



**The Chinese University of Hong Kong
Computer Science and Engineering Department**

Final Year Project (LYU 9902)

Wireless Campus

Report version 3.1

Supervisor: Prof LYU, Rung Tsong Michael

Marker: Prof CAI, Leizhen

By Wong Ho Yin, Starsky 97544174

Wong Kwok Hung, Marti 97570894

Date: 1/12/99

Table of Contents

| | Page |
|---|-------------|
| 1. Introduction | 4 |
| 1.1 Project Overview | 4 |
| 1.2 Programming platform | 5 |
| 1.3 Project Architecture | 5 |
| 2. System level of our project | 6 |
| 2.1 Overview of the system level of our project | 6 |
| 2.2 DirectX library | 6 |
| 2.2.1 What is DirectX? | 6 |
| 2.2.2 Our own class of library | 6 |
| 2.2.3 The prototype of our Direct Draw class | 7 |
| 2.2.4 Description of Direct Draw function | 7 |
| 2.2.5 Class "Sprite" | 10 |
| 2.2.5.1 Introduction to spt files | 10 |
| 2.2.5.2 The prototype of Class Sprite | 11 |
| 2.2.6 Class "Frame" | 11 |
| 2.2.6.1 Frame splitting | 12 |
| 2.2.6.2 Algorithm | 12 |
| 2.2.6.3 The prototype of our Class Frame | 12 |
| 2.2.7 Class "Region" | 13 |
| 2.2.7.1 The prototype of our Class Region | 13 |
| 2.2.8 Attachment of other Frame Attributes | 13 |
| 2.2.9 An example of using a Spt file to represent an interactive object | 14 |
| 2.2.10 Our Direct Sound Class | 16 |
| 2.2.10.1 The prototype of our Direct Sound Class | 16 |
| 2.2.10.2 Mixer problem in Direct Sound | 18 |
| 2.3 DirectShow Library | 18 |
| 2.3.1 What is DirectShow? | 18 |
| 2.3.2 DirectShow Architecture | 19 |
| 2.3.3 What we had done on DirectShow library? | 20 |
| 2.4 WinSock library | 20 |
| 2.4.1 What is WinSock? | 21 |
| 2.4.2 The Sockets Programming Paradigm under Windows | 22 |
| 2.4.3 Why use WinSock? | 23 |
| 2.4.4 Blocking and event-Driven Application | 23 |
| 2.4.5 Function prototype of WinSock library? | 23 |
| 2.4.6 Description of WinSock library | 24 |
| 2.5 The Server | 24 |
| 2.5.1 Overview | 24 |
| 2.5.2 The simple broadcast server | 25 |
| 2.5.3 Problems of the simple broadcast server | 25 |
| 2.5.4 The Improved server | 27 |
| 2.5.5 The Improved server Architecture | 27 |
| 2.5.6 Message structure | 29 |
| 2.5.7 The client list | |

| | |
|---|-----------|
| 3. Collaborative Environment(CE) | 30 |
| 3.1 Introduction | 30 |
| 3.2 Interface | 32 |
| 3.2.1 Roomlist | 33 |
| 3.2.2 Classmate List | 34 |
| 3.2.3 Using CAL (self learning application) | 35 |
| 3.2.4 Chatroom | 36 |
| 3.2.5 Writeboard | 37 |
| 3.2.6 Mediaroom | 39 |
| 3.2.7 Voting | 40 |
| 4. Role Play Collaborative Environment(RPCE) | 41 |
| 4.1 Introduction | 41 |
| 4.2 Walking | 42 |
| 4.3 Talking | 43 |
| 4.4 Whispering | 44 |
| 4.5 Paging (Private Message) | 44 |
| 4.6 Playing games (group learning application) | 45 |
| 4.7 Using CAL (self learning application) | 45 |
| 4.8 Client-Server interaction | 46 |
| 4.9 Picture Compression | 47 |
| 4.9.1 Overview | 47 |
| 4.9.2 Our approach | 47 |
| 4.9.2.1 Huffman Code | 48 |
| 4.9.2.2 Disadvantages of Huffman Code | 48 |
| 4.9.2.3 LZW | 49 |
| 4.9.2.4 Conclusion | 53 |
| 4.10 Further Improvement | 54 |
| 4.11 Conclusion on CE | 55 |
| 5. Personal Application | 56 |
| 6. Conclusion | 58 |
| Appendix A References | 59 |
| Appendix B Progress Report | 60 |
| Appendix C Statistics of our program | 62 |

1. Introduction

1.1 Project Overview

Our aim of doing the project is to enhance the learning experiences among, students in team projects and allow students to perform joint work even they are physically apart.

In our project, we have implemented the whole system, system level and application level or programming.

In system level, there are a server and some libraries.

The server works as a central unit of the whole system. It controls every message to pass through the system and controls the usage of resource inside the system. The whole system cannot work without it.

The client programs are used by the students. They can invoke one or more activities (chat room, voting, write board or media sharing) in it. The instance of each activity runs locally at each participant's site and the response of each user is distributed to all the participants.

The client program is for received the response from the user and send it to the server. After server received their response, it will calculate the new state of the user and distribute it to all of the users within the system.

The libraries help us to develop a highly interactive network application. The libraries include Winsock, Direct X and Direct Show library . Based on them, we have designed and built the system which is suitable for future world. We have tried to develop the software in 2 approaches. The first one's interface is more efficient and clear and the second one's is more user friendly and simple. Finally, we have compared these 2 approaches and give a conclusion.

1.2 Programming Platform

All of the programs are written in Microsoft Visual C++.

For the server program, it needs to run on Microsoft Windows 95/98/NT platform with Winsock support.

For the client program, it needs to run on Microsoft Windows 95/98 platform with DirectX version 5.0 or above support.

1.3 Project Architecture

This project **Digital School** is mainly divided into two parts.

The first part is the server, which mainly responsible for central control of the whole system. The second part is the client program, which also called “Digital School”. The client is mainly response for display for the learning material and interactive activities.

Basically, our work includes implementing the libraries, building the server, and building the client program.

For those libraries we had created, they will be talked in details at Chapter 2.1 to Chapter 2.4 in this report.

For the server we had built, they will be talked in details at Chapter 2.5 in this report.

For the client program we had built, detail will be talk in Chapter 3, 4 and 5 in this report. Here is the architecture overview:

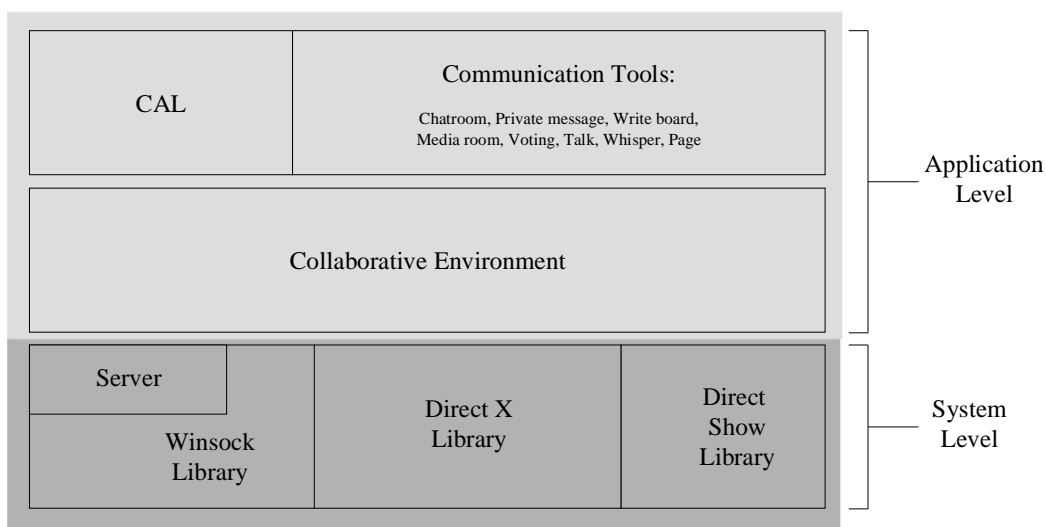


fig 1.1 Architecture Overview

2. System level of our project

2.1 Overview of the system level of our project

There are four main parts to help us to construct our project:

1. DirectX library
2. DirectShow library
3. WinSock library
4. The Server

Basically, our project is based on some libraries we had built. These libraries are the DirectX library, DirectShow library and WinSock library. The DirectX library is mainly responsible for the graphic display. The DirectShow library is mainly responsible for video display. The WinSock library is mainly response for sending and receiving data in the network.

Also, there is a server running at the background to support our project.

2.2 DirectX library

2.2.1 What is DirectX?

In order to write a fancy and interactive application for our project, we have built our own graphical libraries and audio libraries. DirectX is a Microsoft Windows® API such that it can provide display of images in 2D/3D and playback sound files on Windows.

2.2.2 Our own class of library

Since the APIs provided by Direct Draw are so confused and complicated, we have constructed our Class of Direct Draw to encapsulate the details of the function calls. With this Class, we can create and access the objects of Direct Draw easily.

2.2.3 The prototype of our Direct Draw class (written in pseudo-code)

```
Class DD {  
Public:  
    DDStartup();  
    CreateDesktopWindow( Window_handler, Direct_X_object ); DDFullConfigure( Direct_X_Object );  
    DDWinConfigure( Direct_X_Object );  
    DDLoadPalette( Bmp_file, Palette_object );  
    DDLoadBitmap( Bmp_file, Surface_object);  
    DDSetColorKey( Surface_object, Key_color );  
    DDMakeOffscreenSurface();  
    DDCreateFlipper( Flipper_Object);  
    DDCreateFakeFlipper( Flipper_Object );  
    DDFlipping();  
    DDFillSurface( coordinates, color );  
    DDTextOut ( Surface_object );  
    Cleanup();  
Private:  
    Direct_X_Object DX;  
    Surface primary_surface, secondary_surface;  
    Window_handler ghwnd;  
    Character Mode;  
};
```

2.2.4 Description of Direct Draw function:

DDStartup():

This function is used to create an instance of a DirectDraw object.

CreateDesktopWindow(Window_handler, Direct X object):

This function is use to create a window. We can also configure the window's attributes like width, height, style, menu name, and background by using this function.

DDFullConfigure(Direct X Object):

This function is used to initialize the DirectDraw object with a full screen mode.

DDMakeOffscreenSurface():

This function is used to make offscreen surface (secondary surface).

Typically, we will construct at least 2 surfaces when using DirectDraw. The primary surface and the secondary surface. Primary surface can be considered as the memory content in the video card and is always be shown on the screen. In the flipping process, the content of secondary surface will be copy to the primary surface.

If we don't use flipping, the cleanup process must be done directly to the primary surface. Because the background surface will cover the foreground surface when updating, the showing time of background picture will be longer than that of foreground picture. Therefore, there will be twinkling effect in the human eyes.

If we use flipping, the cleanup and updating process will be done behind the primary screen. We will flip the secondary surface only when all of the foreground surfaces are ready. So it can eliminate the twinkling problem.

DDCreateFlipper(Flipper_Object):

This function is used to create a flipper instance

DDCreateFakeFlipper(Flipper_Object):

This function is used to create a fake flipper.

In window mode, flipper object is not supported, so we need to use this function to create a fake flipper. It simulates the flipping process by using the memory copy function. Its performance is as good as a real flipper.

DDFlipping():

This function is used to do the Flipping. This function will know whether we are using the fake flipper or the real flipper.

DDFillSurface(coordinates, color):

This function is used to fill a rectangle on the surface. We can use this function to cleanup the screen if we do not have a background picture.

DDTextOut (Surface_object):

This function is used to print text on a surface.

CleanUp():

This function is used to cleanup all Direct Draw Object.

2.2.5 Class “Sprite”

Class *Sprite* is built to manipulate the *.spt files. It has included the **data structure for a spt file**, the **method for opening spt file** and the **method for handling spt’s data**.

2.2.5.1 Introduction to spt files:

In order to deal with the *picture* files (*.bmp) easily and efficiently. We have defined a file type called *spt* file. *Spt* file can be used to represent a man, a dog, a robot, or other interactive object in our application. Each *sprite* file (*.spt) has its corresponding *picture* files (*.bmp). The file name of the *picture* files (*.bmp) and the information of the *sprite* will be stored in its *sprite* file (*.spt).

Here is an example:

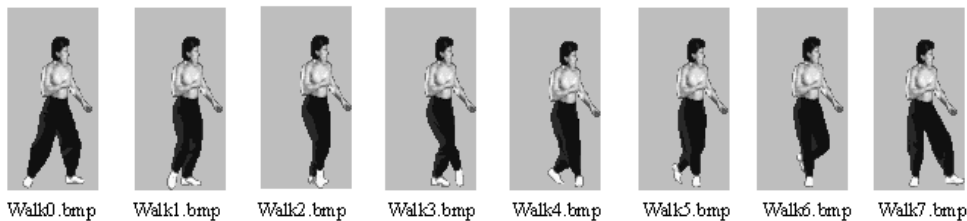


Fig 2.1 8 action pictures for a walking man

We have got the pictures of a walking man and want to play as animation. So we can define a spt file for that walking man.

Here is the content of the spt file:

```
Sprite_name: "Walking Man"  
No_of_frame: 8  
Size_of_Picture: 150 x 100  
Picture_filetype: bmp  
Picture_filename: walk0, walk1, walk2, walk3, walk4, walk5, walk6, walk7  
Transparency_color: 27  
Frame_rate: 25  
Reserved_space_for_future_use: .....
```

By using this kind of spt file (Our programs have a different format of spt file, this example is used for demo only), the programmer can write a generic animation player to play some simple animation. Actually spt file can be used to represent some interactive object rather than animation. There will be an example later in this chapter.

X.2.5.2 The prototype of *Class Sprite*: (written in pseudo-code)

```
class Sprite
{
public:
    LoadSprite ( String sprite_file );
    SaveSprite ( String sprite_file );
    Setbmpfilename( string );
    SetId ( int );
    InsertFrame ( int, Frame );
    DeleteFrame ( int );
    SetReserved_string ( int, string );
    SetReserved_integer ( int, int );
    String getbmpfilename();
    Integer getId ();
    String GetReserved_string ( int );
    Integer GetReserved_integer ( int );

protected:
    Frame frm[];
    /* Frame will be introduced in the next session */
    /* Frame is not private because its member Retion need to be read by some friend functions for
region calculations */

private:
    String Bmp_filename;
    Integer id;
    Integer no_of_frame;
    String Reserved_string[20];
    Integer Reserved_integer[100];
};
```

2.2.6 Class “Frame”:

***Class Frame* is built to store the information for each action.**

(*Frame* instance is defined as the member under *Sprite*.)

A *Sprite* may include several actions. In page 10, the example of *Sprite*, the *spt* file is representing a walking man with 8 frames.

2.2.6.1 Frame splitting:

Because each frame has its action picture, if there are many frames, it is so troublesome to deal with large number of picture files. Therefore, we have written a function for frame splitting. It let us to draw all the frame pictures in one picture file. And the method for opening a *spt* file can recognize all the frames' position automatically.

2.2.6.2 Algorithm

(Assumption: each frame is drawn within a square)

```

For y = 0 to height of picture
  For x = 0 to width of picture
    If Pixel(x, y) != background_color and Pixel(x, y) is not used
      begin
        /*Pixel (x, y) is the upper left corner of this frame*/
        Find the upper right corner of this frame
        Find the bottom left corner of this frame
        Save the coordinates of this frame in a linked list
      end
    Next x
  Next y

```

2.2.6.3 The prototype of our *Class Frame*: (written in pseudo-code)

```

class Frame
{
public:
  Frame get_frame ();
  Set_frame ( Frame );

private:
  String name_of_frame;
  Integer no_of_region;
  Region rg[];
  /* Region will be introduced in the next session */
  int go; /* for animating purpose */
  int wait; /* for animating purpose */
  String Reserved_string[20];
  Integer Reserved_integer[100];
};

```

2.2.7 Class “Region”:

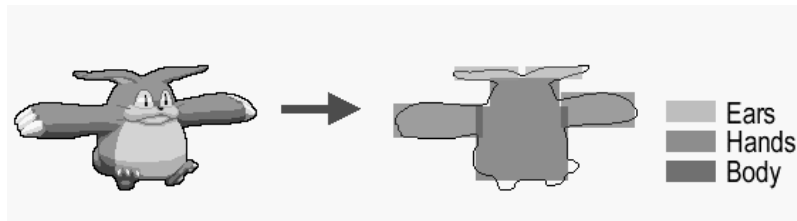


Fig2.2 Define regions for different parts of body

Region is an attribute of *Frame*. It stores the coordinates of a rectangle in the action picture. (*Region* instance is defined as the member under *Frame*.)

The 2 main purposes of region definition:

1. Because most of the pictures have an irregular shape, it is expensive to detect the collision between them. On the other hand, if we define some rectangular regions to represent the picture body, the **collision detection will become simple**.
2. Moreover, **regions can represent** buttons, different parts of a body or other **interactive objects** on the picture.

2.2.7.1 The prototype of our *Class Region*: (written in pseudo-code)

```
class Region
{
protected:
    Integer x, y, w, h;
    /* They are not private because they need to be read by some friend functions for region calculations
    */
};
```

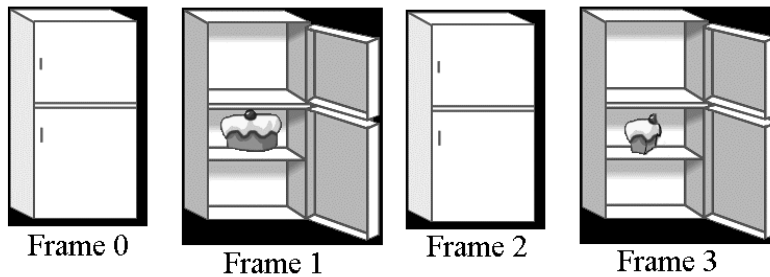
2.2.8 Attachment of other Frame Attributes

Actually, frames' and regions' positions are just some integer data attached in the sprite file. We have also reserved some string and integer in a *Spt* file for future use. These reserved data can have many meanings in future. For example, string can represent a message, or a *.wav file name for the sound effect. An integer can represent the frame rate, or other animating information.

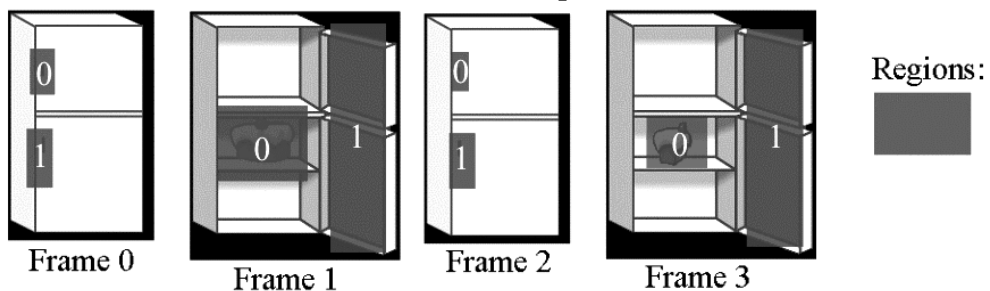
2.2.9 An example of using a Spt file to represent an interactive object:

We want to create a refrigerator using a spt file. The user can open and close the refrigerator by clicking on the side of the doors. There is a cake inside the refrigerator. The user can eat the cake when click on it. But if the cake has been eating once, it cannot be eat again.

The pictures for the refrigerator:



The regions defined for the frames:



We construct the spt file using the technique of constructing a finite state machine.

Here is the DFA for the refrigerator:

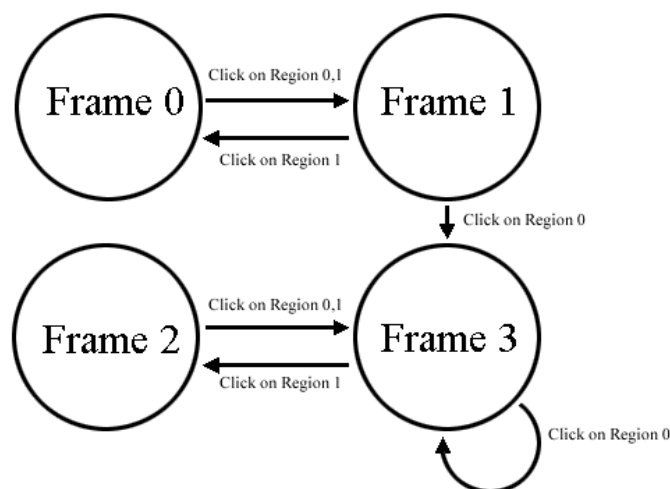


Fig. 2.3 DFA for the refrigerator

Here is the content of the spt file:

```
Sprite_name: "Refrigerator"
No_of_frame: 4
Transparency_color: 0

Frame0:
  No_of_Regions: 2
  Coordinates_of_regions: 20 30 20 30 20 80 20 40
  When_clicking_region0_goto_frame: 1    message: "Open the refrigerator"
  When_clicking_region1_goto_frame: 1    message: "Open the refrigerator"

Frame1:
  No_of_Regions: 2
  Coordinates_of_regions: 24 60 80 70 110 30 40 160
  When_clicking_region0_goto_frame: 3    message: "Eat the cake"
  When_clicking_region1_goto_frame: 0    message: "Close the refrigerator"

Frame2:
  No_of_Regions: 2
  Coordinates_of_regions: 20 30 20 30 20 80 20 40
  When_clicking_region0_goto_frame: 3    message: "Open the refrigerator"
  When_clicking_region1_goto_frame: 3    message: "Open the refrigerator"

Frame3:
  No_of_Regions: 2
  Coordinates_of_regions: 24 60 80 70 110 30 40 160
  When_clicking_region0_goto_frame: 3    message: "I have just eat!"
  When_clicking_region1_goto_frame: 2    message: "Close the refrigerator"
```

2.2.10 Our Direct Sound Class:

We have constructed our **Class of Direct Sound to encapsulate the details of the function calls**. With this Class, we can create and access the objects of Direct Sound easily.

The features of our audio library:

1. **Support** Wav file with **different sample rate**.
2. Different **sounds can be overlap** at the same time.

2.2.10.1 The prototype of our *Direct Sound Class*: (written in pseudo-code)

```
class SoundBuffer
{
public:
    SetupBufferFromWave( String );
                                /* load a wave file into sound buffer */

    Play ( integer );
                                /* play the wave, there is 2 mode of sound playing
                                the normal mode and loop mode.
                                Loop mode are usually use for background music */

    Stop ();
                                /* stop the playing of the wave */

    Static InitializeDirectSound();
                                /* Initialize Direct Sound, This function is static because
                                it function is common to all instance */

    Static ShutdownDirectSound();
                                /* Shut down Direct Sound, This function is static because
                                it function is common to all instance */

private:
    LPDIRECTSOUNDBUFFER soundbuffer;
                                /* the actual sound buffer */

    Static DirectSound_Object DS;
                                /* Direct Sound Object */
};
```

2.2.10.2 Mixer problem in Direct Sound:

Direct Sound supports sound mixer which let different sounds to overlap at the same time. Without mixer, when we want to play a sound buffer, we need to wait until there is no sound playing.

We find that there is a restriction on the Mixer provided by Direct Sound. The same sound buffer cannot be overlapped itself. For example, when we click on a man, there will be a “hello” sound in our program. But if we click on the man many times so frequently, the man will not respond to say “hello” every time. It is because there is a “hello” sounds still playing.

We have considered 3 possible solutions for the Mixer problem:

1. Create duplicate buffers for the same sound file

Advantage:

- a. It is easy to implement.

Disadvantages:

- a. It wastes a lot of memory space.
- b. Don't know how many buffers is enough. Longer sound may need more buffers.

2. Create a constant number of buffers, load the sound file into the buffer only when it needs to play.

Advantage:

- a. It saves the memory space.
- b. It leads to substantial delay of the sound, which is not acceptable.

Disadvantages:

- a. It is difficult to implement.

3. When play a sound buffer, check whether it is being played. If yes, stop the sound and play again from the beginning.

Advantage:

- a. It is easy to implement.

Disadvantages:

- a. If the sound have a lower volume at beginning and higher volume at the end, user may realize that there is a cropping of sound.
- b. If the sound is a human conversation, user may realize that there is a cropping of sound.

Finally, we have use method 3 for the solving the problem.

It is because usually, only sound effect will be overlapped itself.

And Sound effect is always very short and has a higher volume at the beginning.

2.3 DirectShow Library

2.3.1 What is DirectShow?

DirectShow is a Microsoft Windows® API such that it can provide playback multimedia streams from local files or Internet servers, and capture of multimedia streams from devices. Specifically, it enables playback of video and audio content compressed in various formats, including MPEG, Apple® QuickTime®, audio-video interleaved (AVI), and WAV, and both Video for Windows-based capture and WDM-based (Windows Driver Model) capture.

At the heart of the DirectShow services is a modular system of pluggable components called “filters”, arranged in a configuration called a “filter graph”. A component called the “filter graph manager” oversees the connection of these filters and controls the stream's data flow. We will talk about this later.

We are using the Microsoft Visual C++® for DirectShow development

2.3.2 DirectShow Architecture

Here is the DirectShow Architecture:

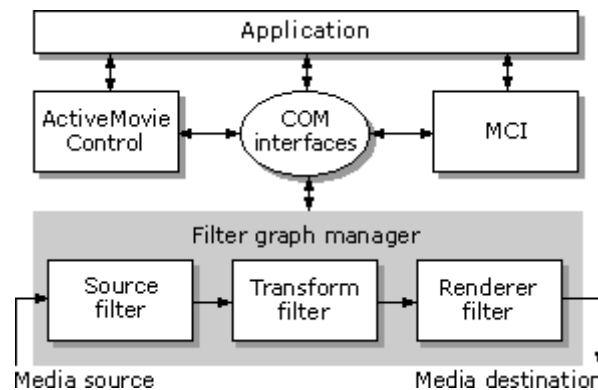


Fig 2.4 DirectShow Architecture

Let's focus on the gray part of the figure. In each DirectShow application, there must be a software component call “filter graph manager” to control the entire “filters”. A graph like the above gray one is called “filter graph”. It is used to describe the data flow within the DirectShow application. A filter graph is composed of a collection of “filters” of different types.

Most filters can be categorized into one of the following three types.

1. Source filter, which takes the data from some source, such as a file on disk, a satellite feed, an Internet server, or a VCR, and introduces it into the filter graph.
2. Transform filter, which takes the data, processes it, and then passes it along.
3. Rendering filter, which renders the data, typically this is rendered to a hardware device, but could be rendered to any location that accepts media input (such as memory or a disk file).

For example, a filter graph whose purpose is to play back an MPEG-compressed video from a file would use the following filters.

1. A source filter to read the data off the disk.
2. An MPEG filter to parse the stream and split the MPEG audio and video data streams.
3. A transform filter to decompress the video data.
4. A transform filter to decompress the audio data.
5. A video renderer filter to display the video data on the screen.
6. An audio renderer filter to send the audio to the sound card.

The following illustration shows such a filter graph.

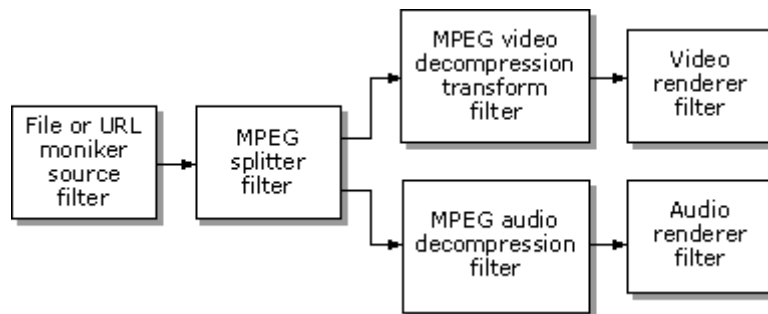


Fig 2.5 Example of the filter graph

2.3.3 What we had done on DirectShow library?

We had tried to unify all those filters into one function called “PlayMMFile” in our project. Also, we try to add some extra functions for those MultiMedia files which DirectShow haven’t provide to us. Here is the function prototype of the function “PlayMMFile”:

```
int PlayMMFile(char * address );
```

Inside “PlayMMFile”, we first try to unify all different source filters. That means we can try to open a MultiMedia file on different kind of source, such as from hard disk or from Internet, by just one function.

Secondly, we try to unify all different transform filters inside “PlayMMFile”. That means we can open different kinds of MultiMedia file format, such as MP3 (audio), WAV (audio), DAT (video), MPG (video), AVI (video), ... etc, by just one function.

Finally, we try to implement two more extra functions to those video files. The functions are

1. To re-size the window that showing a video file
2. To pause to video or audio that are playing

As these two functions are NOT provide by DirectShow, we need to implement it by ourselves.

2.4 WinSock library

2.4.1 What is WinSock?

WinSock is the network application-programming interface (API) for Microsoft Windows Operating System.

For programmers, it provides generic network services; WinSock translates those generic network services into protocol-specifics requests and performs the necessary task. Thus, WinSock shields the programmers from the details of low-level network protocol.

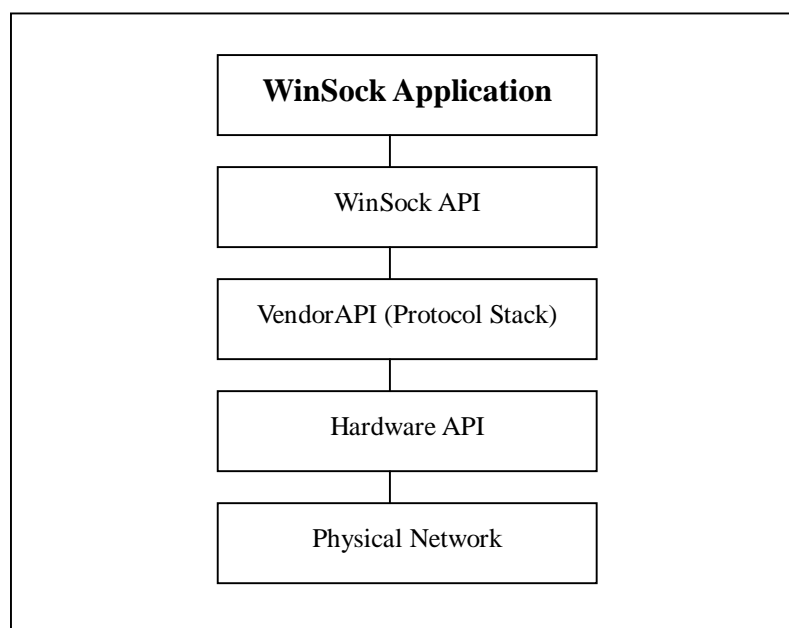


Fig 2.6 Overviews of how WinSock works

2.4.2 The Sockets Programming Paradigm under Windows

Most of the WinSock development follows the Berkeley sockets model. With some exceptions, WinSock includes the most Berkeley sockets API. Most WinSock function names and parameters are identical with the Berkeley sockets library. WinSock offers additional functions that are used to cope with 16-bit Windows system.

WinSock use a client/server approach to communicate. One application is theoretically always available (server side) and another request services as needed (client side).

The server “creates” a socket, name it so that it can be identified and found by a client, and then “listens” for services requests. A client application creates a socket, finds a server socket by name or address, and then “plugs in” to initiate a conversation. Once a conversation is started, data can be sent in either direction.

At application level, both server and client need to know what messages and data expected from the other. They must use the same protocol.

Two fundamental types of client/server application pair exists in WinSock also: connection-oriented and connectionless application. Here is the Overview of Connection-Oriented and Connectionless Application:

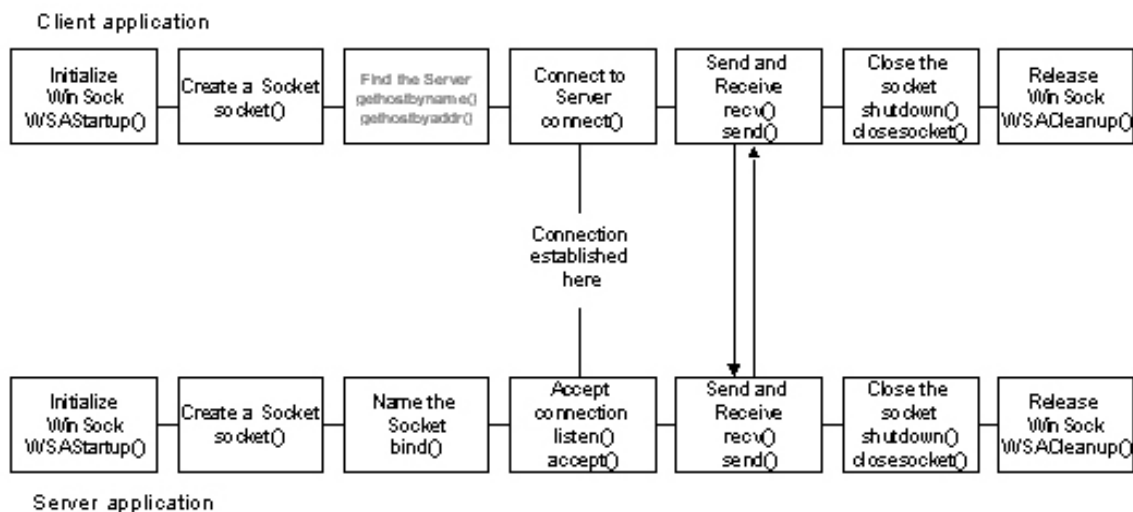


Fig 2.7 Overview of Connection-Oriented Application

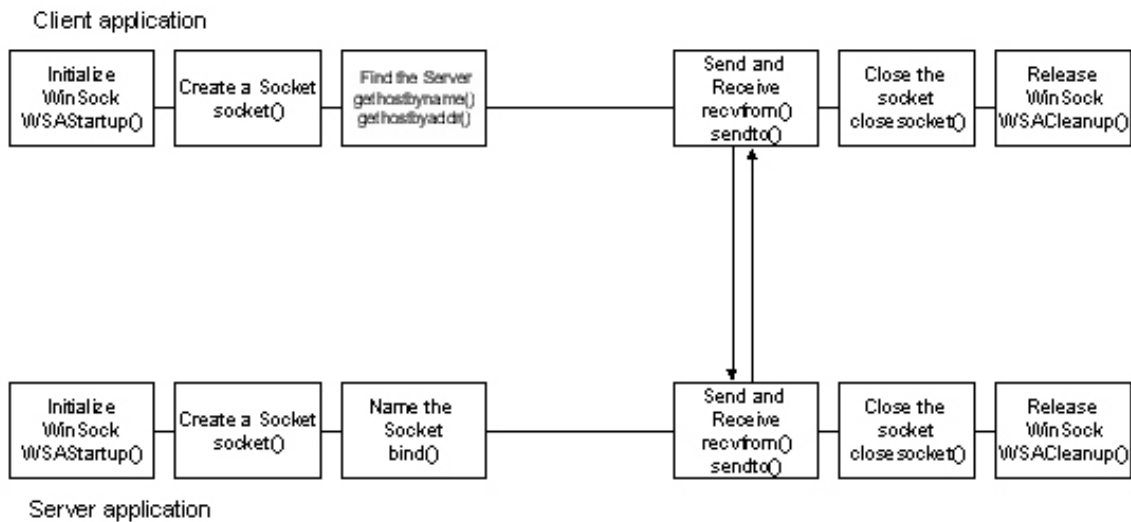


Fig 2.8 Overview of Connectionless Application

2.4.3 Why use WinSock?

Here is some reason to explain why we choose Winsock as our network programming API:

- Multi-protocol support

Winsock allow an application to use the familiar socket interface to achieve simultaneous access to any number of installed transport protocols. WinSock is no longer to use on TCP/IP only.

- Asynchronous I/O and event objects

With Win32 programming environments, WinSock can be extended to asynchronous communication. Asynchronous I/O enables an application to continue with other process while waiting for the I/O operation to complete.

- Quality of Service

The newest WinSock established conventions for applications to negotiate required service levels of communication service such as bandwidth and latency for some quality demanding application such as multimedia communication.

As we can see, WinSock has some advantages over traditional Berkeley socket model under Microsoft Windows®.

2.4.4 Blocking and event-Driven Application

Blocking means that when a function is called, it stops all other processes in the application and does not return until that function is completed. This is not a problem in console application. However, in Windows GUI environment, it is a great problem. Windows GUI are based on **event driven paradigm**. They receive and must quickly respond to all the events (e.g. keystroke, mouse movement ... etc.).

Luckily, the WinSock specification includes many new extensions to the original Berkeley socket API. One of the most important extensions for Win32 programming is the support of **asynchronous mode**.

Asynchronous notification is the best way to doing Win32 GUI applications. Unlike traditional blocking operations, asynchronous operations return immediately, no whether it fail or success. After the operation is finished, the WinSock DLL will send a message to that calling application, indicate the function is finished.

This helps us a lot in designing our program.

2.4.5 Function prototype of WinSock library?

Since the APIs provided by WinSock are too low-level and complicated, we have constructed our own network library to encapsulate the details of the function calls. Here are some prototypes of our own library:

```
class network
{
public:
    int initial_winsock();
    int connect(char * machine_ip);
    int disconnect(char * machine_ip)
    int send(char * data, char * machine_ip);
    int receive(char * data)
    int cleanup_winsock();
};
```

2.4.6 Description of WinSock library

1. **int initial_winsock()**

This function is used to initialize the environment for running WinSock function.

2. **int connect(char * machine_ip)**

This function is used to connect to a computer (of course that computer must running a server program) which is stated in the parameter.

3. int disconnect(char * machine_ip)

This function is used to disconnect to a computer (of course that computer must running a server program) which is stated in the parameter.

4. int send(char * data, char * machine_ip)

This function tries to send data to a particular computer that already connected previously.

5. int receive(char * data)

This function is tries to get data from the buffer. When a machine receive any data from the network, it will first store the data into a buffer. Then the OS will raise an event to tell the application that some data is inside the buffer. When the application get this event, it should call “receive” to get those data.

6. int cleanup_winsock()

This function is use to release resources after the program finished using WinSock function.

2.5 The Server

2.5.1 Overview

In order to allow different user to communicate with each other over a network, a server is need to handle messages (data) passing to and from each client. Moreover, a server can also allow central control of resource to be distributed over the network.

Thus, in our project, there is a server to serve each client of our system.

At the beginning, we have tried to write a very simple server that can provide broadcast function. However, in later stage, we find that a server which can only provide broadcast function is inadequate. So, we try to totally re-write the server to provide more functions for us.

We are now trying to describe that two servers we wrote in detail.

2.5.2 The simple broadcast server

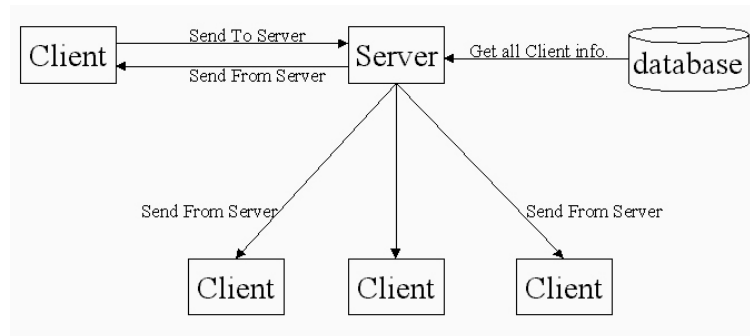
The main function for this server is to allow communication between every client programs.

We try to write a server that stors each client information (name, ID, network address) into list. Later, if there are some data need to send to the clients, the server will broadcast the data to all the clients.

Here is the procedure which a client want to send a message to another client:

1. First, the client must be register to the server. If the client is already registered and not disconnect from the server, this step can be skipped.
2. The client will send the data to the server.
3. The server will check for all the client location from the list of clients.
4. The data is sent to all the clients (broadcast) inside the list of client.

Here is the illustration of the above procedure:



2.5.3 Problems of the simple broadcast server

Actually, this broadcast mechanism is too simple. Here are some disadvantages of this server:

1. Totally no security
2. No fault tolerance
3. Cannot send point-to-point messages

The last problem is the main problem which leads us to construct the improved server. If no point-to-point messages can be supported, this means that each client cannot send message to one of the particular client only. Of course this problem can be solved at application level (by checking the client ID at each client program), however, this will lead to waste of network bandwidth. Thus, we try to provide point-to-point function at the server side.

2.5.4 The Improved server

As we mentioned before, the simple server we developed is too simple. It is not enough for us to use it for our project. This causes us to develop another server, the improved server. Here are some functions that the improved server can provide:

1. Broadcast message
2. Point-to-point message
3. Check each register client is still alive or not
4. Encryption of each message passing to and from

Broadcast message

This function is same as that of the previous server that can provide. When a client send this kind of message to the server, the server will then forward the message to all the clients that had already connected to the server.

Point-to-point message

This function is new function compared with the previous server. The client can send a message to only one of the particular client that is already connected to the server.

Check each register client is still alive or not

This function is new function compared with the previous server. In the list of clients, there is a field called "time counter". When a registered client send an "alive message" to the server, this field will set to 0. On each second, the value of this field of each registered client will be increase by 1. When this value is greater or equal to 10, the server will assume client is dead (the computer hangs, the network connection become unavailable, ...etc). Then, the server will delete the client from the list. Thus, each client need to send an "alive message" to the server within the 10 second period in order to let the server know it is alive. By using this function, the server can save much resource and network bandwidth.

Encryption of each message passing to and from

This function is also new function compared with the previous server. For every message, the message is encrypted with real time generated encryption key. Of course, this method is not very secure in front of professional hackers; however, this can help us to solve several problems:

- 1. Some messages from outside the system accidentally send into the system*

As the outside messages may not be encrypted, when the server or clients get those messages, it will still try to decrypt it. However, after decryption, the message will become meaningless and no action will be take for those messages

- 2. Some messages from the system accidentally send to outside*

If messages from the system accidentally send to outside without encryption, others people may able to see the content of the message. However, with this encryption, if others don't know the encryption key and method of encryption, they can't view the content of the message.

2.5.5 The Improved server Architecture

The network protocol we use in our project is UDP. The reason for we choosing UDP rather than other protocol is that:

1. Compared with TCP, it is much faster
UDP is connectionless and TCP is connection-oriented. UDP runs faster than TCP because there is no need to setup a path before any connection can start. Thus, less pre-connection overhead.
2. With a closed-network, we can assume there is no packet loss
As we mentioned before, UDP is connectionless. That means some packets may be lost during data transfer. However, within a closed-network, we can assume no packet is loss. This can also allow us to use UDP rather than TCP.
3. Compare with IPX/AppleTalk, it can run on Internet
One advantage of UDP is that it can also run on Internet. Thus, our project is not limited to Local Area Network only. But of course, if our project is really try to run on Internet, some change is needed to handle to followings: 1) slow connection speed, 2) large delay time and 3) possible of loss of packets.

2.5.6 Message structure

Also, not like last time, we had defined the message header of different kind of messages passing to the server. Thus, when the server gets those messages, it will know what action should be operated.

There are three main kinds of message will be passed between to server and the clients. They are:

1. System message
The first byte of the message must be ASCII '0' character.
2. Broadcast message
The first byte of the message must be ASCII '1' character.
3. Point-to-point message
The first byte of the message must be ASCII '2' character.

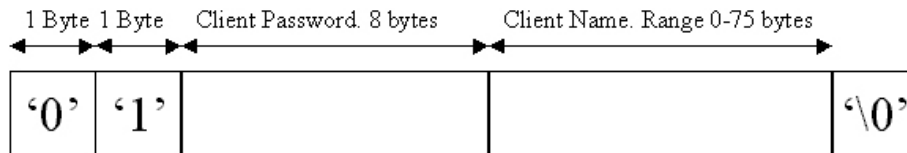
System Message

Several messages will be treated as system message, such as "connect message", "disconnect message" and "alive message" ... etc. The

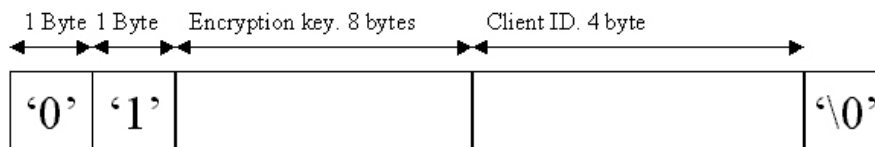
first byte of the message must be ASCII '0' character.

Connect message

When the client want to connect to the server, it must send a "connect message" to the server first. Here is the message structure:



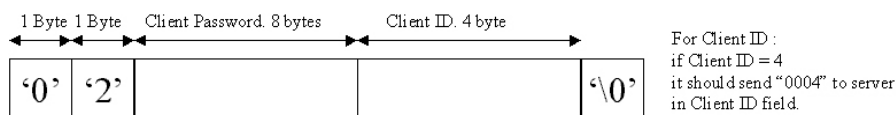
After the server gets this message, the server will check for this client can be connect or not. There are two reasons why this client cannot be connected: 1) the password is incorrect, 2) there are too many client connected to the server already. If the connection is granted, than the server will send back a "granted message" back to the client. Here is the message structure:



One more thing need to notice is that after this message, all messages passing between this client and server will be encrypted with the encryption key.

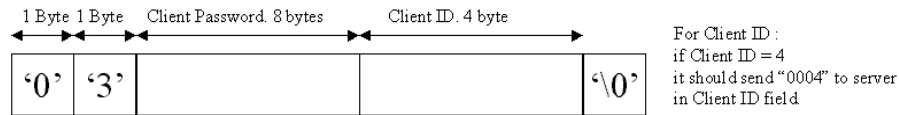
Disconnect message

When the client want to disconnect from the server, it need to send a "disconnect message" to the server. After this, there will be no more message send from the server to this client. Also, after sending this message, the client is no need to wait for the server accept. Here is the message structure:



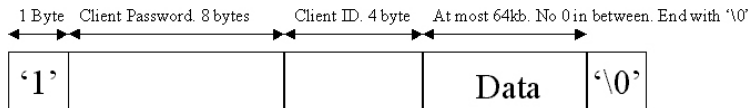
Alive message

By default, the client will send this message to the server at every 3 seconds. This lets the server know that the client is alive. Here is the message structure:



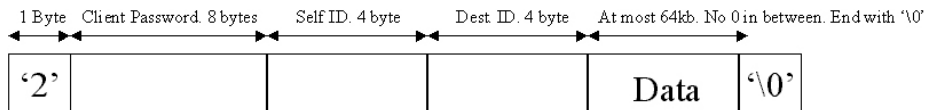
Broadcast message

When the server gets this message, it will forward this message to all the clients. The first byte of the message must be ASCII '1' character. Here is the message structure:



Point-to-point message

When the server gets this message, it will forward this message to the client that is specified at the message. The first byte of the message must be ASCII '2' character. Here is the message structure:



2.5.7 The client list

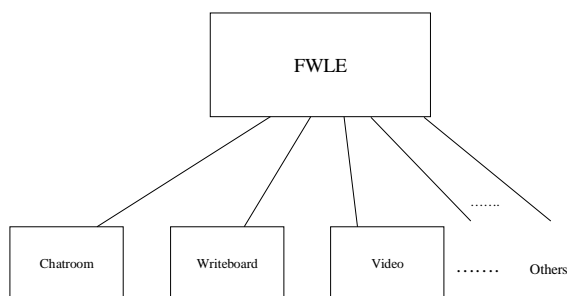
As we mentioned before, the server will contain a list of client. Here is the actual structure of the list of client. (The programming language we use here is C)

```
#define MAX 500
struct {
    int time_counter;           //use to check the client is alive or not
    char encrypt_key[8];       //the encryption key for each client
    char password[8];          //the password for each client
    char client_name[80];      //the client name
    int cleint_id;             //the client ID
    SOCKADDR_IN saClient;     //the client network address
} client_list[MAX];
```

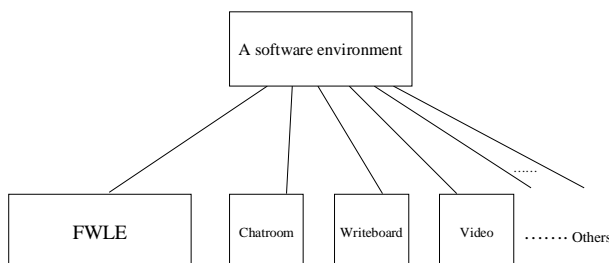
3. Collaborative Environment (CE)

3.1 Introduction

In the first semester, we have built the libraries and tools for developing multimedia network application. And have built an application called FWLE. At the beginning, we want to integrate all the components under FWLE (see the diagram below).



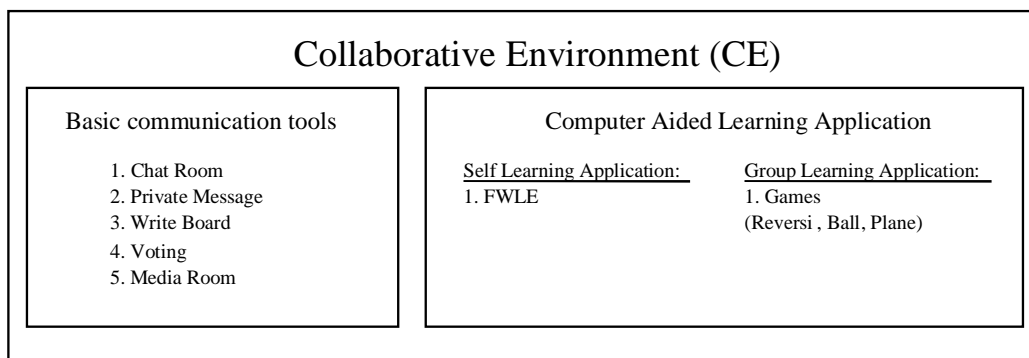
But we found that this approach is not general enough. Since FWLE is just a CAL (computer aided learning) which specified in teaching English. It can be considered as a single component in digital campus only. Therefore, we start to think of a software environment which provide the basic communication tools for teachers/students and the CALs like FWLE can be plugged into this application environment (see the diagram below).



We called such software environment **Collaboration Environment (CE)**. Our aim for building CE is to enhance the learning experiences among, learners in team projects/discussions and allow students who are physically apart to perform joint work.

In our CE, participants can invoke one or more activities (chat room, voting, write board or media sharing). The instance of each activity runs locally at each participant's site and the response of each user is distributed to all the participants.

Our CE's components can be divided into 2 groups – “Basic communication tools” and “Computer Aided Learning Application” (see the diagram below).



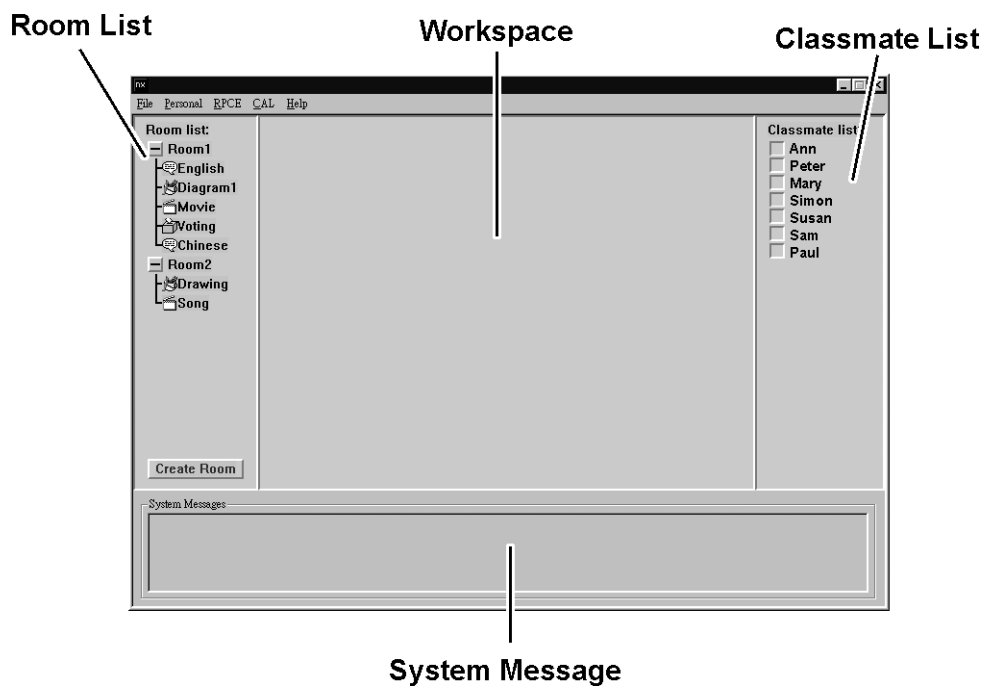
* have not implemented yet.

Basic communication tools is mainly for communications and interactions among teacher/students. It includes chatroom, private message, writeboard, voting and mediaroom. These components will be introduced in the later session.

Computer Aided Learning (CAL) is for students to “self learn” or to “group learn”. FWLE is one of the self learn CAL. We have not designed group learn CAL but we have used some games to demonstrate how can the students to start a group learning application with others.

3.2 Interface

We have built a software to illustrate Collaborative environment, below is a snapshot of its interface.



Workspace

Chatroom, voting, mediaroom and writeboard runs on workspace.

Room List

User can create or join Chatroom, voting, mediaroom and writeboard by using room list.

Classmate List

User can send private message or start group learning CAL using classmate list.

System Message

It will print out the system/warning message from the server.

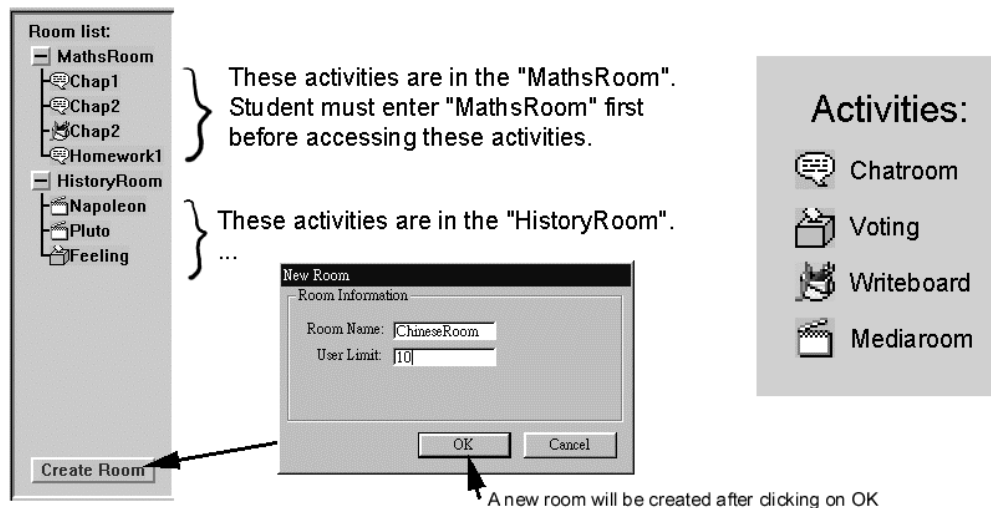
3.2.1 Roomlist

In our software, when a user want to create an activity, he need to find or create a room to hold this activity. If other user want to access this activity, he needs to enter the room first.

This approach has the following advantages:

1. Room can group the related activities together. It facilitates the activity management.
2. Room creator can set a limit for the number of user in a room.
3. Room creator can monitor all the activities in his room.
4. *Room creator can set a password such that only the authorized participants are allowed to access the room.

Interface



Message format for room interaction

| Function | From, To | Format |
|------------------|-----------------|--|
| Create Room | Client -> Admin | Create_room <room_name> <user_limit> |
| Enter Room | Client -> Admin | Enter_room <room_id> |
| Leave Room | Client -> Admin | Leave_room <room_id> |
| Close Room | Client -> Admin | Close_room <room_id> |
| Room information | Admin -> Client | Room_info <room_name> <room_id> <user_limit> <no_activity> <activity_info1> <activity_info2> <activity_info3>..... |

Activity_info

| | |
|-------------|---|
| Chatroom | Chatroom <title> |
| Voting | Voting <title> <question> <choice1>...<choice5> <number1>...<number5> |
| Write Board | Writeboard <title> |
| Media Room | Mediaroom <title> <media_name1>...<media_name5> <URL1>...<URL5> <user_limit1>...<user_limit5> |

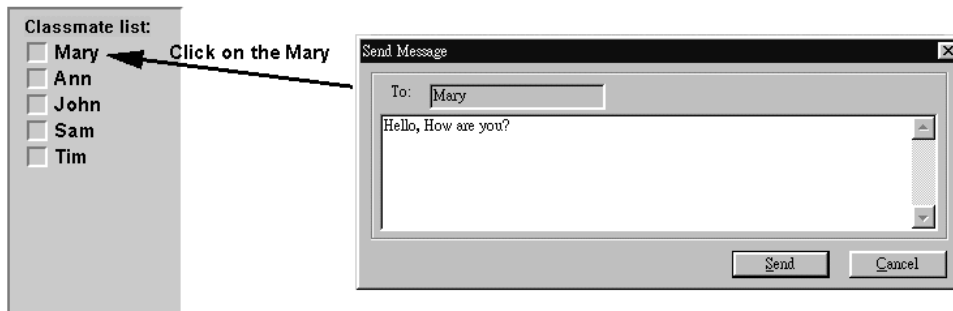
* (have not implemented yet)

3.2.2 Classmate List

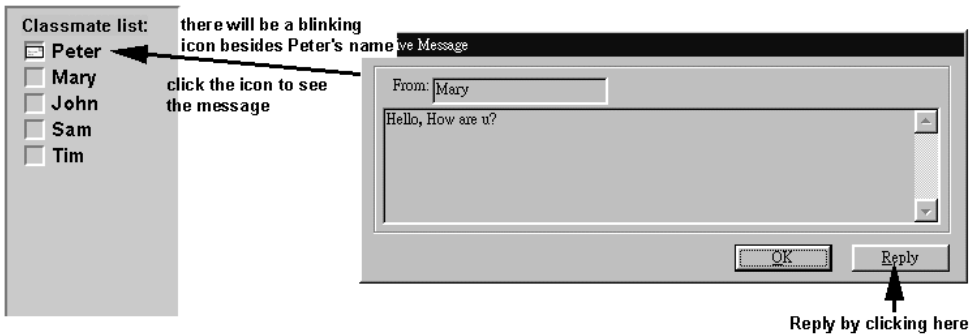
A participant can send private message to classmate by clicking on the name of him on the name list (at the right side of screen). This function is very useful when a participant want to discuss with a person but do not want other to know.

Interface: (Peter sends a message to Mary)

At Peter's Side



At Mary's Side

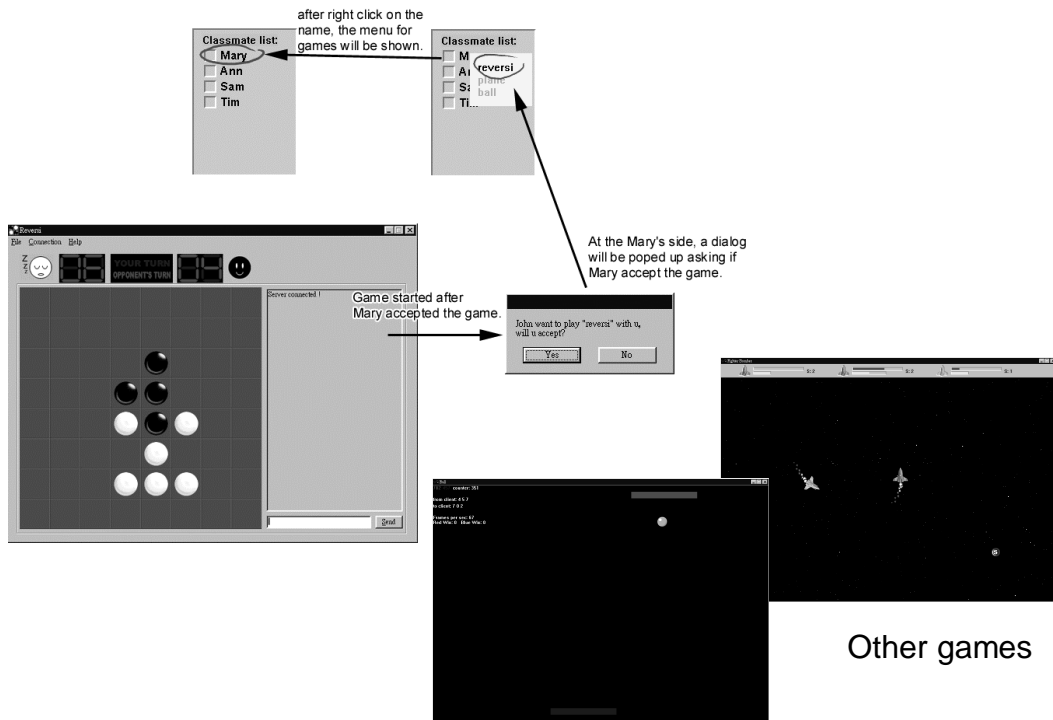


Message format for private message

| Function | From, To | Format |
|----------------------|--------------------|------------------------------------|
| Send private message | Client1 -> Client2 | Private_msg <message_id> <message> |
| Acknowledgement | Client2 -> Client1 | Private_msg_ack <message_id> |

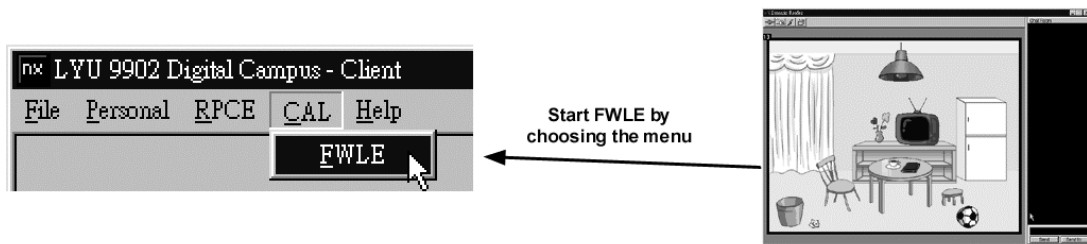
3.2.2 Classmate List (Cont'd)

User can start group learning CAL using classmate list. (Since we have not designed any group learning CAL, we just use games to illustrate how to start a group learning CAL)



3.2.3 Using CAL (self learning application)

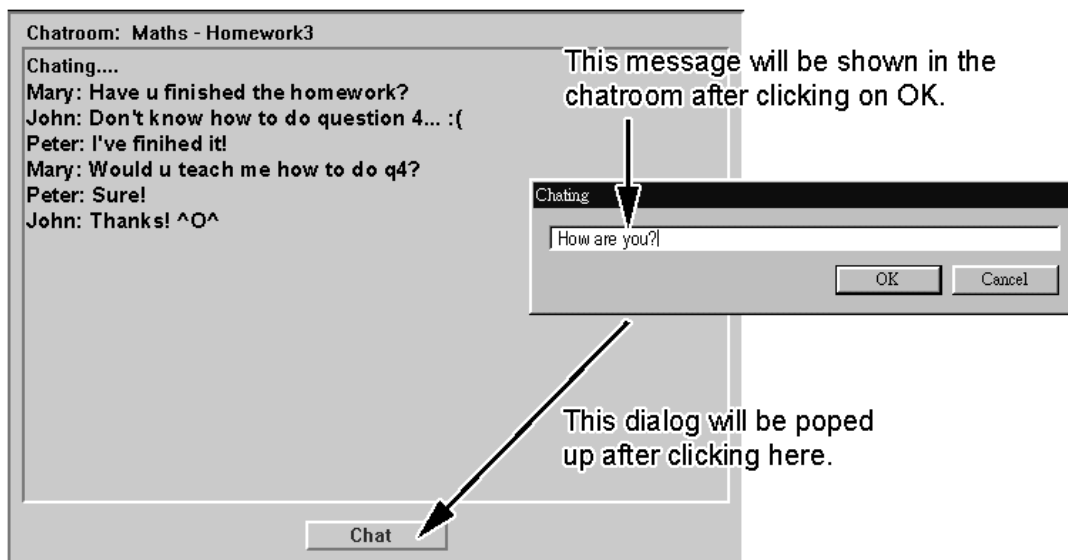
User can start self-learning CAL by choosing the menu. Below is an example:



3.2.4 Chatroom

Obviously, sending private message is not a efficient way for group discussion. In our CE, user can create a Chatroom which help them to discuss with a group of people.

Interface:



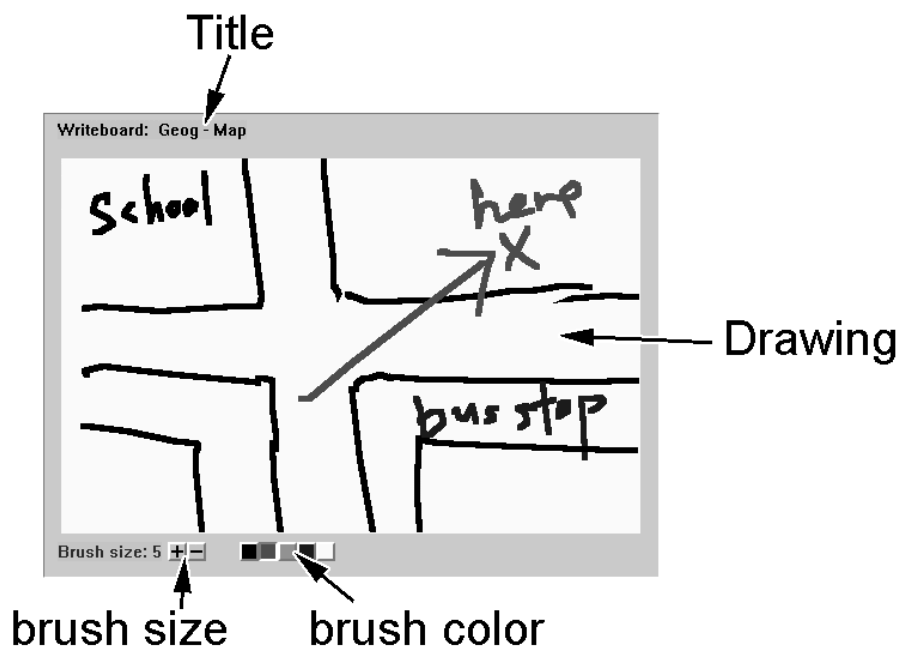
Message format for chatroom

| Function | From, To | Format |
|------------------|------------------|--|
| Create ChatRoom | Client -> Admin | Create_chatroom <room_id> <title> |
| Close ChatRoom | Client -> Admin | Close_chatroom <room_id> <activity_id> |
| ChatRoom Message | Client -> Client | Chatroom_msg <room_id> <activity_id> <message> |

3.2.5 Writeboard

Sometimes, participant may need a drawing to express their idea in a discussion. Writeboard allow them to show their drawing to others. In the past, when a teacher have drawn something on board, he need to wait the students to copy his drawing. In this situation, Writeboard is very useful. When teacher is drawing on its computer, all the student can see it immediately. *Moreover, student can save the drawing to harddisk by clicking on a button only.

Interface:



Message format for writeboard

| Function | From, To | Format |
|-------------------|------------------|---|
| Create Writeboard | Client -> Admin | Create_writeboard <room_id> <title> |
| Close Writeboard | Client -> Admin | Close_writeboard <room_id> <activity_id> |
| Draw Writeboard | Client -> Client | Draw_writeboard <room_id> <activity_id> <x1> <y1> <x2> <y2> <radius> <color> |

3.2.5 Writeboard (cont'd)

Bresenham Line Algorithm[2]:

Since we want to save the bandwidth, we will not send the whole bitmap to all other participants when updating the whiteboard. We will cut the movement of the mouse into line segments and send the coordinates of the lines only.

We use Bresenham Line Algorithm to draw the lines. Below is the pseudo code:

Input : Pair of integers x_1, y_1, x_N, y_N $\Delta x = x_N - x_1, \Delta y = y_N - y_1$
 Output : Pairs of integers $\{(x_i, y_i)\}$

Minimizing error $\|y_i - (y = mx_i + b)\| (m \leq 1)$
 i.e. closest to line $y = mx + b$; $x \mapsto y$ for $m > 1$

Data Structures : $\begin{cases} |m| \leq 1 & x \text{ is monotonic} & \Rightarrow x_i = x_1 + (i - 1) \\ |m| > 1 & y \text{ is monotonic} & \Rightarrow y_i = y_1 + \text{sign}(\Delta y)(i - 1) \end{cases}$

Error measure

$$P_1 = 2 \Delta y - 2 \Delta x$$

$$P_{i+1} = P_i + 2 \Delta y - 2 \Delta x (y_{i+1} - y_i)$$

Algorithm :

```

    Swaps so  $\Delta x > 0$ ;
    If  $|m| > 1$  (or  $\Delta y > \Delta x$ )
        swap roles of  $(x, y), (m, 1/m)$ 
    Initialize :  $P = 2 \Delta y - \Delta x$   $y = y_1$ 
    Loop for  $x$  from  $x_1$  to  $x_N$ 
        Begin
            Draw(  $x, y$  )
            Set  $y_{old} = y$ 
            If  $P < 0$  set  $y = y_{old}$ 
                else set  $y = y_{old} + \text{sign}(\Delta y)$ 
            Set  $P = P + 2 \Delta y - 2 \Delta x (y - y_{old})$ 
        End
    
```

The advantage of this algorithm is that it deals with integers only (no floating points) so that its performance is very fast.

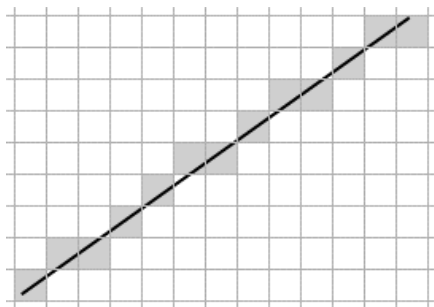


fig 3.1 the line

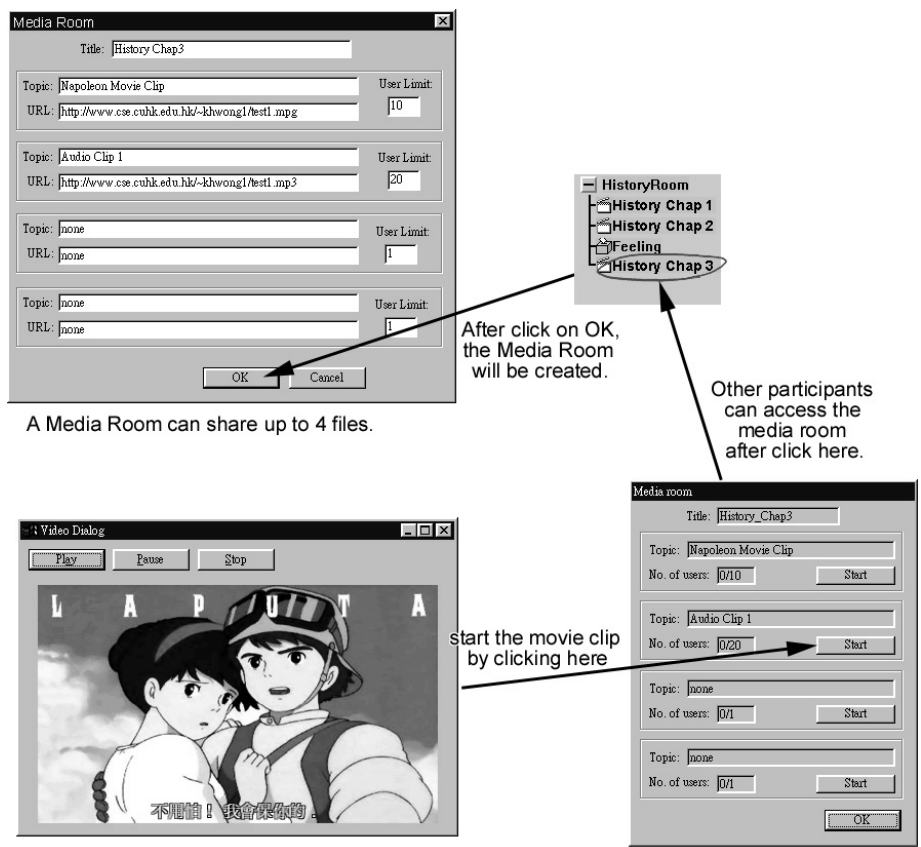
Pixel positions along the line path plotted with Bresenham's line algorithm.

3.2.6 Mediaroom

Teacher can share a media file to a group of students by creating mediaroom. Our media room supports video files (MPEG, AVI, DAT) and audio files (MP3, WAV).

Sometimes, teacher may feel boring such that they always do the thing repeatedly in the same lesson but in different classroom. In this situation, teacher can record some video clips by themselves and share it to the students. They can then spend more time on answering the question by students or concerning the students' response. Moreover, student can see the video clips as many times as they want. They can be reminded if they have something missed on the lesson.

Interface:



Message format for mediaroom

| Function | From, To | Format |
|------------------|-----------------|--|
| Create Mediaroom | Client -> Admin | Create_mediaroom <room_id> <title> <media_name1>...<media_name5> <URL1>...<URL5> <user_limit1>...<user_limit5> |
| Close Mediaroom | Client -> Admin | Close_mediaroom <room_id> <activity_id> |
| Play Mediafile | Client -> Admin | Play_mediafile <room_id> <activity_id> <file_no> |
| Finish playing | Client -> Admin | Finish_mediafile <room_id> <activity_id> <file_no> |

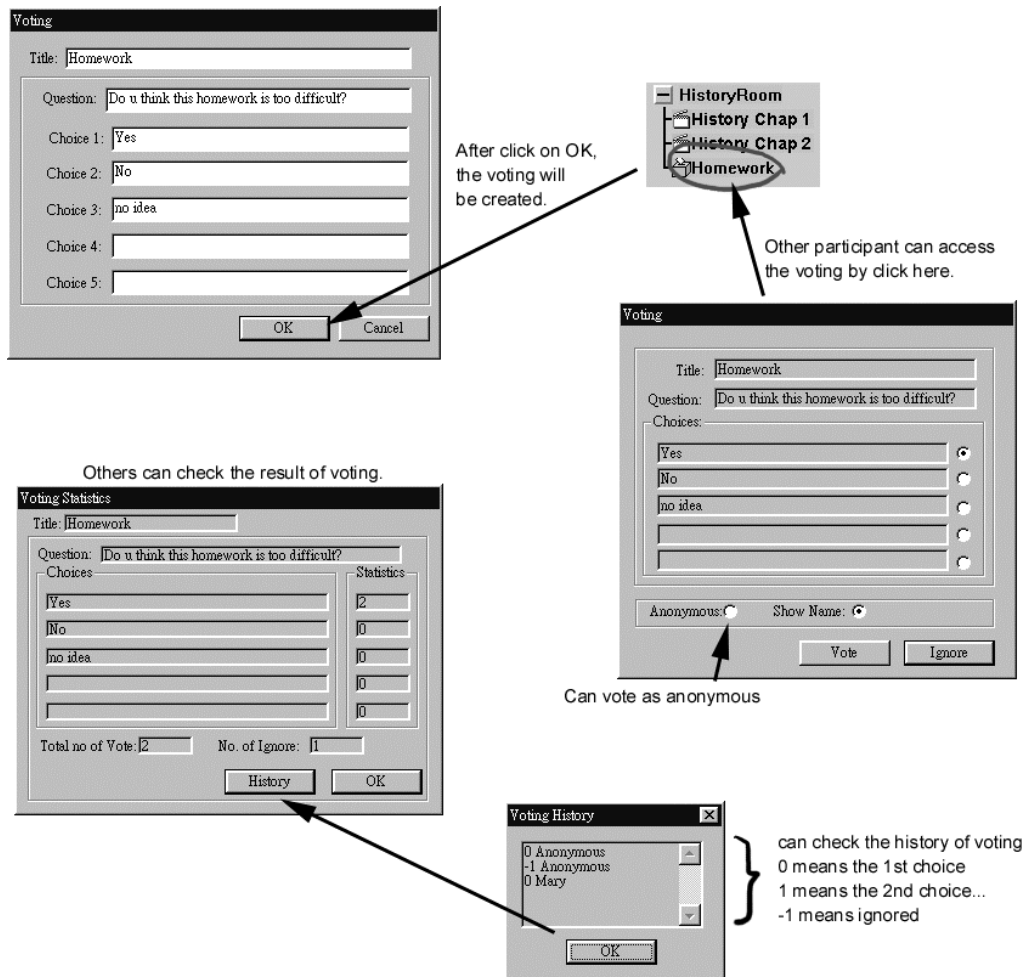
3.2.7 Voting

Sometimes, when a group of people want to make a decision, voting will be used.

Voting is also useful when a student want to make a statistics on the opinion/feelings of others.

In our voting, the vote holder can check the voting progress in voting history. Voters can choose anonymous mode if they want to hide his name in the voting history.

Interface



Message format for voting

| Function | From, To | Format |
|-----------------|-----------------|---|
| Create Voting | Client -> Admin | Create_voting <title> <question> <choice1>...<choice5> <number1>...<number5> |
| Close Mediaroom | Client -> Admin | Close_voting <room_id> <activity_id> |
| Vote | Client -> Admin | Vote <room_id> <activity_id> <choice> <name> |

4. Role Play Collaborative Environment (RPCE)

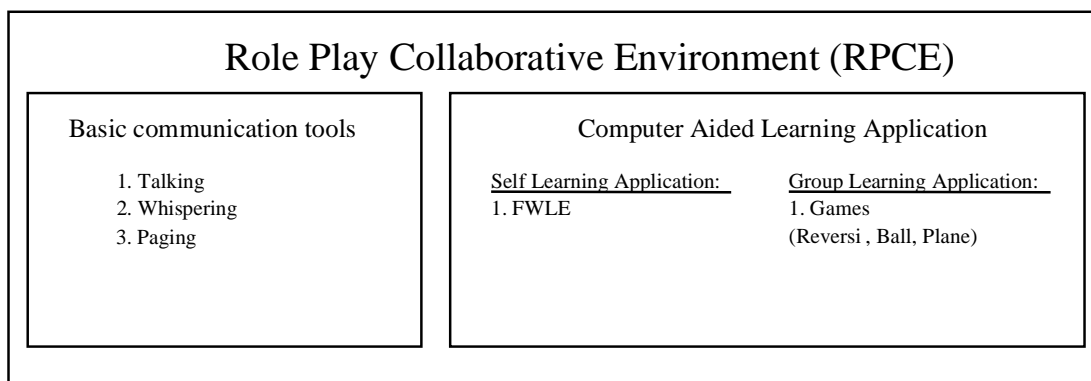
4.1 Introduction

The Collaboration Environment (CE) we mentioned in the previous chapter is a traditional menu driven application. Actually, we think that menu driven style interface may not be suitable to all kind of students. Young student or old teacher may feel difficult in adapt to menu driven application. Sometimes, they feel so confused in dealing with so many buttons, windows and dialogs. Moreover, There are student categories, which face problems in adapting to network-based education because their learning paradigms require in-class (social) interaction and discipline. It is estimated that at least 30% of the student population fall into this category [1].

Therefore, we have built RPCE which try to simulate the learning environment of the real world and help the students in this student category to learn. The control method of RPCE is far simpler than menu driven applications. User can control all the operation by simply pressing the arrow keys and a few buttons only.

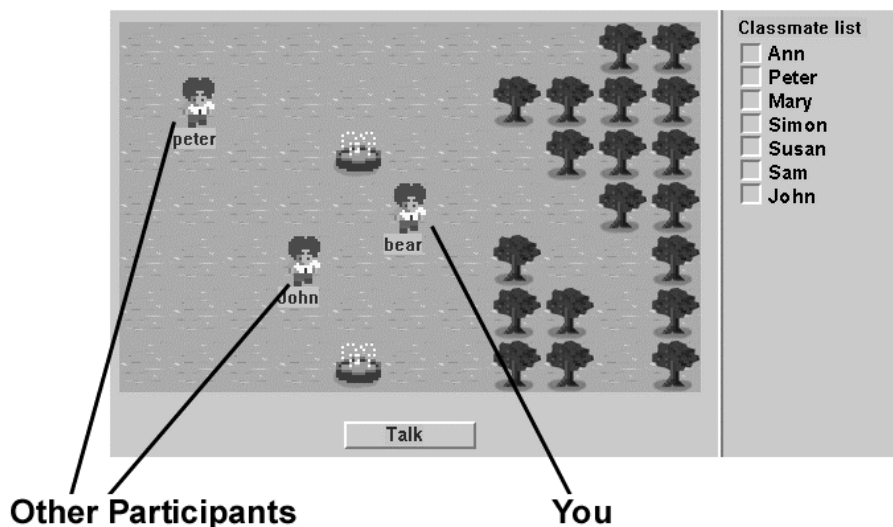
The application structure of RPCE is similar to CE's (see the diagram below).

The only difference is RPCE's basic communication tools consists of Talking and Private Message only.

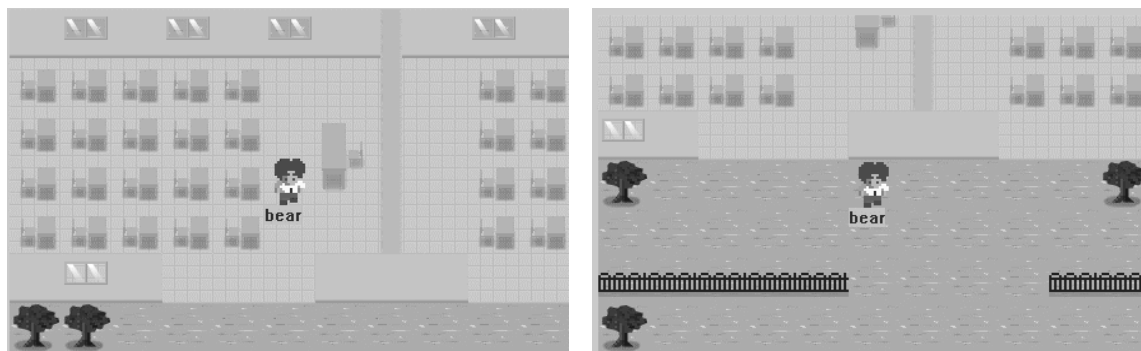


* have not implemented yet.

4.2 Walking



In RPCE, each participant control a character in a 2D virtual world. He can control his character to walk around or to interact with other participants. Each user’s response will be distributed to others and they can see each other in their screen.



(Screen can only show a part of the map, so when the user walk, the map will scroll such that the character remains in the central position of the screen)

Message format for walking in the map

| Function | From, To | Format |
|--|-----------------|----------------------------------|
| Client tells admin he moves. | Client -> Admin | Map_walk <dx> <dy> |
| Admin tells client one of the client’s position. | Admin -> Client | Map_position <client_id> <x> <y> |

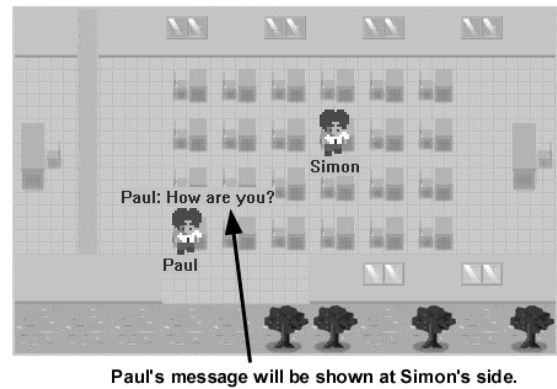
4.3 Talking

If a participant want to talk with others, he can click on the “Talk” button below. A dialog will then popped up for him to enter a message. After entering the message, the participants in the same screen with him will be able to see his message.

At Peter's Side:



At Simon's Side:



This function is called “talk” because only the participants in the same screen will be able to “talk” with each other. It is like the real world. If 2 participants are too far apart, their voice will not be loud enough to talk with each other.

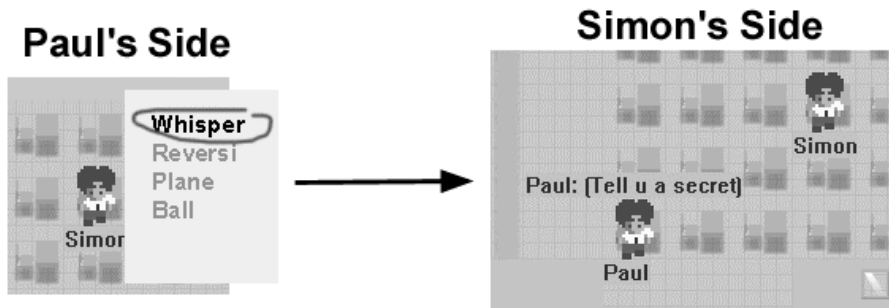
Message format for talking in the map

| Function | From, To | Format |
|---|-----------------|------------------------------------|
| Talk in the map. | Client -> Admin | Map_talk <message> |
| Admin tells client one of the client is talking | Admin -> Client | Map_one_talk <client_id> <message> |

4.4 Whispering

If a participant want to talk with a specified person, he cannot use “Talk”. It is because in “Talk”, all the people in the same screen can see his message.

By using “Whispering”, participant can choose a target in the screen to send him a message such that only the target can see his message.



User can whisper by simply clicking on other participant in the screen and then choose “whisper”.

Message format for whispering in the map

| Function | From, To | Format |
|------------|------------------|--------------------------|
| Whispering | Client -> Client | Map_whispering <message> |

4.5 Paging (Private Message)

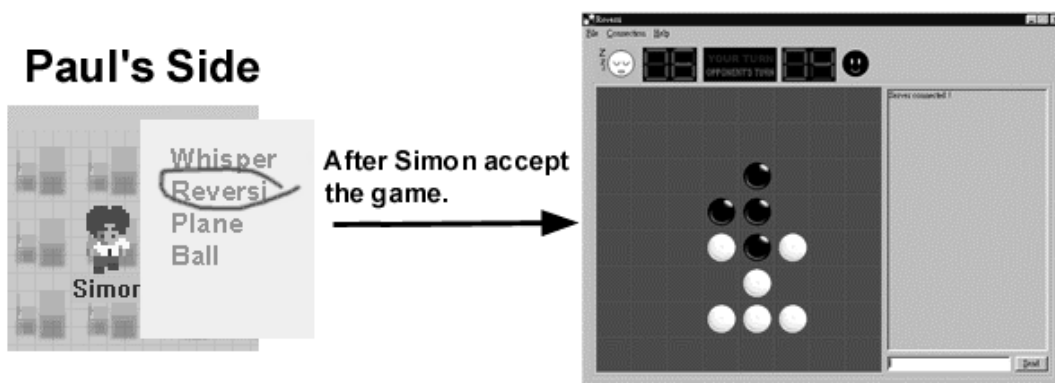
“Whispering” and “Talking” is sometimes no use when a participant want to talk to someone but he is out of the screen. By using “Paging”, participant can send message to a person wherever he is. Actually, the usage of “Paging” is same as sending private message in CE (see page 34). User can page someone by clicking his name on the classmate list at the right of the screen.

Message format for paging in the map

| Function | From, To | Format |
|----------|------------------|----------------------|
| Paging | Client -> Client | Map_paging <message> |

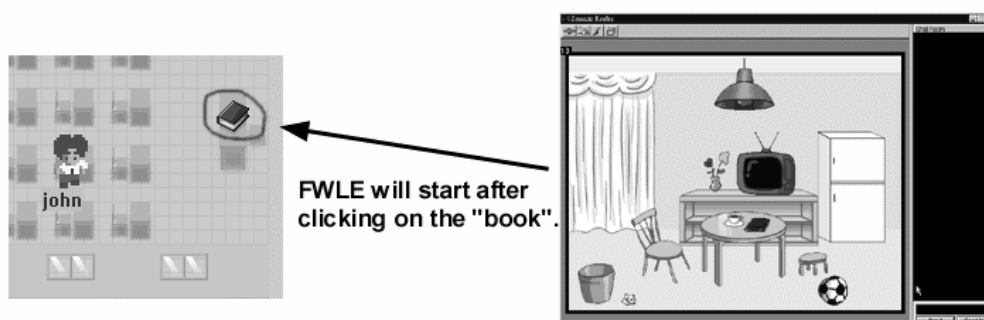
4.6 Playing games (group learning application)

User can start group learning CAL by clicking on other's character in the screen or clicking on other's name on the classmate list. (Since we have not designed any group learning CAL, we just use games to illustrate how to start a group learning CAL)



4.7 Using CAL (self learning application)

In RPCE, there are some objects on the map such that when the user click on it, a specified program will run. CAL is started by using this mechanism. There is an example below:



4.8 Client-Server interaction

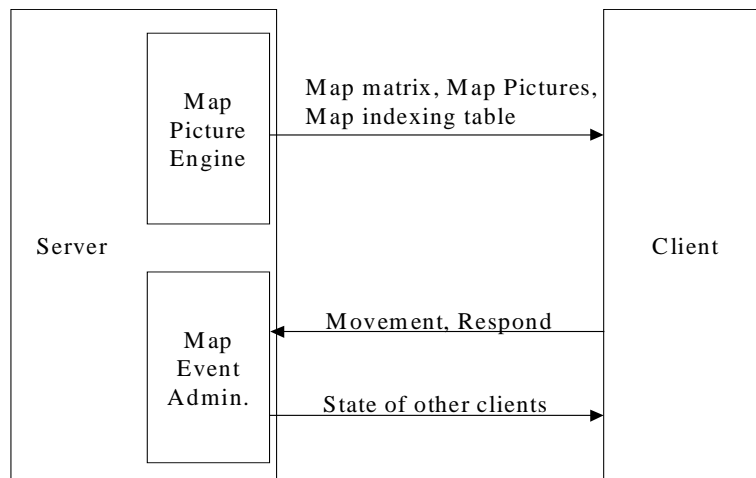
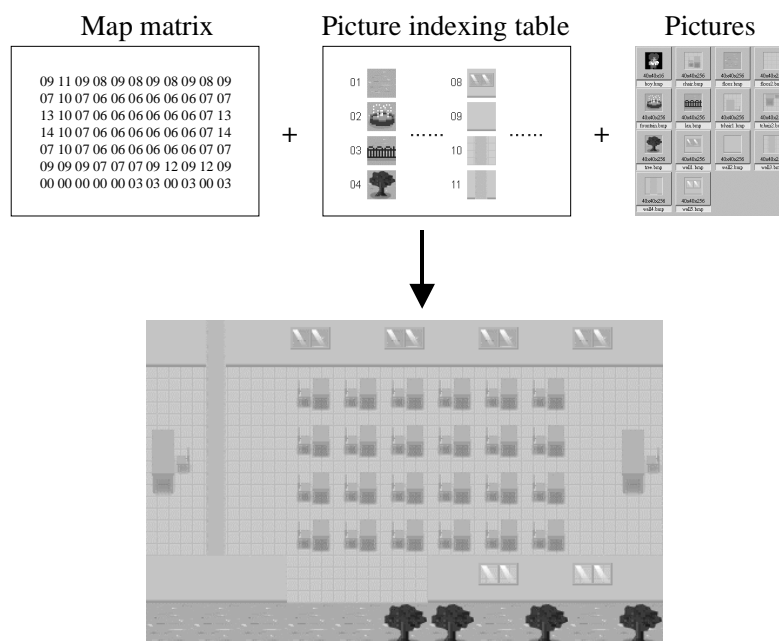


Fig 4.1 Client-server interaction in RPCE

In our RPCE, all the map information (included the pictures) are stored in the server. Every time when the client is logged in, they will ask server for the map information and nothing will be stored in disk locally.

The advantage of this approach is that update of the map can be done in the server only. Otherwise, different version of the client program may have different map.

When the client wants to walk/talk, they will send their request to the administrator, afterwards, the administrator will calculate the new coordinates/state of the client and send them to all of the participants.



Map information includes the map matrix, picture indexing table and the pictures

4.9 Picture Compression

4.9.1 Overview

All the pictures of the map in the **Role Play Collaborative Environment** are need to send to the client from the server at each time the client login. This can prevent inconsistency between each client's map, as the map's picture may be changed as needed. (For example, the administrator of the system wants to set up one more classroom in the map)

If we do not compress the pictures before sending, this will leads to 2 problems:

- 1, Waste of bandwidth
- 2, Large size of data will be causes an overhead in cropping and merging for data transfer.

Thus we apply some compression algorithm into our project.

4.9.2 Our approach

Before we start, we must first choose one of the compression skills to apply.

There are several compression algorithm, some are

1. Lossless, which means no data loss after the compression and decompression step, and some are
2. Lossy, which means some data loss is allow (to order to get higher compression rate) after the compression and decompression step

After we discuss, we design to choose one of the "lossless" algorithms. The reason is that the compression rate of these "lossless" algorithms is already good enough for our project. Two of the "lossless" algorithms we had try, they are:

- 1, Huffman Code
- 2, LZW

4.9.2.1 Huffman Code

A simple method that generates a kind of prefix code.

Here is the Algorithm (bottom-up approach):

1. Assume the probabilities (frequencies) of symbols used are known
2. Label each node with its correspondence probability and put all nodes into the candidate list.
3. Pick two nodes with smallest probabilities, create a parent to connect them and label this parent with the sum of probabilities of these two children.
4. Put the parent node into the candidate list.
5. Repeat the last two steps until one node (root) left in candidate list.
6. Assign 0 and 1 to left and right branches of the tree in the candidate list.

4.9.2.2 Disadvantages of Huffman Code

Although Huffman Code is very easy to implement, however, it still has some disadvantages.

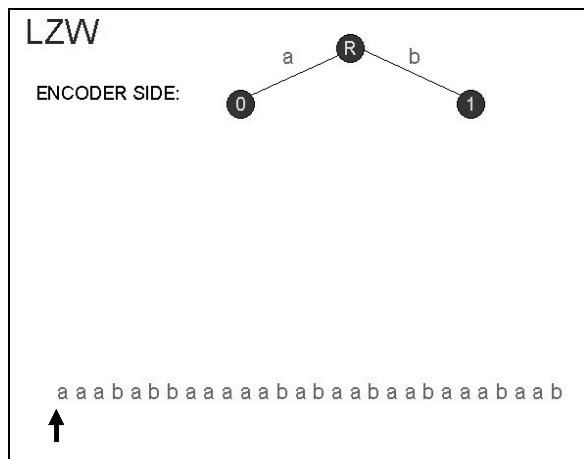
1. A table is needed that lists each input symbol and its corresponding code.
2. Need to know the character frequency distribution in advance => need two passes over the data.
3. More seriously, it does not explore the coherence between symbols. You cannot group a set of symbols and output one single code for them, e.g. pattern "the" is usually used in English

For our experience, using Huffman Code for our project, the compression is usually 1.5–2.0:1, which is not good enough. That's why we try the others method: LZW.

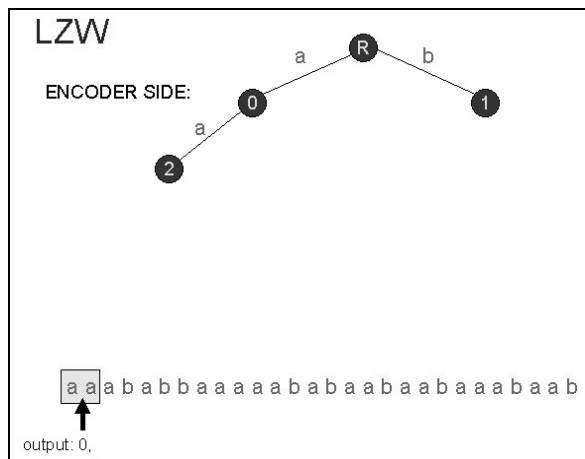
4.9.2.3 LZW

Huffman code cannot encode multiple input symbols by a single codeword. Hence, a lot of patterns (e.g. “the”, “of”, “an” in English are regular patterns) are frequently seen. However, in LZW, we can build a dictionary of all frequently seen patterns and encode them with the table index. Let see the example:

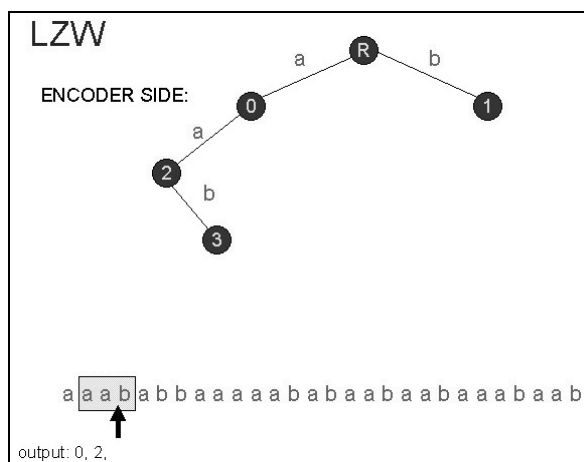
Assume an input source with only two symbols “a” and “b”. $S = \{a, b\}$. Let’s further assume the input data stream is “a a a b a b b a a a a b a b a a b a a b a a a b”. The initial LZW tree contains root, a-descendant and b-descendant.



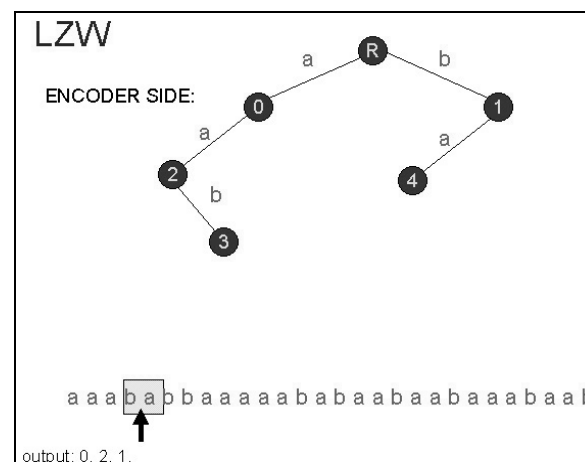
Step 1



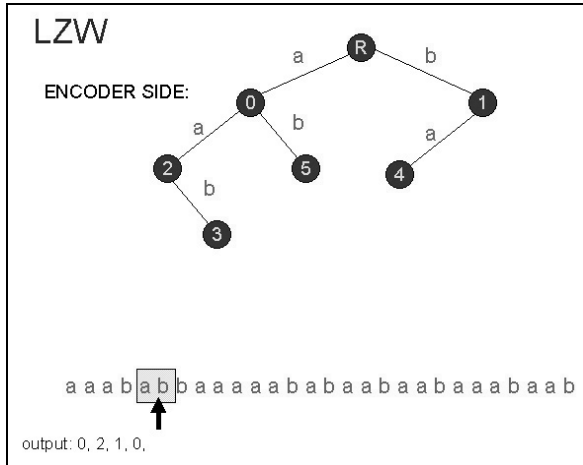
Step 2



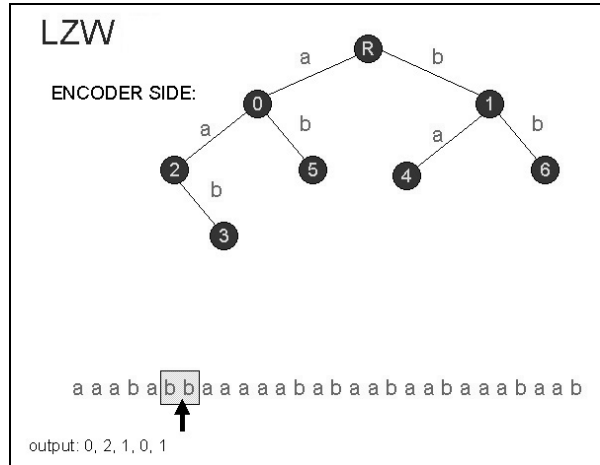
Step 3



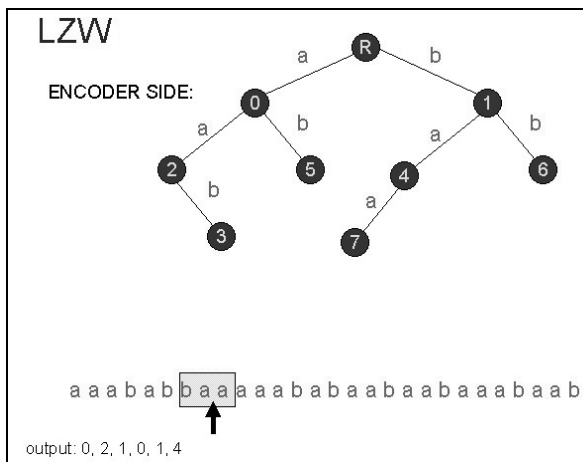
Step 4



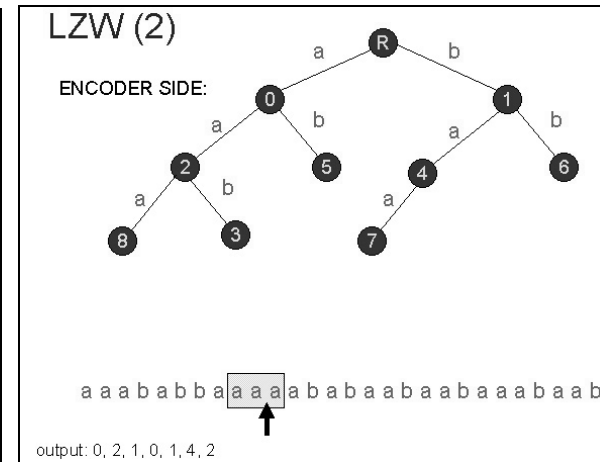
Step 5



Step 6



Step 7

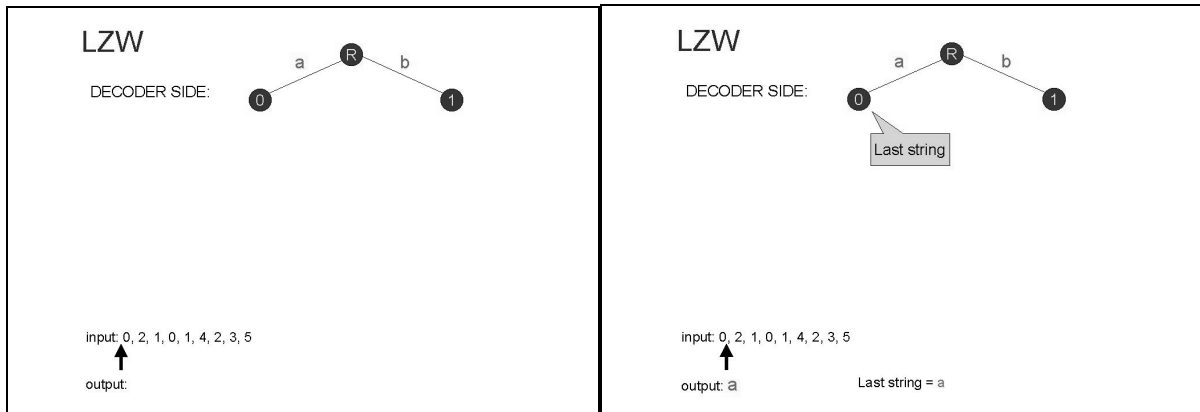


Step 8

Thus, by using this method, the output will finally become “0, 2, 1, 0, 1, 4, 2, 3, 5 ... “

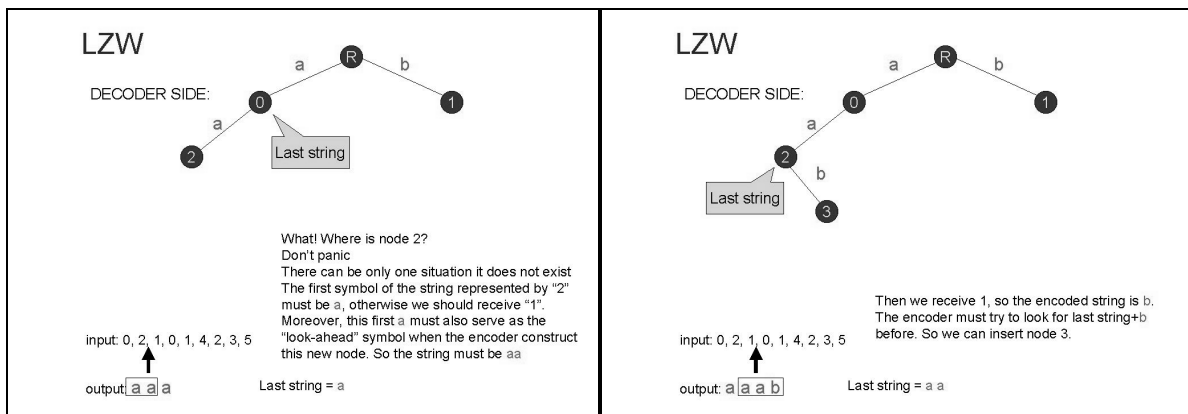
Note the output only contains codewords, no symbol is send in uncompressed form. Let’s see how can we decode. This will be a little bit tricky. Again we start with a LZW tree containing 3 nodes. Let’s us a string variable “Last string” to memorize the last decoded string.

For decompression,



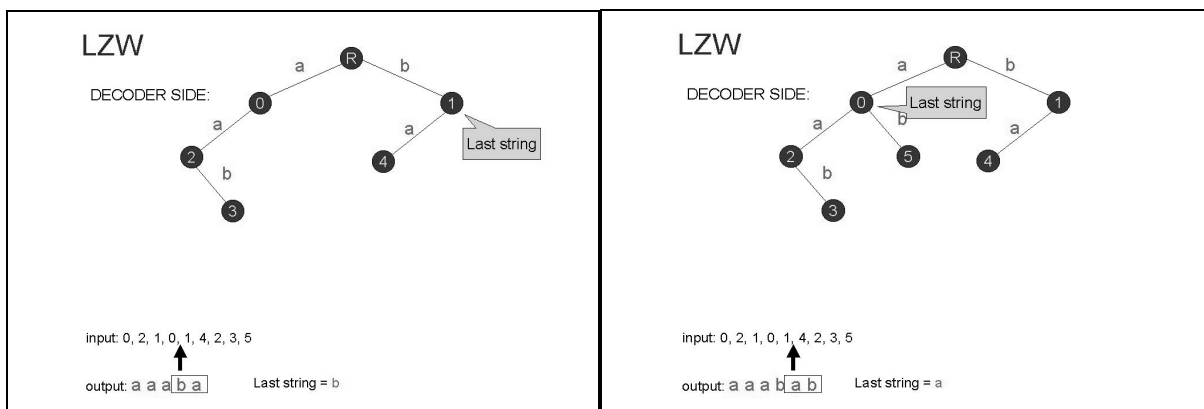
Step 1

Step 2



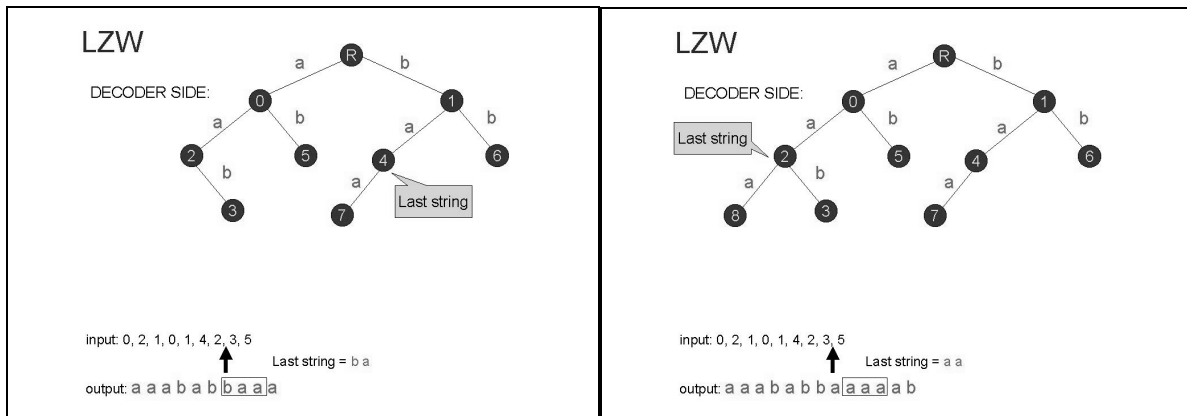
Step 3

Step 4



Step 5

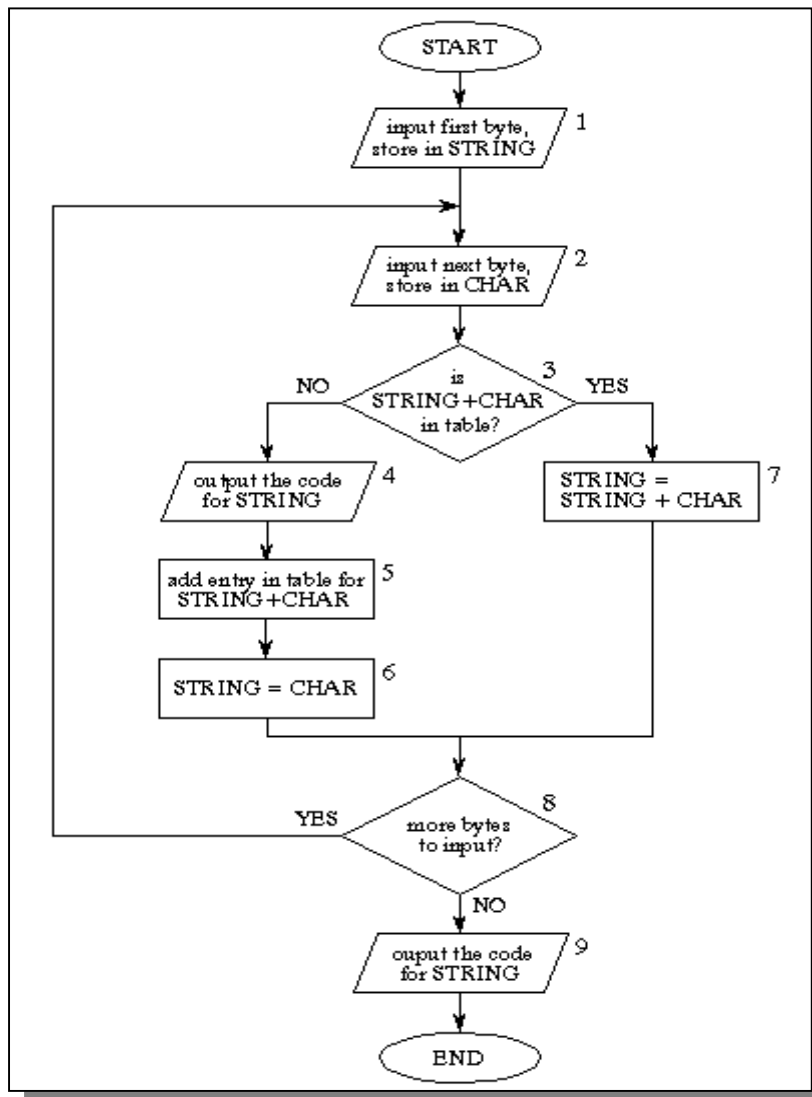
Step 6



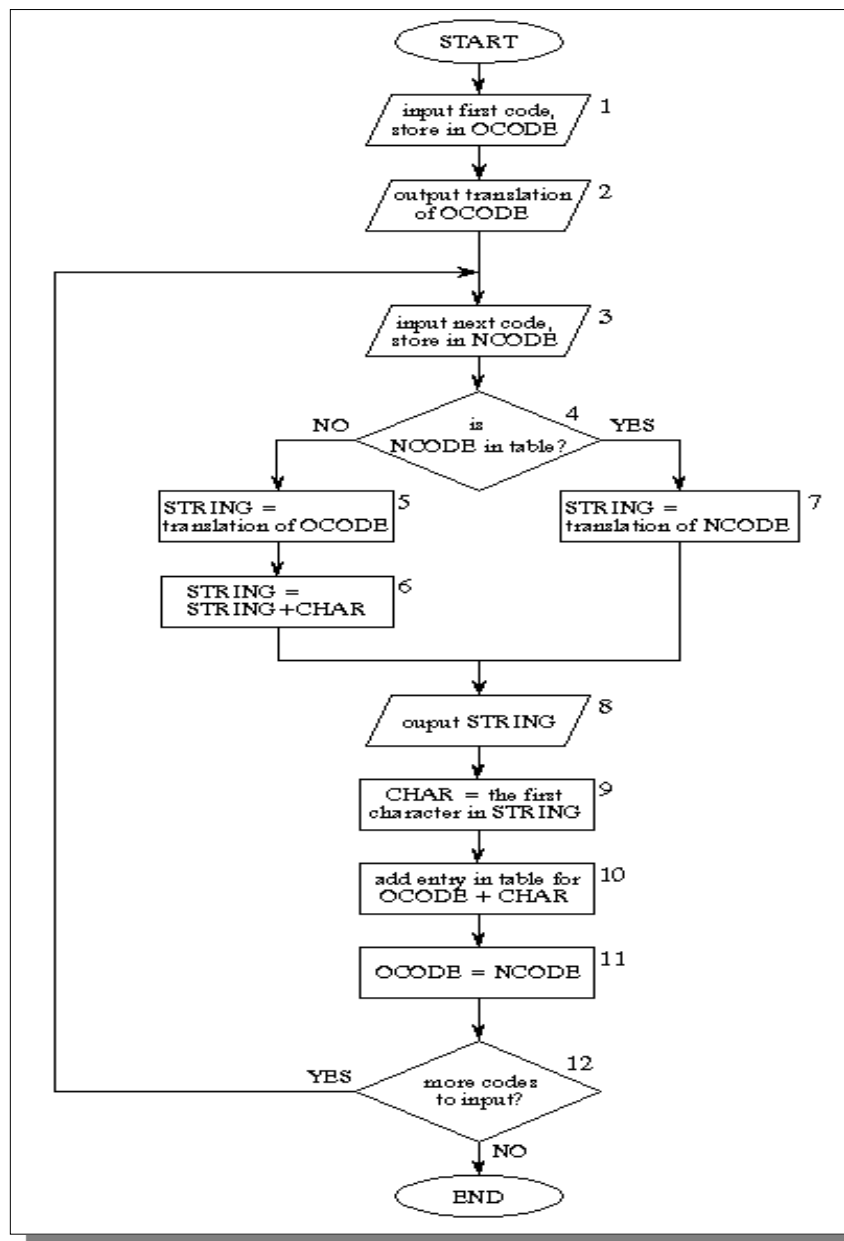
Step 7

Step 8

And, here is the algorithm outline of the LZW method (encoder):



Here is the algorithm outline of the LZW method (decoder):



4.9.2.4 Conclusion

For LZW, we can usually get the compression ratio to about 3-4:1

Finally, we choose LZW as our final algorithm to help us to compress those pictures.

4.10 Further Improvement

Compared to Menu Driven Collaborative Environment, Role Play Collaborative Environment is far more user friendly. The communication method in RPCE is similar to the actual world. Young students or Old teacher will find it is easy to understand/ use the communication methods.

On the other hand, since RPCE is our new idea, it still has so many weakpoints: The communication method in RPCE is weaker than the Menu Driven one. It does not support voting, writeboard and media room. It is because such tools are too complicated to be operated by a simple interface. Moreover, the communication method in RPCE is not so efficient such that most of the functions require the target in the same screen as user.

We have thought about some ideas to improve our RPCE. One of them is the “paging” function. By choosing the target from the name list, the participant can send message to anyone wherever he is.

Another improvement can be made is that the whole map can be reduced into a smaller one and shown at the side of the screen. In this small map, each character will become a small dot. It can help the students to search each other and explore the map. The teacher can also monitor the activity of all the students by looking at this small map.

In Menu Driven Collaborative Environment, student can create rooms to hold the activity. In RPCE, it is also possible to let participants to create their own area such that when others enter their area, they should follow the instructions set by the area owner.

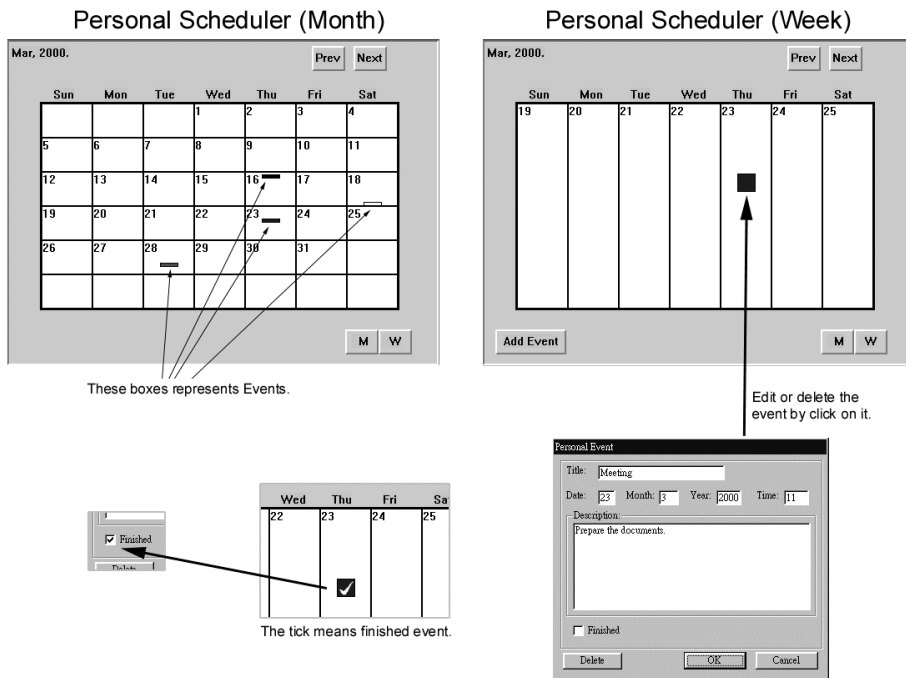
Finally, we have tried to integrate the RPCE into the Menu Driven one in our final product. But it is not so success. The components of RPCE has so few interactions with that of Menu Driven one's and it is like running 2 different applications together. Anyway, we think it is a good trial and worth to be continued explored in future.

4.11 Conclusion on CE

Compared to Menu Driven Collaborative Environment, Role Play Collaborative Environment is far more user friendly. The communication method in RPCE is similar to the actual world. Young students or Old teacher will find it is easy to understand/ use the communication methods.

On the other hand, RPCE is not suitable for the students who require complicated discussion/interaction. It is suitable for the group the students with simple discussion or interactions only. Therefore, RPCE is more likely to be used in kindergarten or primary school while the Menu Driven CE is used in secondary school and university.

5. Personal Application



In our final product, other than the communication tools and CALs, we have also tried to build some applications for teacher/student's personal use.

Personal scheduler (PS)

We have built a personal scheduler that helps the user to schedule their time. The interface of PS is like a calendar. User can add event on the calendar by clicking on a day in the calendar. User can input the event description, the event time and set the color of the event in the calendar (the event will be represented as a color box in the calendar). When the user has finished the event, he can mark a "tick" on the event box. This function can help users to remember their assignments' deadline, meetings, and datings.

The main difference between our PS and that in palm pilots is that the event in our PS can be divided into 2 classes: the personal event and the school event. Similar to palm pilots, the personal events are inputted and accessed by the owners only. On the other hand, the school events are inputted/edited by school teachers or tutors. If the school event is a homework or project. It will be marked finished when it is handed-in. (Unfortunately, we do not have enough time to implemented "school event" in our final products.)

Booking System (BS) (Have not implemented)

Lunch box booking menu

Name: Class:

Std. ID: Pwd:

| Choice: | Quantity: |
|----------------------|----------------------|
| <input type="text"/> | <input type="text"/> |
| <input type="text"/> | <input type="text"/> |
| <input type="text"/> | <input type="text"/> |

The student can book the textbook, uniform or lunch box using this system.

Library System (LS) (Have not implemented)

```
The Chinese University of Hong Kong
University Library System
Public Access Catalogue

L > LIBRARY Catalogue
R > RESERVE Lists for Courses
I > Library INFORMATION
O > OTHER Library Catalogues
A > Other Information Resources
X > Change Dialogue Language to Chinese
D > Disconnect

Choose (L,R,I,O,A,X,D) █
```

It is same as the Library System in CUHK.

Actually, the personal applications mentioned here are not the new and creative idea. Therefore, we have put most of the effort on Collaborative System and leaving most of the personal applications incomplete. Anyway, we just want to integrate make a complete school application but we do not have enough time.

6.

6. Conclusion

In this project, we hope to demonstrate a kind of new learning environment which is suitable to future world. It shows a way for students to learn, communicate and perform joint discussion at different place all over the world.

We can conclude this year work into the following points.

1. We have developed the libraries for socket programming.
2. We have developed the libraries for handling the multi media stuff.
3. We have developed a server.
4. We have developed a CAL called FWLE for studying English
5. Based on the libraries and tools we built, we have developed a system called Collaborative Environment (CE) which allow students who are physically apart to perform joint work. And FWLE becomes a component of CE.
6. We have further extended CE into called Role-Play Collaborative Environment (RPCE) of which the interface is simple, more easy to understand and more suitable for young students.
7. Finally, we have compared the CE (menu driven) and RPCE and give a conclusion.


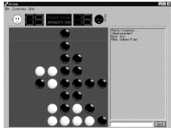


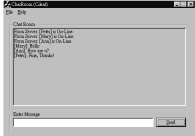

Due to time limitation, some of our idea cannot be implemented. However, we think it is easy to integrate other kind of CAL tools into our system. As our system the whole system is developed by us, it is so flexible to change or plug in new add-ons.

We think that our project is worth to be extended. For example, many CALs can be written for our system. RPCE also have some many possible extensions, such as design more communication methods or build the map in 3D. Finally, we hope that our project can contribute to the education in future.


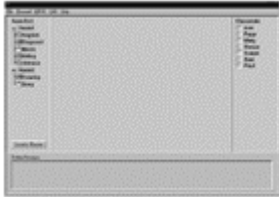


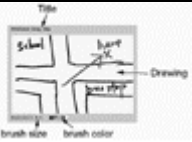


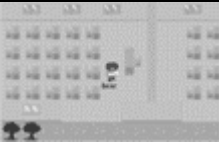
Appendix A References

- [1] Mladen A. Vouk, Donald L. Bitzer and Richard L. Klevans, "**Work flow and End-User Quality of Service Issues in Web-Based Education**", Department of Computer Science, North Carolina State University, Raleigh, NC 27695, USA
- [2] Donald Hearn, M.Pauline Baker "**Computer Graphics 2nd Edition**" Prentice Hall
- [3] Ralph Davis. **Win32 Network programming: Windows 95 and Windows NT network programming using MFC**. Addison-Wesley, c1996.
- [4] Bob Quinn, Dave Shute. **Windows sockets network programming**. Addison Wesley Pub. Co., c1996.
- [5] Stevens. **TCP/IP Illustrated Volume 1: The Protocols**. Addison Wesley. Feb 1998.
- [6] Pat Bonner. **Network programming with Windows Sockets**. Upper Saddle River, NJ : Prentice Hall PTR, c1996.
- [7] Martin Heller. **Advanced Win32 programming**. New York: Wiley, c1993.
- [8] **MSDN Online**: <http://msdn.microsoft.com/default.asp>
- [9] **Microsoft DirectX**: <http://www.microsoft.com/directx/>

Appendix B Progress Report

| Date | Description |
|--|--|
| June, 99. | Evaluating among different OS (WinNT, Win95/98, WinCE and Linux) |
| June, 99. | Evaluating among different programming language (Visual C++, Java) |
| start at June, 99 | Studying Direct X |
| start at June, 99 | Studying Winsock |
| 12 nd -14 th July, 99 | Trying to setup a intranet |
| 15 th July, 99 | Trying the Wireless Devices |
| 14 th July, 99 |  <p>The first testing program - WinTalk, released</p> |
| 25 th July, 99 | The second testing program – “Apple Chess”, released |
| 10 th Aug, 99 |  <p>The third testing program – “Reversi” (Actually it is a newer version of Apples Chess), released</p> |
| 11 th - 30 th Aug, 99 | Build our Direct Draw library (graphical library) |
| 1 st - 14 th Sep, 99 |  <p>Using our Direct Draw library to write the forth testing program – “Plane”</p> |
| 15 th Sep, 99 | Start to build our Direct Sound library (audio library) |
| 22 nd Sep, 99 | Add the audio library to “Plane”. |
| 6 th Oct, 99 |  <p>Write a new game “Ball”, to test combining Winsock and Direct Draw together.</p> |
| 9 th Oct, 99 | Add the Winsock to “Plane” |
| 11 th - 16 th Oct, 99 | Studying Multi Client for WinSock |
| 18 th – 20 th Oct, 99 | Design the structure of Chat Room – WinChat and a generic server for chat room. |
| 21 th – 26 th Oct, 99 |  <p>Write a Chat Room - WinChat using WinSock</p> |
| 24 th Oct –15 th Nov, 99 | Construct the scenario editor and preparing the spt file for FWLE |
| 12 th – 20 th Nov, 99 |  <p>Write the scenario reader for FWLE</p> |

Appendix B Progress Report (cont'd)

| Date | Description |
|---|---|
| 3 rd – 10 th Jan, 00 | Rebuild the Server |
| 9 th -17 th Jan, 00 | Develop the Direct Show Library |
| 19 th – 30 th Jan, 00 |  Integrate chatroom, writeboard and voting into FWLE |
| 4 th Feb, 00 |  Design the Collaborative Environment |
| 6 th – 13 th Feb, 00 | Finish the basic interface of Collaborative Environment |
| 20 th – 24 th Feb, 00 |  Integrate Private Message into CE |
| 5 th – 6 th Mar, 00 |  Integrate Chatroom into CE |
| 12 nd -13 rd Mar, 00 |  Integrate Writeboard into CE |
| 12 nd – 15 th Mar, 00 |  Integrate Mediaroom into CE |
| 17 th – 20 th Mar, 00 |  Integrate Voting into CE |
| 27 th Mar, 00 |  Design the Role Play Collaborative Environment |
| 3 rd Apr, 00 | Draw the map's picture |
| 7 th – 8 th Apr, 00 | Implement the LZW compression algorithm |
| Apr, 00 | Implement the walking |
| Apr, 00 | Integrate Talk, Whisper and Paging |
| Apr, 00 | Integrate FWLE and the games into RPCE and MDCE |

Appendix C Statistics of our program

| Component | Number of lines in the source code (approximate) |
|----------------------|--|
| Libraries | 14,000 |
| FWLE | 8,500 |
| Server | 3,500 |
| CE – Shell | 2,200 |
| CE – Chatroom | 300 |
| CE – Writeboard | 700 |
| CE – Voting | 700 |
| CE – Private Message | 400 |
| CE – Media room | 1,200 |
| RPCE – Shell | 2,500 |
| RPCE – Talk, Whisper | 700 |
| Total | 34,700 |