JACIE – an Authoring Language for the Rapid Prototyping of Collaborative Applications

ABDUL S. HAJI-ISMAIL, MIN CHEN, PHILIP W. GRANT and
MARK KIDDELL

Department of Computer Science

University of Wales Swansea, SA2 8PP, UK

(E-mail {csabdul, m.chen, p.w.grant, m.kiddell}@swan.ac.uk)

March 20, 2001

Abstract. With the World Wide Web (WWW) becoming the de facto standard for human-computer interaction and human-human communication, there is a need to develop net-centric, multimedia and collaborative applications. We introduce a new scripting language, JACIE, designed to support rapid prototyping and implementation of such applications. The necessity to support the management of multimedia interaction and communication in collaborative applications is highlighted. JACIE facilitates such support through the concepts of channels and a collection of 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. All these features characterise a desirable collaborative software engineering tool.

Keywords: scripting language; authoring tool; multimedia software engineering; 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 is for the distribution of multimedia information, a number of applications have demonstrated that it is equally feasible and effective for WWW to facilitate collaborative activities in multimedia (Woo and Rees, 1994; Frivold et al., 1994).

However, most existing collaborative applications were not developed under the framework of WWW, and they often restrict users' ability to communicate in multimedia. The development of new applications is hindered by expensive development costs, lengthy development period, and shortage of expert knowledge and programming skills. Although there are many powerful general purpose development environments (e.g. Microsoft Visual J++, Borland JBuilder, IBM VisualAge, Supercede Java and Symantec Visual Cafe), to develop multimedia 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 (Frivold et al., 1994; Javawp, 1998)). The resurgence in the popularity of scripting languages such as TCL/Tk, Python, Perl, VBScript and JavaScript further highlighted the importance of programming efficiency in the development of Internet-based applications (Khare, 1997; Dr Dobbs, 2000; Ousterhout, 1998).

In recent years, efforts have also been made to develop special purpose development environments for collaborative applications. They include DIVE (Carlson and Hagsand, 1993), AC3D (Coleburne, 1999), SOL (Smith and Rodden, 1995), Yarn Web (Woo and Rees, 1994), VRML-extension (Broll, 1997), JCE (Abdel-Wahab, 1996), TANGO (Beca et al., 1997), Eventware (EventWare, 1998) and Hesse (Hesse, 1998). Some, such as DIVE, are platform dependent. Many target 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 his recent book (Chang, 2000), Chang discussed the dual roles of multimedia software engineering (MSE), and in particular, highlighted the needs for flexible software engineering tools to support the development of multimedia systems. The authors' work focuses on the support for engineering multimedia systems involving net-centric interaction and communication following Chang's views. Examples of such applications include web-based groupware, collaborative research and development environments, teamwork couseware and multi-user games.

In this paper, we present a scripting language called JACIE (Java-based Authoring language for Collaborative Interactive Environments). JACIE has been designed to support rapid prototyping of network-based applications that feature both multimedia and collaboration. In particular, the language facilitates the management of multimedia interaction and communication through simple communication primitives such as *channels* and *interaction protocols*, which hide much network programming. It supports net-centric collaboration through web browsers as well as stand-alone applications. It also features a template-based programming style, a single program for both client and server, and platform-independence via the target language Java. JACIE provides a high-level tool for engineering a class of software applications which would otherwise incur high development costs as well as long development cycle.

Section 2 overviews net-centric, multimedia and collaborative applications and their system requirements. In Section 3, the design principles of JACIE and its main functional features are described. Section 4 focuses on the management of multimedia interaction and communication in JACIE. In Section 5, we will describe the software architecture on which the implementation is based. In Section 6 a typical application written in JACIE is presented. Section 7 concludes with some observations and plans for further development. The details of the JACIE syntax are outlined in an appendix.

2. Net-centric, Multimedia and Collaborative Applications

The fundamental objective of most collaborative applications is to facilitate effective interaction and communication among users who are engaged in joint 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 (Greenhalgh and Benford, 1995; Greenhalgh, 1996), it is generally recognised 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 (Carlson and Hagsand, 1993), 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 (Normand and Tromp, 1996), DEVRL (Slater et al., 1996), COMIC, VirtuOsi (Benford et al., 1994), MASSIVE (Greenhalgh and Benford, 1995) and MASSIVE-2 (Greenhalgh, 1996). 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.

In parallel with 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: visualising data (Disz et al., 1995; Duce et al., 1998; Lovegrove, 1998), having a meeting (S. Gasner and Ja-

cobson, 1994; Ginsberg and Ahuja, 1995), editing a document, giving a distance learning course (Christiansen and Dirckinck-Holmfeld, 1995; Shotsberger et al., 1995), etc.

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

- (a) collaborative work environments (for conducting collaborative work such as engineering design, visualisation, documentation etc.);
- (b) meetings, seminars and conferences over the Internet;
- (c) simulation of face-to-face contacts where visual quality is critical (e.g., 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).

Table I. Role of communication media (*** necessary, ** desirable, * occasionally useful)

media	a	b	c	d	e	f
text online chat	***	*	**	***	*	*
voice conferencing	**	***	***	**	*	*
video conferencing	*	***	***	**	*	*
shared applications	***	***	**	***	*	*
shared canvas	***	*	*	***	***	**
3D virtual space	*	*	*	*	**	***

Modern data communication technology enables users in a collaborative environment to interact through a variety of communication media. As summarised in Table I, 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 (Summers, 1998), or application software of a distributed nature such as CAVE (Rantzau et al., 1995). In contrast, implementation of a shared interactive canvas is generally application-specific and usually

requires considerable knowledge of system and network programming. This is reflected by the fact that applications in groups (a), (d) and (e) are poorly supported by currently available development tools. Although the implementation of 3D virtual space requires a range of modelling and programming skills in addition to powerful graphics hardware, Table I also indicates that there is limited practical interest in 3D virtual space except for applications in groups (e) and (f).

Consider three typical applications shown in Figure 1. Figure 1(a) shows a CACSD (Computer-Aided Control System Design) environment for designing and simulating dynamic systems (Barker et al., 1991). Such an application may be re-engineered into a net-centric application, allowing engineers in different sites to collaborate in a design activity. Figure 1(b) shows a commercial visualisation system, AVS (AVS, 1991) which utilises visual programming to construct a visualisation. The software already supports distributed data management and visual programming provides an ideal paradigm for multi-user participation. Figure 1(c) shows an online bridge game, which is one of many web-based games available on the Internet (Playsite, 2000). It allows complete strangers from all over the world to participate in an enjoyable leisure activity. It is desirable that scientific and engineering activities (such as the first two applications in Figure 1) can be conducted over the Internet in the same manner. The main difficulty involved in developing net-centric software systems for 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.

Figure 1. Examples of Collaborative Applications: a) CACSD, b) AVS and c) Bridge

In event-driven programming each event is 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 net-centric 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 present in such an environment a user may easily generate inputs that would become invalid upon their arrival at a remote site. Event validation thereby becomes a critical step in the event processing. Furthermore, ensuring 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 (Coulouris et al., 1994).

Furthermore, the management of these events differs depending on individual applications. For instance, a 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, a 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. A collaborative visualisation environment may require the access of data distributed among different users, while allowing each user to exercise certain control over all data in order to construct a visualisation that meets the individual needs. A network-based game is normally governed by a set of rules, and all actions must follow a predefined order and be restricted to their corresponding access domains. To handle correctly events generated by interaction 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. Design Principles and Main Features of JACIE

3.1. OVERVIEW

JACIE is a scripting language designed for rapid prototyping net-centric, multimedia and collaborative applications. It targets a collection of applications for which the existing programming tools would incur expensive development costs. From a software engineering perspective, 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 summarised in the following two sections (see (Haji-Ismail, 1998) for more detail).

It is difficult to describe the whole JACIE language within this limited space. We thus provide the syntax of JACIE in the Appendix, and some example code fragments in Section 6. In the rest of this section, we focus on the description of JACIE's design principles and main features.

3.2. SPECIAL PURPOSE

3.2.1. *Typical applications*

These include groupware (e.g., a distributed timetabling system), courseware (in particular for teamwork-based courseware) and games (e.g., board and card games). These applications commonly feature real-time collaborative activities, a shared working canvas, controlled access domains 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 possibly inexperienced programmers.

3.2.2. *Images and carefully-selected graphics function*

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 modelling 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 scripting 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.

3.2.3. *Structured communications*

On the other hand, the majority of such applications would require a structured communication, where, 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. JACIE addresses this problem by providing a set of built-in protocols that can be easily utilized by JACIE programmers.

3.3. PROGRAMMING EFFICIENCY

3.3.1. *Scripting language*

In the initial design stages of JACIE, considerations were taken into account as to whether JACIE should be a scripting 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 scripting languages. In most cases, the development of a scripting language involves a software library in the target language to support the compilation of common functions. It is easy for a scripting language to provide a software library as a side product, but not vice versa.

3.3.2. *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 compilers job to generate server and client programs to run on different computers.

3.3.3. *Channels*

In an application, communications may take place in a number of different forms of media. JACIE introduces the concept of *channel* 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.

3.3.4. *Interaction Protocols*

An *interaction protocol* defines the rules that govern the means of interactions between users in a 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 users 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 communication 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.

```
JACIE{
  //SYSTEM CONFIGURATION
  applet name X;                      //program name
  appletlauncher imagebutton "X.gif"; //launcher (optional)
  configuration {...}                 //networking parameters
  messages {...}                     //attributes of messages

  //CLIENT BODY
  client implementation {
    declaration {...}                //constants, variables ...
    on canvas {...}                  //initialize canvas
    on session start {...}           //processes on connection
    on session {...}                  //main client control
    on session end {...}              //processes on termination
  }
  //SERVER BODY
  server implementation {
    declaration {...}                //constants, variables ...
    on server start {...}             //initialize server processes
    on session start {...}            //processes on users connection
    on session {...}                  //main server control
    on session end {...}              //processes on user termination
    on server end {...}               //housekeeping on termination
  }
}
```

Figure 2. JACIE Components

3.3.5. *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. 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 straightforward.

3.3.6. *Interfacing with Java*

It is inevitable that 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.

```

...                               //JACIE code
int x[5] = {100, 200, 250, 50, 100}; //JACIE assignments
int y[5] = {50, 50, 200, 200, 50};

Java {                             //Java code

public void paint (Graphics g) {
    g.setColor(Color.blue);        //calling methods
    g.fillPolygon(JACIE_x, JACIE_y, 5); //in a Java class
}

...                               //JACIE code

```

Figure 3. A Java code segment in a JACIE program

4. Multimedia Interactions and Communications in JACIE

4.1. BUILT-IN MULTIMEDIA CHANNELS

As mentioned in Section 3, there are six built-in channels in JACIE, namely *canvas*, *message*, *chat*, *voice*, *video* and *whiteboard*. We described each in more detail in the following subsections.

4.1.1. *Canvas channel*

This 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 initialised 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.

4.1.2. *Message channel*

This 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.

4.1.3. *Chat channel*

This 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

4.1.4. *Voice channel*

This 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.

4.1.5. Video channel

This 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.

4.1.6. Whiteboard channel

This is used for arbitrary drawings on a dedicated canvas.

Figure 4. Canvas, message and chat channels

Figure 4 shows a collaborative scrabble game written in JACIE. It consists of a canvas channel as its main game board, and a private chat channel for each group, through which players of the same group may inform each other of the tiles in their hands and discuss their game strategy. Several message channels are used for displaying the status of the game. Figure 5 shows examples of video and whiteboard channels.

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

Figure 5. Video and white channels

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.

Our study (Haji-Ismail et al., 1999) shows that there are quite a number of issues involved in the design of any collaborative application. Based on natural interaction and collaboration techniques exercised by a human, we have formulated the upper layer interaction protocols for JACIE, which can be categorised into *session management*, *delivery management*, *floor control management* and *group management* which are described in the following subsections.

4.2.1. Session Management

Session management deals with a participating user joining the active session upon established connection, continuously maintaining a session for interactive or collaborative tasks, and closing a session upon termination be it gracefully or ungracefully. Unlike the natural interaction, networked-based collaborative applications should be able to handle multiple concurrent physically dispersed users. The issue of reliability

Table II. Different versions of noughts and crosses

Name	Players	Board	Turn Control	Cell Control	Winner
Traditional Game	2 players	3x3	Play alternates.	One symbol in an empty cell	First with 3 symbols in line
Generalised Game	2 players	8x8	Play alternates.	One symbol in an empty cell	First with 3 symbols in line
Speed Fight	2 players	8x8	Place symbol as fast as can. No turn control.	One symbol in an empty cell	First with 5 symbols in line
Vicious Fight	2 players	8x8	Player places symbols in any cell as fast as they can. No turn control.	An existing symbol can be replaced.	First with 5 symbols in line
Gentlemen's Fight	2 players	8x8	Each player signals other player if he would like to play, and cannot place his symbols until permission is received from other player (or the server).	One symbol in an empty cell	First with 5 symbols in line
Dictator's Game	2 players	8x8	Each player must receive a signal from the dictator (the server) before placing a symbol, the dictator randomly selects a player each time.	One symbol in an empty cell	First with 5 symbols in line
Group Game	2 groups 2 players each group	8x8	Play alternates	One symbol in an empty cell	First group with 3 symbols in line

Table III. 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.	Whiteboard
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.	Speed fight noughts-and-crosses
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
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.	Gentlemen noughts-and-crosses
centralised	Events are generated by a server (or a master client via a server) to activate communication activities of clients in a controlled manner.	Dictator noughts-and-crosses

especially in terms of the management of joining and leaving remote users at arbitrary time, as well as the affect on the current running state of the application, has to be accommodated.

Table IV. Traffic modes and interaction protocols available to each channel

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

4.2.2. *Delivery Management*

Delivery management deals with sending and receiving of data through message passing. This is another important consideration so as to guarantee consistency between the running processes be they at the client or the server side.

4.2.3. *Floor Control Management*

Floor control management coordinates users' turn. While many collaborative application developers approach these problems by adhering to either 'token passing' (in turn, one after the other) or 'contention' (uncontrolled), the fact is that in real life there are also other floor control techniques being adopted. There are 'reservation' (a participant has to flag before being allowed to be involved in activity), 'tapping' (one participant passes the turn to anybody he/she chooses) and 'directed' (participations are being directed by a central figure). There is also a version of token-based — 'timed token' — which is similar to token passing but within a predefined time.

4.2.4. *Group Management*

Group management deals with group formation and interaction within the group. Naturally, a group may be formed by mutual agreement among participants, random selection or some kind of order. If multiple groups are involved, normally, an inner-group interaction may be required as well as an inter-group interaction.

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, as a case study, were implemented in Java prior to the development of JACIE. All games in the case study are designed to run across the Internet, and users on different computers may interact through WWW browsers. Each game implements a different rule that simulates a possible interaction protocol, and Table II lists the main features of a selection of these games in this case study.

4.3. BUILT-IN INTERACTION PROTOCOLS

There are five basic built-in interaction protocols, as detailed in Table III, where we utilise the technical terms used for specifying medium access control methods in low level network communication (Kiddell et al., 1999).

As mentioned in Section 4.1, a channel may be customised 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 IV. 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.

5. Software Architecture of JACIE

JACIE utilises WWW as the main vehicle of delivery for collaborative applications, although it also supports stand-alone implementation of net-centric applications. Its general functionalities, 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 (Summers, 1998). Figure 6 illustrates the software architecture of JACIE.

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

`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_*.java` a set of application specific classes in Java for supporting client operations in the individual components of `x.jacie`;

Figure 6. Software Architecture of 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_*.java a set of application specific classes in Java for supporting server operations in the individual components of **x.jacie**;

x_vconf.java a program which is written in Java 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.

There is a significant amount of coding efficiency achieved through JACIE. We carried out a comparison between a JACIE ‘hello_world’ program (with a canvas channel and two message channels) and the equivalent Java program (Haji-Ismail, 1998). The size of the JACIE program is 4% of that of the Java program in terms of the number of lines. 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 7. Teamwork Application for Network Troubleshooting

In order to give an impartial and objective comparison, we also implemented a ‘knock-knock’ program (Campione and Walrath, 1998) which is used by several Java textbooks as an example for client-server programming. Although this is not a typical JACIE application, about 40% code reduction in terms of the number of lines was achieved.

6. Example Application

Figure 7 shows a multimedia application written in JACIE, which supports teaching and learning of *network troubleshooting* (NTS). Students are given the description of a problem in a set of interconnected networks, and they have to diagnose and locate the malfunctioning part collaboratively. The application uses a canvas channel for displaying a collection of interconnected networks. Each student has control of one or more networks, and can manipulate them (e.g., switching individual devices on or off, connecting or disconnecting a link). Most of the drawings on the canvas are written with JACIE’s `draw_image` statement. Small pre-recorded videos are used to give a more realistic display of the current status of each device through the `draw_video` statement (Figure 8).

Figure 8. A separate window displays a pre-recorded video which shows the traffic status of a particular hub selected in a local view.

As no single user has the overall control of all networks, collaboration among users is essential for successful troubleshooting. A shared chat channel is thus implemented to facilitate the communications among users. Such a channel can easily be replaced by a voice channel or video channel should there be an access to a multi-point conferencing unit.

The NTS application employs a *contention* protocol for its user-user interaction protocol. Remote users can send messages to the server any time without the need of turn control. Messages are used for requesting the status of a selected local device, informing the server of any operations (such as switching off) that have been made to a device, activating a ‘ping’ command on remote devices, etc. The server’s role is to maintain the global status of the testing networks and to provide clients with requested information about individual devices.

As shown in Figure 2, JACIE employs a template-based programming style with its program divided into a set of standard components. All components are written within the delimiter:

JACIE {...}

Each program is divided into three main components, namely *system configuration*, *client body* and *server body* and are described in the following subsections.

6.1. SYSTEM CONFIGURATION

This component is used to specify the program name and some basic networking parameters. For example, the NTS application was defined with following system configuration component:

```

applet name NetTrouble
appletlauncher text
"Collaborative Network Troubleshooting Training"
configuration {
    host "csbean.swan.ac.uk";
    port 1234;
    username prompt;
    channel canvas, chat;
    about file "NetTrouble.txt";
    number of users maximum 3;
    protocol contention;
}
messages {
    roomAssigned, problemStatement,
    userRequest, deviceConfig,
    problemSolved, ...;
}

```

The first statement specifies the program type (in this case, an applet) and program name. Under the **configuration** construct, there are statements that specify the **hostname** where the server program will reside, the **port number** where the communication will take place and the **username** of the client. Alternatively the user may also specify the **hostname**, **port number** and **username** during execution. These are followed by specification of a canvas and a chat channel to be used for the application. This component also specifies the maximum number of concurrent users allowed during the execution, and the floor control. Following **configuration** is the **messages** construct, which declares all message identifiers.

6.2. CLIENT BODY

In a client/server environment, processes acting as clients and servers are normally distributed on different computers. The client body of a JACIE program specifies the code for a client process. All interactions between the client and the server are through message transfer by means of message identifiers declared in the system configuration component.

The client body consists of several program constructs as shown in Figure 2. Because JACIE is a strongly-typed language, the **client implementation** component begins with a **declaration** construct where all the variables used within the component are declared. In addition

to user-defined variables, JACIE also provides a set of system variables that can be used by programmers. The declaration construct also allows methods to be defined. An example extracted from the NTS application is given as follows:

```
client implementation {
  declaration {
    int roomNumber;
    string problem;
    int currentView = 1; // 1:global, 2:local
    string deviceName;
    string ipAddress;
    string netMask;
    string defaultGateway;
    string deviceStatus;
    image globalViewLayout="Layout.gif";
    image globalViewOn="GlobalOn.gif";
    image globalViewOff="GlobalOff.gif";
    image localViewOn="LocalOn.gif";
    image localViewOff="LocalOff.gif";
    image problemIcon="ProblemIcon.gif";
    image workstation="PC.gif";
    image printer="Printer.gif";
    image hub="Hub.gif"; ...
  } ...
}
```

Following `declaration` is the `on canvas` construct which specifies the default workspace canvas (such as background image) on which all user-defined interactions and collaborations will take place. The canvas image may be changed at a later stage during the execution by other graphics statements (e.g. `define canvas`, `use canvas`, etc). Some example statements for the NTS application are:

```
on canvas{
  foreground white;
  define canvas globalView {
    draw grid ViewIcons at 10,10
      step 100,25 size 3,1;
    draw image grid ViewIcons
      globalViewOn at 0,0;
    draw image grid ViewIcons
      localViewOff at 1,0;
    draw image grid ViewIcons
      problemIcon at 2,0;
    draw image globalViewLayout
      at 10,35; ...
  }
  use canvas globalView; ...
}
```

With this set of statements, an initial global view is drawn onto the canvas (Figure 7).

The next three constructs, `on session start`, `on session` and `on session end`, define the main interaction and communication activities of a client process. The `on session start` construct normally defines a set of initialisation operations, while `on session end` performs operations upon termination.

The `on session` construct is the main focus of a JACIE program, where most user interactions and client-server communications will take place. Interactions specified within `on session` normally take place on the canvas, and they are processed in an event-driven manner. The following code segment is extracted from the NTS application.

```

on session { ...
  if (currentView==1) {
    use canvas globalView;
    on MOUSECLICK {
      if (GETGRID==ViewIcons)
        if (GETGRIDX==0)
          print "You are in Global View";
        else if (GETGRIDX==1)
          currentView = 2;
        else
          print problem;
    }
  } else { // currentView = 2
    use canvas localView;
    on MOUSECLICK {
      // display information of device if
      // on a hot spot or change to global
      // view the user request to ...
    }
    on TEXTENTERED {
      input instruction;
      send userRequest instruction;
    }
    on NEWMESSAGE {
      if (MESSAGEID==deviceConfig)
        receive deviceConfig,
        deviceName, ipAddress,
        netMask, defaultGateway; ...
    }
  }
} ...

```

This code segment shows an event handler in response to a user's mouse click action during his/her turn. For example, if the current view is the global view and the mouse position is on 'Local View' button (Figure 7), a mouse click will result in changing the view to the local view (Figure 8).

6.3. SERVER BODY

The `server implementation` component specifies code for a server process. As shown in Figure 2, it consists of six constructs. The function of the `declaration`, `on session start`, `on session`, and `on session end` are very similar to those in the client body. The two additional constructs, namely `on server start` and `on server end`, are used to interact with the operating system for maintaining the system status of the server. The `on server start` construct initialises the server process upon invocation of the server program. The server process will stay alive, waiting for the client to establish connection from then on. The `on server end` construct performs some housekeeping operations upon server termination.

For the NTS application, the main function of the server is to keep track of all the entities in the network (their names, their network configurations, and their status). It coordinates the cause and effects from one user's action in relation to other remote users. For instance, if one user disconnects one hub under his control, it will affect the connectivity from other nodes in the section to other sections).

In this example, the problem is predefined by the programmer. A more complex version with multiple problems can also be implemented in JACIE. However, the availability of the Internet as the main medium enables us to maintain the simplicity of a JACIE program while facilitating multiple problems through a front-end page written in HTML. At the beginning, a network problem is generated using:

```
server implementation {
  declaration {
    ...// declaring variables
    // declaring methods
    void createProblem() { ... }
    void replyInstruction
      (String instruction) {
      ...
      send deviceConfig, deviceName,
        ipAddress, netMask, ...
      send problemSolved; ...
    }
  } ...
  on session start {
    createProblem();
    send roomAssigned USERNUMBER;
    send problemStatement problem; ...
  } ...
}
```

`USERNUMBER` is a system variable assigned to each user upon a connection. In this case, the room number takes the `USERNUMBER` value so that the first remote user gets Room 1, and so on. `createProblem()` is a

method defined in the `declaration` construct of `server implementation`. It initialises the status of all entities in the network.

The `on session` construct further defines the responses of the server to the messages from clients as:

```
on session {
  ...
  on NEWMESSAGE {
    receive userRequest instruction;
    replyInstruction(instruction);
    ...
  }
} ...
```

where `replyInstruction(...)` is a method that parses user instructions and send back the reply.

From the above code segment, one can also easily see that the declaration and assignment statements in JACIE follow a very conventional style which is common to most programming and scripting languages. In fact the syntax is simpler than Java in some respect.

7. Conclusions

We have described a novel scripting 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 focuses on the development of some practical applications of a considerable scale in JACIE, including a groupware system.

References

- Abdel-Wahab: 1996, 'Using Java for multimedia collaborative applications'. In: *Proc. 3rd International Workshop on Protocols for Multimedia Systems (PROMS'96)*.
- AVS: 1991, 'AVS User's Guide, Release 4'.

- Barker, H. A., M. Chen, P. W. Grant, I. T. Harvey, C. P. Jobling, A. P. Parkman, and P. Townsend: 1991, 'The making of eXCeS — a software engineering perspective'. In: *Proc. 5th IFAC/IMACS Symposium on Computer Aided Design in Control Systems*. pp. 27–32.
- Beca, L., F. G. Cheng, T. G. C. Jurga, X. Olszewski, M. K. Podgorny, and K. Walczak: 1997, 'Web technologies for collaborative visualization and simulation'. In: *Proc. the 8th SIAM Conference On Parallel Processing*.
- Benford, S. D., J. Bowers, S. Gray, T. R. Rodden, M. Rygol, and V. Stanger: 1994, 'The VirtuOsi project'. In: *Proc. VR'94 (London Virtual Reality Expo 1994)*. Meckler.
- Broll, W.: 1997, 'Distributed virtual reality for everyone — a framework for networked VR on the Internet'. In: *Proc. of the IEEE Virtual Reality Annual International Symposium 1997 (VRAIS'97)*. Albuquerque, NM, IEEE Computer Society Press.
- Campione, M. and K. Walrath: 1998, *The Java Tutorial*. Reading, Massachusetts: Addison–Wesley, 2 edition.
- Carlson, C. and O. Hagsand: 1993, 'DIVE: a platform for multi-user virtual environments'. *Computers and Graphics* **17**(6).
- Chang, S. K.: 2000, *Multimedia Software Engineering*. Kluwer Academic Publishers.
- Christiansen, E. and L. Dirckinck-Holmfeld: 1995, 'Making distance learning collaborative'. In: *Proc. Compute-r-Supported Collaborative Learning '95 (CSCL'95)*. Lawrence Erlbaum Associates, Inc.
- Coleburne, A.: 1999, 'ACD3 User manual'. Technical report, Department of Computer Science, University of Lancaster, UK. <http://www.comp.lancs.ac.uk/computing/research/cseg/>.
- Coulouris, G., J. Dollimore, and T. Kindberg: 1994, *Distributed Systems: Concepts and Design*. Addison–Wesley, 2 edition.
- Disz, T. L., M. E. Papka, M. Pellegrino, and R. Stevens: 1995, '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*. pp. 217–237, Springer-Verlag.
- Dr Dobbs: 2000, 'Scripting and Alternative Languages'. *Dr. Dobb's Journal: Software Tools for the Professional Programmer* (308).
- Duce, D. A., J. R. Gallop, I. J. Johnson, K. Robinson, C. D. Seelig, and C. S. Cooper: 1998, 'Distributed cooperative visualization — the MANICORAL approach'. In: *Proc. 16th Eurographics UK Conference*. Leeds, UK, pp. 69–86.
- EventWare: 1998, 'WWW page on Eventware: Collaborative Software for a New Age'. <http://eventware.com>, Collaborative Systems Research Inc.
- Frivold, T. J., R. E. Lang, and M. W. Fong: 1994, 'Extending WWW for synchronous collaboration'. In: *Elec. Proc. Second World-Wide Web Conference '94*. Chicago, USA. <http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings/CSCW/frivold/frivold.html>, Nov. 1998.
- Ginsberg, A. and S. Ahuja: 1995, 'Automating envisionment of virtual meeting room histories'. In: *Proc. ACM Multimedia'95*. San Francisco, USA, pp. 65–75.
- Greenhalgh, C. M.: 1996, 'Dynamic Embodied Multicast Groups in MASSIVE-2'. Technical Report NOTTCS-TR-96-8, Department of Computer Science, University of Nottingham, UK.
- Greenhalgh, C. M. and S. D. Benford: 1995, 'MASSIVE: a virtual Reality System for teleconferencing'. *ACM Transactions on Computer Human Interfaces (TOCHI)* **2**(3), 239–261.

- Haji-Ismail, A. S.: 1998, 'JACIE — Java-based Authoring language for Collaborative Interactive Environments'. Technical report, Department of Computer Science, University of Wales Swansea.
- Haji-Ismail, M. A. S., M. Chen, and P. W. Grant: 1999, 'Managing interactions and communications in collaborative multimedia applications: the JACIE way'. In: *Proc. Second International Conference on Information, Communications, and Signal Processing (ICICS'99)*.
- Hesse: 1998, 'WWW page on Hesse Collaborative Framework Overview'. <http://www.praxistech.com/hesse/hesse.html>.
- Javawp: 1998, 'WWW page on Java White Paper: The Java Language: An Overview'. <http://java.sun.com/docs/overviews/java/java-overview-1.html>.
- Khare, R.: 1997, 'Scripting Languages: automating the Web, Editorial'. *World-Wide Web Journal* **11**(2).
- Kiddell, M., M. Chen, D. J. Osborne, F. W. Slater, and M. McCulloch: 1999, 'TRIUMF — A System for Remote Multimedia Interviewing'. In: *Proc. IEEE International Conference on Multimedia Computing and Systems (ICMCS'99)*, Vol. 11. pp. 715–719.
- Lovegrove, S.: 1998, 'Collaborative research within a sustainable community: interactive multi user VRML and visualization'. In: *Proc. 16th Eurographics UK Conference*. Leeds, UK, pp. 53–68.
- Normand, V. and J. Tromp: 1996, 'Collaborative virtual environments: the COVEN project'. In: *Proc. the Framework for Immersive Virtual Environments Conference, FIVE'96*.
- Ousterhout, J. K.: 1998, 'Scripting: Higher Level Programming for the 21st Century'. *Computer* pp. 22–30.
- Playsite: 2000, 'PLAYSITE: the web's premier free multiplayer games community'. <http://www.playsite.com/>.
- Rantzau, D., U. Lang, R. Lang, H. Nebel, A. Wierse, and R. Ruehle: 1995, '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*. pp. 207–215, Springer-Verlag.
- S. Gasner, S. Schooler, R. F. and V. Jacobson: 1994, 'Multimedia teleconferencing on the Internet multicast backbone (MBONE)'. In: *Proc. ACM Multimedia'94*. San Francisco, USA.
- Shotsberger, P. G., K. B. Smith, and C. G. Spell: 1995, 'Collaborative distance learning on the World Wide Web: would that look like?'. In: *Proc. Computer-Supported Collaborative Learning '95 (CSCL'95)*. Lawrence Erlbaum Associates, Inc.
- Slater, M., M. Usoh, S. Benford, D. Snowdon, C. Brown, G. S. T. Rodden, and S. Wilbur: 1996, 'Distributed extensible virtual reality laboratory (DEVRL)'. In: M. Goebel (ed.): *Proc. 3rd Eurographics Workshop on Virtual Environments*.
- Smith, G. and T. Rodden: 1995, 'SOL : a toolkit for cooperative interfaces'. *International Journal of Human Computer Studies* (42), 207–234.
- Summers, B.: 1998, *Official Microsoft Netmeeting Book*. Microsoft Press.
- Woo, T. K. and M. J. Rees: 1994, 'A synchronous collaboration tool for World-Wide Web'. In: *Elec. Proc. Second World-Wide Web Conference '94*. Chicago, USA. (also <http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings/CSCW/rees/SynColTol.html>, Nov. 1998).

Appendix

JACIE LANGUAGE

In this appendix we outline the main features and syntax of the JACIE language.

DATA TYPES

Primitive Types `int`, `byte`, `float`, `boolean`, `image`, `string`

Reference Type `array`

VARIABLE DECLARATION AND INITIALISATION

- Variable names in JACIE follow the convention and rules of variable names in Java.
- JACIE is a strong type-checking language. Except for a for-loop index, all variables have to be declared in declaration section. In the case of for-loops index, the variables can be declared and assigned in the loop counter itself.
- By default all numeric variables of `int` and `float` types are initialised to 0. String variable will be initialised to the null string (a string of no characters). Other initial literal values may also be assigned in the declaration section.
- Variables of type `boolean` may or may not be initialised in the declaration section but have to be assigned with a value before they can be used.
- Variables of type `image` can be assigned a string value representing the image file name. The string should comprise of `".gif"` or `".jpeg"`.
- Declaration and/or initialisation of variables for primitive types is as follows:
`DataType VariableName [ = IntialValue];`
- Declaration of multiple variable names in one declaration statement is not supported.
- JACIE `array` type uses square brackets (`[` and `]`) in front of the data type which represent the element type. Array size and number of dimensions have to be specified during declaration. Java array object instantiation is not used. As in Java, JACIE indices begin at 0.

- Declaration of **array** type variables is as follows:
`ElementType[ArraySize1, ArraySize2, ...] ArrayName;`
- A single value can be initialised to each of the array elements as follows:
`ElementType[ArraySize] ArrayName = InitialValue;`
- Different value may also be initialised to the array elements using curly braces { and } as follows:
`ElementType[ArraySize] ArrayName = Value1, Value2, ... ValueN, or`
`ElementType[ArraySize1] ... [ ArraySizeN] ArrayName = Value1, ... ValueN;`
 where N should tally with the array size(s) specified.

ARITHMETIC, RELATIONAL AND CONDITIONAL OPERATORS

Arithmetic Operators Java's binary arithmetic operators +, −, *, /, % are adopted. Similarly, unary − is used in negation. These operator, except %, can be applied to any arithmetic expression. Operator % (for remainder of division) can only be applied to **int** expressions. Operator + can also be used to concatenate string values.

Relational and Conditional Operators Java's relational operators >, >=, <, <=, ==, != are adopted in JACIE. Likewise, Java's binary conditional operators |, &, ||, &&, ! are also being adopted. Java's tertiary conditional operator ?: is not supported as well as bitwise and logical operators.

Expressions and Operator Precedence JACIE expressions and operator precedence follow the rule of Java's.

STATEMENTS

Comment

```
// one line comment
or /* multiple line comment
...
*/
```

Setup Statements

```
JACIE Program:
JACIE { CreateApplication | CreateApplet }
```

CreateApplication:

```

application name ApplicationName ;
configuration { ProgramConfiguration }
messages { MessageDefinition }
client implementation {
    ClientProgramImplementation }
server implementation {
    ServerProgramImplementation }

```

CreateApplet:

```

applet name AppletName ; [CreateAppletOption]
configuration { ProgramConfiguration }
messages { MessageDefinition }
client implementation {
    ClientProgramImplementation }
server implementation {
    ServerProgramImplementation }

```

CreateAppletOption:

```

appletlauncher [text TextString] | [image ImageFileName] ;

```

ProgramConfiguration:

```

host HostName | prompt ;
port PortNumber | prompt ;
username UserName | prompt ;
[channel [canvas] [chat] [whiteboard] [voice] [video];]
[about AboutString | file AboutFileName ;]
number of users [minimum Number] [maximum Number] ;
[number of observers Number ;]
[number of groups Number ;]
protocol unstructured | contention | token-pass | reservation | tapping | centralised;
[protocol of group userdefined | random | alternate ;]

```

MessageDefinition:

```

MessageId1, MessageId2, ..., MessageIdN;

```

ClientProgramImplementation:

```

declaration VariableMethodDeclarationList
on canvas CompoundStatement
on session start CompoundStatement
on session CompoundStatement
on session end CompoundStatement

```

ServerProgramImplementation:
 declaration *VariableMethodDeclarationList*
 on server start *CompoundStatement*
 on session start *CompoundStatement*
 on session *CompoundStatement*
 on session end *CompoundStatement*
 on server end *CompoundStatement*

Compound Statements

```
{
  Statement1 ;
  Statement2 ;
  :
  StatementN ; }
```

Assignment Statements

VariableName = *Constant* | *Expression*

Java's shortcut assignments (+=, -=, *=, etc.) are not supported.

Conditional Statements

```
if (BooleanExpression)
  Statement1 [else Statement2]
```

Java's switch statement is not supported

Loop Statements

```
for (Initialisation ; Termination ; Increment)
  Statement
while (BooleanExpression)
  Statement
```

Java's do-while statement is not supported

Graphics Statements

```
foreground ColourName
background ColourName
refresh
clean
```

```
move to [grid GridName] CoordX,CoordY
```

```
draw grid GridName [at CoordX,CoordY][step Xstep,YStep]
[size Width, Height][colour ColourName][width Number]
```

```
draw line [grid GridName][from CoordX1,CoordY1]
to CoordX2,CoordY2 [colour ColourName][width Number]
```

```
draw image [grid GridName] ImageVariable [at CoordX,CoordY]
[size Width,Height][flip horizontally | vertically | diagonally ]
```

```
draw string [grid GridName] Constant | StringVariable
[at CoordX,CoordY] [font FontName] [size Number]
```

```
draw video VideoName
```

```
define canvas CanvasName {
    Statement ... }
```

```
use canvas CanvasName
```

Event Control Statements

```
on Event {
    Statement ... }
```

System events:

MOUSECLICK, MOUSEMOVE, NEWMESSAGE, TEXTENTERED, WAITING,
SERVERABORT, CLIENTABORT, TURN, REQUESTCONTROL, RESERVATION,
OBSERVERCONNECTION

System variables:

GETX, GETY, GETGRID, GETGRIDX, GETGRIDY, USERNAME, PORTNO,
HOSTNAME, CURRENTTURN, MYGROUP

Communication Statements

```
send MessageId Variable1 [, Variable2 [, ...]] [to server | group | all]
receive MessageId Variable1 [, Variable2 [, ...]]
exit
abort
```

Input/Output Statements

```
input Variable
print [servermessage] Constant | Variable | Expression
clear localmessage | servermessage
```

Other Statements

```
wait NumberOfSeconds
```

Interfacing to External Java Classes/Codes

```
java {
    Java Statement ... }
```