr95] E. Christiansen and L. Dirckinck-Holmfeld, “Making distance learning collaborative”, *Proc. Computer-Supported Collaborative Learning ’95*

- (CSCL '95), Indiana University, Bloomington, IN, October 17-20, 1995, Lawrence Erlbaum Associates, Inc.
- [CSEG98] *WWW page on Cooperative Systems Engineering Group*, <http://www.comp.lancs.ac.uk/computing/research/cseg/>, Nov. 1998.
- [Eve98] *WWW page on Eventware: Collaborative Software for a New Age*, <http://eventware.com/>, Nov. 1998.
- [Cou94] G. Coulouris, J. Dollimore and T. Kindberg, *Distributed Systems: Concepts and Design*, 2<sup>nd</sup> edition, Addison-Wesley 1994.
- [Dis95] T. L. Disz, M. E. papka, M. Pellegrino and R. Stevens, "Sharing visualization experiences among remote virtual environments", in M. Chen, P. Townsend and J. A. Vince (Eds), *High Performance Computing for Computer Graphics and Visualisation*, Springer-Verlag, London, 1995, pp.217-237.
- [Duc98] D. A. Duce, J. R. Gallop, I. J. Johnson, K. Robinson, C. D. Seelig and C. S. Cooper, "Distributed cooperative visualization – the MANICORAL approach", Proc. 16<sup>th</sup> Eurographics UK Conference, Leeds, UK, March 1998, pp.69-86.
- [Fri94] T. J. Frivold, R. E. Lang and M. W. Fong, "Extending WWW for synchronous collaboration", *Elec. Proc. Second World-Wide Web Conference '94*, Chicago, USA, October 1994. (also, <http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings/CSCW/frivold/frivold.html>, Nov. 1998)
- [Gas94] S. Gasner, S. Schooler, R. Frederick and V. Jacobson, "Multimedia teleconferencing on the Internet multicast backbone (MBONE)", *Proc. ACM Multimedia '94*, San Francisco, USA, November 1994.
- [Gin95] A. Ginsberg and S. Ahuja, "Automating envisionment of virtual meeting room histories", *Proc. ACM Multimedia '95*, San Francisco, USA, November 1995, pp.65-75.
- [Gre95] C. M. Greenhalgh and S. D. Benford, "MASSIVE: a virtual Reality System for teleconferencing", *ACM Transactions on Computer Human Interfaces (TOCHI)*, 2(3):239-261, Sept. 1995.
- [Gre96] C. M. Greenhalgh, *Dynamic Embodied Multicast Groups in MASSIVE-2*, Technical Report NOTTCS-TR-96-8, Department of Computer Science, University of Nottingham, UK, 1996.
- [Hag96] O. Hagsand, "Interactive multiuser VES in the DIVE System", *IEEE Multimedia Magazine*, 3(1), 1996.
- [Haj98] A. S. Haji-Ismail, *JACIE - Java-based Authoring language for Collaborative Interactive Environments*, Second Year Postgraduate Report, Department of Computer Science, University of Wales Swansea, September 1998.
- [Hal95] F Halsall, *Data Communications, Computer Networks and OSI*, Addison-Wesley, 1995.
- [Lov98] "Collaborative research within a sustainable community: interactive multi user VRML and visualization", Proc. 16<sup>th</sup> Eurographics UK Conference, Leeds, UK, March 1998, pp.53-68.
- [Kha97] R. Khare, "Scripting Languages: automating the Web" Editorial, *World-Wide Web Journal*, Vol II, Issue 2, Spring 1997.
- [NCSA98] *WWW page on NCSA Habanero*, <http://www.ncsa.uiuc.edu/SDG/Software/Habanero/HabaneroHome.html>, Nov. 1998.
- [Pra98] *WWW page on Hesse Collaborative Framework Overview*, <http://www.praxistech.com/hesse/hesse.html>, Nov. 98.
- [Ran95] D. Rantzau, U. Lang, R. Lang, H. Nebel, A. Wierse and R. Ruehle, "Collaborative and interactive visualization in a distributed high performance software environment", in M. Chen, P. Townsend and J. A. Vince (Eds), *High Performance*

*Computing for Computer Graphics and Visualisation*, Springer-Verlag, London, 1995, pp.207-215.

- [Sho96] P. G. Shotsberger, K. B. Smith and C. G. Spell (1996) “collaborative distance learning on the World Wide Web: would that look like?”, *Proc. Computer-Supported Collaborative Learning '95 (CSCL '95)*, Indiana University, Bloomington, IN, October 17-20, 1995, Lawrence Erlbaum Associates, Inc.
- [Sla96] M. Slater, M. Usoh, S. Benford, D. Snowdon, C. Brown, T. Rodden, G. Smith, and S. Wilbur, “Distributed extensible virtual reality laboratory (DEVRL)”, M. Goebel (Ed), *Proc. 3<sup>rd</sup> Eurographics Workshop on Virtual Environments*, Feb. 1996.
- [Sum98] B. Summers, *Official Microsoft Netmeeting Book*, Microsoft Press, 1998.
- [Sun98a] *WWW page on Java White Paper: The Java Language: An Overview*, <http://java.sun.com/docs/overviews/java/java-overview-1.html>, Nov. 1998.
- [Sun98b] *WWW page on Java White Paper: Java Remote Method Invocation - Distributed Computing for Java*, <http://java.sun.com/marketing/collateral/javarmi.html>, Nov. 1998.
- [Woo94] T. K. Woo and M. J. Rees, “A synchronous collaboration tool for World-Wide Web”, *Elec. Proc. Second World-Wide Web Conference '94*, Chicago, USA, October 1994 (also <http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings/CSCW/rees/SynColTol.html>, Nov. 1998)

## Vitae



**Abdul Samad Haji-Ismail** is currently a PhD student in Computer Science at University of Wales Swansea. He received his BSc from University of Wisconsin Superior in 1985 and his MSc from Central Michigan University in 1988. He then joined University of Technology Malaysia as a member of academic staff. His research interests include Internet-based technologies, CSCW and computer networking.



**Min Chen** is a senior lecturer in the Department of Computer Science, University of Wales Swansea. He received his BSc from Fudan University in 1982 and his PhD from University of Wales in 1991. He joined University of Wales Swansea as a member of staff in 1987. His research interests include WWW-based technologies, multimedia communications, computer graphics and volume visualization.



**Phil W. Grant** is a senior lecturer in the Department of Computer Science, University of Wales Swansea. He received his BSc in 1968 and Diploma in Pure Mathematics in 1969, both from the University of Manchester, UK. He obtained his DPhil in Mathematical Logic (Computability Theory) from Oxford University in 1972. He was a lecturer at the University of Sunderland from 1971 to 1975. He joined the University of Wales Swansea in 1976. His research interests include WWW-based technologies, HCI and the application of logic, functional and parallel programming to engineering problems.