GUI PACKET

Graphical User Interface On Packet Radio

Keith Sproul, WU2Z
698 Magnolia Road
North Brunswick, NJ 08902
WU2Z @ W2EMU.NJ
AppleLink: Sproul.K

Mark Sproul, KB2ICI
1368 Noah Road
North Brunswick, NJ 08902
KB2ICI @ W2EMU.NJ
AppleLink, Sproul.M

The Radio Amateur Telecommunications Society (RATS)

ABSTRACT

In the 1990's the popularity of Command Line interfaces on personal computers is rapidly fading in favor of Graphical User Interfaces (GUI) on most platforms. A graphical environment using windows, icons and mice has been available on the Macintosh@ since '1984. It has also been available on several other computers for many years. Even computer industry insiders were surprised by the headlong rush to Microsoft@ Windows by owners of IBM-PCs and clones. Surely the time for packet radio to respond to the overwhelming preference of the majority of computer users is long overdue. It's high time we had a Graphical User Interface that runs over the air-waves! The author has had a packet radio Call Book server working on the air for over a year which can be accessed with normal terminals using off-the-shelf packet software. Users can also connect to it with a GRAPHICAL USER INTERFACE and get access to the same Call Book information, but with a slick GUI instead of the old Command Line interface!

INTRODUCTION

When packet radio started back in the early 80's, most of the people using packet were using 'dumb terminals'. Later on they migrated to computers of some kind running programs to make them look like 'dumb terminals'. Then we started seeing programs such as PC-PakRATT™ and DigiCOM that knew specific things about packet radio. This type of program made the day-to-day activities of packet radio easier. Now we have a version of PC-PakRATT that uses a mouse and we have a Macintosh version called MacRATT™ that takes full advantage of the Mac's graphical user interface. There is also a DOS program called Lan-Link that can do many things for you automatically. These are significant improvements over the 'dumb terminal' programs of the past.

Today, most packet radio operators are using a computer of one kind or another. There are lots of Commodore-64's, IBM-PCs / PC clones and a good number of Macintosh computers and Atari's too. Therefore, since most stations use computers, let's take advantage of this *Local* Intelligence. The local computer has to do two things: First and foremost, it must take care of all of the *graphics.,* i.e. the drawing, icons, and mouse handling. Fortunately the

Macintosh system and MS-Windows take care of all of this. Second and much more important to the designers of the protocols, is that the protocol must be extensible. That is, it must be able to define other primitives within itself like a macro language. This is VERY important to reduce the amount of data that is transmitted over the air.

GRAPHICAL USER INTERFACE OVER PACKET RADIO

The most frequent first reaction I get when I talk about doing graphics over packet radio is that 1200 baud is too slow to be useful when used to transmit the greatly increased formatting overhead that GUI's require, and that doing a graphical user interface over packet radio would not be possible.. Although I agree that 1200 baud is slow and that GUI is not desirable at 1200 baud, it can be made **useable.** I have done it!

Another common comment that I get is that if someone puts up a BBS that supports a GUI type of connection, what becomes of the people with dumb terminals, or computers that can't handle the graphics, or people that just don't want to use a graphical interface? The answer to this is simple; the BBS knows your name and other things about you, it can easily remember what type of interface you want to use. Also, why can't the BBS ask "what are **you?",** and if it doesn't get the appropriate answer, assume that you have a normal 'dumb **terminal".** The system that we have running automatically recognizes whether the station is a normal station or a GUI station.

The biggest problem is shaking the packet community out of its complacency. We must demand progress in the area of user interfaces. The second biggest problem is establishing a standard protocol that will work. Once this is done, programs on most computers will become available.

I use Macintosh computers exclusively and have some excellent software for use with packet radio. For connecting to normal PBBSs, I use the program called **MacRATT** from AEA. This program is everything I need for connecting to NORMAL command line PBBSs. It also adds several nice GUI touches that allow me to have an easier interface to PBBS systems that otherwise do not know anything but command line interfaces. This is especially true with the most recent release, **MacRATT** Version 2.1. Although this program is very nice, what I REALLY want is a true Macintosh-like interface, not a command line interface. A true Graphical User Interface will require standardized protocols and local intelligence. This can be accomplished with some existing software such as **MacWorkStation™** or **HyperCard™** on the Macintosh or **Toolbook™** on the PC or it can start from scratch. Due to the variety of computer systems in use, the best way will be to define LOCAL INTELLIGENCE and a DYNAMIC PROTOCOL LANGUAGE.

LOCAL INTELLIGENCE

For an example of local intelligence let's ask, "How many times does a typical user requests on-line help**?"** If instead of just sending the message up to the PBBS, why doesn't the local packet program check to see what version/date of this particular **PBBS's** help file it has saved. Then ask the current PBBS what version/date is the most recent version, At that point it would ONLY download a new file if it is more recent than the one saved within the program. If the one that is saved is current, then just display it, without re-downloading it every time. That way, the help file would come up on the screen immediately unless there was a new **version** on the PBBS, in which case, it would take no longer that it does now.

DYNAMIC PROTOCOL LANGUAGE

A dynamic protocol language is a system that can take a long series of commands and call them with a single command. Macros are a simple form of a dynamic protocol language. The important thing that a DYNAMIC protocol language have is the ability to define and/or redefine the commands on the fly. The reason for this is that each different application will probably want to do the same thing over and over. If it can tell the end user software that a simple command of only a few letters ACTUALLY means to dlo a couple of pages of commands, then you can greatly reduce the amount of data being transferred over the communications link. Another thing that a dynamic protocol language needs is the ability to pass arguments to these 'routines'. A simple example of this would be if you had a system that could draw straight lines given x and y coordinates, you would want to be able to define a routine that could draw a square given only two coordinate pairs. Once you have defined this SQUARE routine, you would not have to send nearly as many pieces of information to get your square, and it could be drawn anyplace on the screen and any size.

For examples of a dynamic protocol language, we can look at several different common systems in use today. The most common system out there is POSTSCRIPT? Although mostly recognized as a page description language, Postscript was originally designed as a dynamic protocol language for use on networks. Another common example of a dynamic protocol language is FORTH. Forth is a fully functional, interpreted computer language that can be changed on the fly. Both of these systems are described in a little more detail below.

CURRENT IMPLEMENTATION OF A GUI CALL BOOK SERVER

Our current call book server is done via Apple MacWorkStation [1] (MWS) software from Apple Computer. There is also a Windows version available called ALAC, which stands for Any Language - Any Computer and is available from United Data Corporation, San Francisco, CA. ALAC is a MacWorkStation compatible program that allows the Windows user complete access to the same interface that MacWorkStation gives you.

MacWorkStation is a dynamically changeable system, as defined above. It is not as flexible as Postscript, but it is already defined, and handles a lot more than just drawing on the screen. It already knows about menus, windows, dialog boxes, user events etc. See the sidebar at the end of this paper for a brief introduction to MacWorkStation.

The MacWorkStation package allows programmers on host computers to communicate with remote terminals in 'normal' text-based ways but have the end-user terminal actually display graphics, windows, icons, menus, etc. The initial design goal of MacWorkStation was to allow the main-frame programmer that knew nothing about how to program graphics on the Macintosh an easy mechanism to implement a GUI environment on a remote work station. MacWorkStation went a lot further than the original design goal and has provided a very rich environment. It has provided a very easy way to implement a GUI environment over packet radio. MacWorkStation is only one way that a remote GUI interface can be implemented. It just happened to be a very easy way to reach my goals without having to write a full-function graphics language.

Mark Sproul, KB2ICI, has written a 'Call Book Server' program which enables you to look up Call Book information on anyone with a US FCC issued license. After getting this system running, we wanted to be able to access it via a Graphical User Interface. In addition to this we did NOT want to exclude anybody that did not have the capability to use a GUI interface.

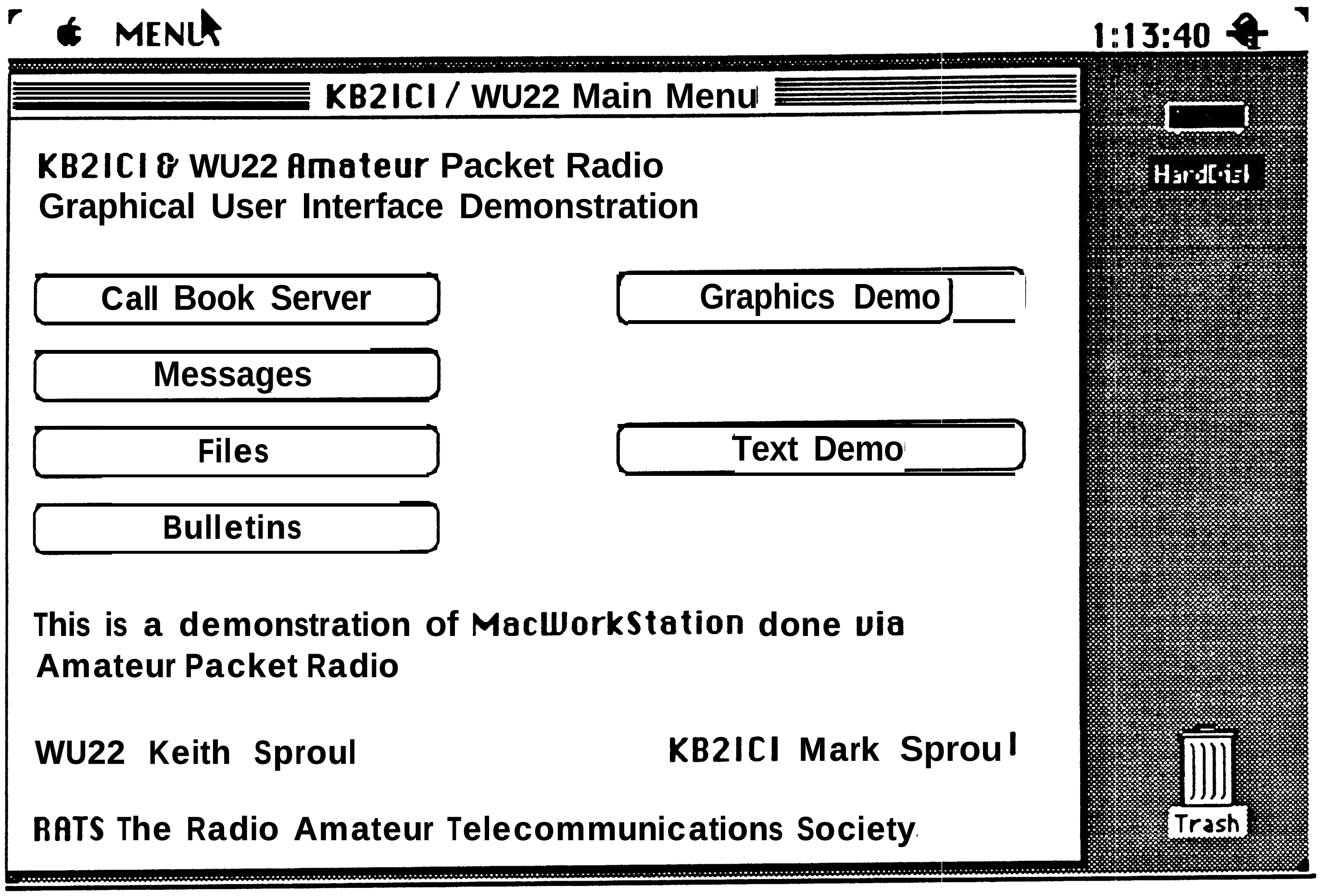
We took the MacWorkStation protocols and added them directly to **KB2ICI's** Call Book **Server** software. This turned out to be easier than first expected due to the fullness and completeness of the MacWorkStation protocols. **A** couple of screens are shown below in Figures 1 & 2. Figure 1 is the ACTUAL screen you get when you sign onto the **KB2ICI** call book server with the GUI software running on the operator's computer. Listing 1 is the ENTIRE listing of MacWorkStation code needed to generate that first screen. Figure 2 is the Call Book Server screen you get when you push the first button on the main screen. This listing puts up the screen, the code that inserts the actual call book information is not listed here.

The MacWorkStation listings are given to show the quantity of the code and the general nature of the types of commands that are sent. They are not meant to teach you how the MacWorkStation commands work. That information is available in the MacWorkStation manuals. **[1]** One of the many things that MacWorkStation allows you to do is once these commands have been sent to the remote work station, they can be saved, and called back up later with a simple one line command. This is done in the current implementation of the Call Book Server software. They can also be saved from session to session so that the next time you log on it only takes one command to pull up the entire screen. In this case it will take LESS data to get logged in that even with the minimum command line user interface!

Another thing that MacWorkStation can do very easily is the ability to check for versions and only download the information if it is newer than what is already saved locally. This is the type of thing that was discussed above about local intelligence.

Listing 3 is the same information obtained when you log into the Call Book Server via a normal 'dumb' terminal. In Listing 3, the underlined commands are what the user typed. In response to the W (or WHO) command, the computer gives a listing of who logged in, when, and if they were logged in normally (indicated by **'-')** or logged in via a Macintosh (indicated by a **'M')** The program can also detect if it was an IBM using the IBM Windows program, **ALAC,** that is compatible with **MacWorkStation.**

FIGURE 1



KB2ICI / WU22 Main Menu

KB2ICI & WU22 Amateur Packet Radio
Graphical User Interface Demonstration

Call Book Server

Messages

Files

Bulletins

Graphics Demo

Text Demo

This is a demonstration of MacWorkStation done via Amateur Packet Radio

WU22 Keith Sproul

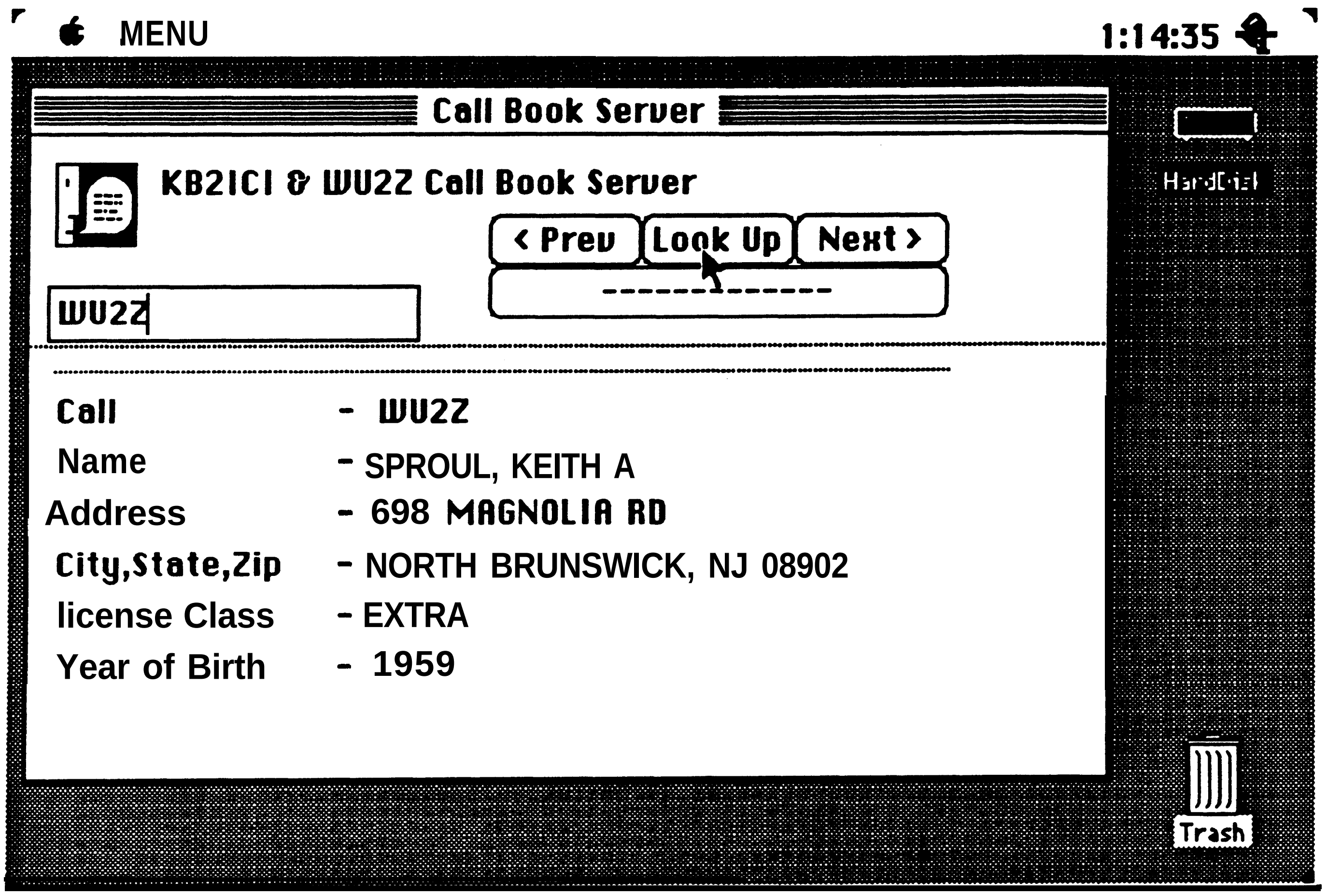
KB2ICI Mark Sproul

RATS The Radio Amateur Telecommunications Society

Listing 1:

```
[M001 1;
[M004 1;MENU,5,RETURN TO MAIN MENU,QUIT;
[M003 1;
[A001;
[D001 1; F; KB2ICI / WU2Z Main Menu; 4; T; 1,43,422,337;
[D009 1; B; 10,60,170,80; Call Book Server; T; F; 0;
[D009 1; B; 10,90,170,110; Messages; T; F; 0;
[D009 1; B; 10,120,170,140; Files; T; F; 0;
[D009 1; B; 10,150,170,170; Bulletins; T; F; 0;
[D009 1; B; 240,60,400,80; Graphics Demo; T; F; 0;
[D009 1; B; 240,120,400,140; Text Demo; T; F; 0;
[D009 1; T; 10,10,280,46; KB2ICI & WU2Z Amateur Packet Radio Graphical User
Interface Demonstration;
[D009 1; T; 10,190,380,226; This is a demonstration of MacWorkStation done via
Amateur Packet Radio;
[D009 1; T; 10,240,150,256; WU2Z    Keith Sproul;
[D009 1; T; 260,240,400,256; KB2ICI    Mark Sproul:
[D009 1; T; 10,270,400,286; RATS The Radio Amateur Telecommunications
Society:
[A001;
```

FIGURE 2



Call - WU2Z
Name - SPROUL, KEITH A
Address - 698 MAGNOLIA RD
City,State,Zip - NORTH BRUNSWICK, NJ 08902
license Class - EXTRA
Year of Birth - 1959

LISTING 2

```
[D008 1;
[D001 2; F; Call Book Server; 4; T; 8,49,430,300;
[D009 2; B; 180,50,360,70; -------------; T; F; 0;
[D009 2; B; 180,30,240,50; < Prev; T; F; 0;
[D009 2: B; 240,30,300,50; Look Up; T; F; 0;
[D009 2; B; 300,30,360,50; Next >; T; F; 0;
[D009 2; E; 10,60,150,76; T; 0; T; ; ; Enter Call Here;
[D009 2; T; 120,100,410,116; -;
[D009 2: T; 120,120,410,136; -;
[D009 2; T; 120,140,410,156; -;
[D0092; T; 120,160,410,176; -;
[D009 2; T; 120,180,410,196; -;
[D009 2; T; 120,200,410,216; -;
[D009 2; T; 10,100,120,116;Call;
[D009 2; T; 10,120,120,136;Name:
[D009 2; T; 10,140,120,156;Address:
[D009 2; T; 10,160,120,176; City,State,Zip;
[D009 2; T; 10,180,120,196; License Class;
[D009 2; T; 10,200,120,216; Year of Birth;
[D009 2: I; 10,10; 1; T; F; 0;
[D009 2; T; 50,10,310,26; KB2ICI & WU2Z Call Book Server;
[D009 2: L; 360,90; 10,90;
[D009 2; L: 0,80; 420,80;
[A001;
```

LISTING 3

```
01:24:08 CONNECTED to KB2ICI-7
Welcome to the KB2ICI/WU2Z Ham server system
Version 1.03 (C) 1990 by Mark Sproul (KB2ICI)
WU2Z DE KB2ICI-7 @ 20:26  - command (?,B,F,G,H,N,P,W)> F WU2Z
Call entered: WU2Z - searching
        Call: WU2Z
        Name: SPROUL, KEITH A
      Street: 698 MAGNOLIA RD
 City,ST,Zip: NORTH BRUNSWICK, NJ 08902
       Class: E issued in 89353, expires in 99353
  Birth year: 59

WU2Z DE KB2ICI-7 @ 20:26  - command (?,B,F,G,H,N,P,W)> W
System running since 8/ 9/1991 20:19
08-09-1991 20:26 -   WU2Z
08-09-1991 29:10 M   WU2Z
...       ...
   Number of records in data file: 515325
  Connections made to this server: 2
Call requests made to this server: 2
WU2Z DE KB2ICI-7 @ 20:26  - command (?,B,F,G,H,N,P,W)> B
73 from the KB2ICI Call Book server
(Note: Underlined characters where typed by the user)
```

GENERAL GUI REQUIREMENTS

To be able to support graphics of any kind over packet radio, we must set some initial guidelines and some minimum requirements:

- Any system developed should have a “dumb terminal” mode so those users that do not have GUI capability can still obtain useful functionality, and so that all users can connect to the system and establish themselves as GUI capable users on that system. The “dumb” terminal mode would at least be able to providle basic information and help about the system protocols and the ability to register.

- Older computers such as Commodore-64's, CP/M computers, Apple IIs, and even some 8088 and 8086 based PCs must be treated as “dumb” terminals since these computers do not have enough processing power to perform adequately in a graphical system.

- We need a system that is complete yet very flexible.

- We need the ability to most of the things described above in the MacWorkStation description, such as windows, menus, graphics, file I/O, text manipulation, etc.

- The communications protocol must be terse.

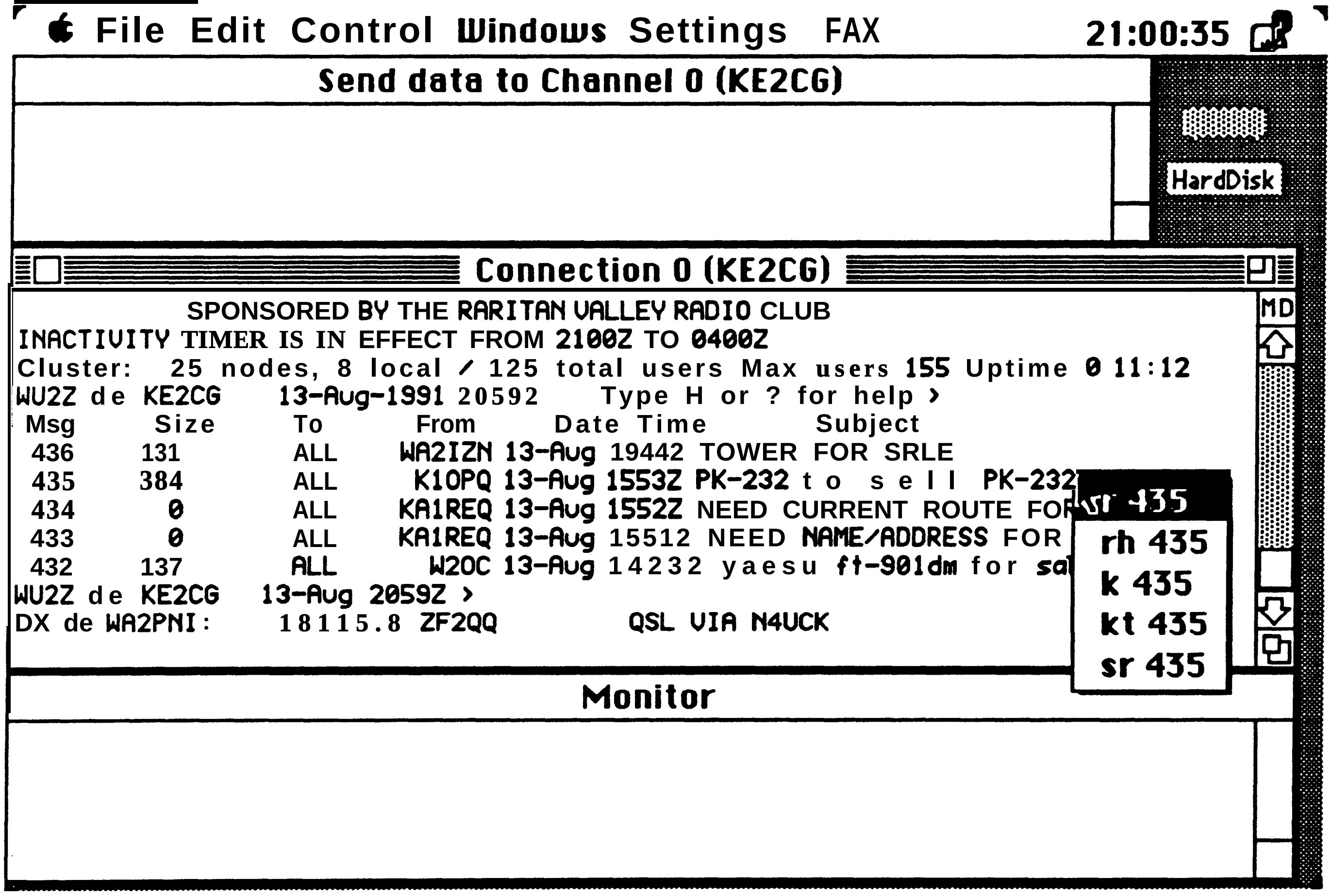
- The protocol must be dynamic, i.e. macro type capability.

- It must have the ability to determine what type of environment the end user is using.

PREVIOUS WORK

This type of system has been discussed **before.[2].** The authors of that paper discussed application specific programs that knew what each other were doing. This type of program is also needed, and fits in to what I call ‘local intelligence’, In this case, the local intelligence means that less data has to be communicated because each program knows exactly what to expect from the other. In the specific case sited above, the authors developed a Chess program in HyperCard on the Macintosh. This program new about chess moves, and only had to communicate the actual chess moves over the air-waves., it did not have to transmit any graphics images of any sorts since both ends were running the same program which already had all of the chess pieces and chess board defined. As written, this program could only do chess, but the concept is very good. Any type of multiple user game, graphical or otherwise could be done very easily in this fashion as long as the game was not a fast action game. The other game that comes to mind that would be excellent for this would be Battleship

FIGURE 3



```
              SPONSORED BY THE RARITAN VALLEY RADIO CLUB
INACTIVITY TIMER IS IN EFFECT FROM 2100Z TO 0400Z
Cluster:  25 nodes, 8 local / 125 total users Max users 155 Uptime 0 11:12
WU2Z de KE2CG    13-Aug-1991 20592    Type H or ? for help >
 Msg      Size      To       From      Date Time       Subject
 436      131       ALL     WA2IZN 13-Aug 19442 TOWER FOR SRLE
 435      384       ALL      K1OPQ 13-Aug 1553Z PK-232 to sell PK-232
 434        0       ALL     KA1REQ 13-Aug 1552Z NEED CURRENT ROUTE FOR
 433        0       ALL     KA1REQ 13-Aug 15512 NEED NAME/ADDRESS FOR
 432      137       ALL       W2OC 13-Aug 14232 yaesu ft-901dm for sa
WU2Z de KE2CG   13-Aug 2059Z >
DX de WA2PNI:    18115.8 ZF2QQ          QSL VIA N4UCK
```

Another example of ‘local intelligence’ that is already **built** into a packet program is **AEA's MacRATT.** This program ‘knows’ about the standard way in which **PBBS's** list messages, i.e., every message in the list starts with a line number. **MacRATT** knows this, and if you enable a QUICKREAD mode, it will let you click with the mouse ANYWHERE on a line, and **MacRATT will** automatically issue the READ command for that message. Figure 3 shows how this works. The mouse button is clicked down at the moment this screen show was taken, showing the pop-up menu with the optional commands. When the mouse button gets released, the selected command gets copied to the send window. The nice touch with this is that the command that gets issued is user selectable from five commands, done via a pop-up menu, and the user can set which five commands to show. This is good because not all PBBS systems use the same commands. This type of 'local intelligence' is only one sided, so it does NOT reduce the amount of data needing to be communicated, but it DOES make it a lot easier for the end user. MacRATT also has excellent macro capabilities, but so do many of the other packet programs

OTHER POSSIBLE SOFTWARE SYSTEMS

The question then becomes: how do we implement a GUI system over packet without having to completely re-write all the software from scratch? We certainly do NOT wish to reinvent the entire wheel, only the rim!

I used the MacWorkStation system from Apple Computer. This allowed me to achieve my goals quickly. MWS has some advantages and disadvantages. Some of the basic ones are listed below:

Advantage	Disadvantage
Already Working	Commercial program
Macintosh & DOS/Windows	No other platforms without major effort
Fully defined protocol	Possibly over-defined
Has already been shown to work	

Other systems could be used. Several of these are discussed briefly below. The problem with any of these others is that a large development effort would be recluired. For the most part, the alternatives listed below have some or all of the graphics primitives already written. Ideally, we would end up with a system that could be as complete as what is available in MacWorkStation, and would have the ability to be on more computers than just the Macintosh and DOS/Windows. The problem is getting people to agree and to sitar-t doing something.

Display Postscript

Though the most widely known use of Postscript is as a "page description" or "printer" language, it was in fact originally designed as a network communications language. Postscript does meet all of the requirements for packet GUI application. Postscript has a few limitations: First, it is slow unless you have a very fast computer. Second, it is NOT easy to program. Third, it would be quite difficult to get new versions of Postscript installed on different types of computers.

Display Postscript is available on Macintosh, DOS/Windows, NeXT, and UNIX, but most other smaller computers are not fast enough to be able to handle Postscript

Postscript was developed from Forth, it has been extended so much that the two only vaguely resemble each other but the basic structure is still the same.

FORTH

Forth is a very extendable language that is available on ALL known desktop computers. Its use would make it relatively easy to transfer a GUI system from one computer platform to another. Forth is available on all desktop computers that have had any popularity at all. It has graphics capabilities on any computer that can do graphics.

Forth would be a good choice if you wanted to develop a system for the most types of computers.

HYPERCARD

Apple's HyperCard program and the equivalent DOS/Windows programs (such as Spinnaker PLUS and Asymetrix Toolbook), would be a relatively easy platform to develop a graphical user interface protocol. HyperCard has already been used to do Packet Chess on the Macintosh. [2] Both the Macintosh and DOS/Windows versions already have the graphics primitives done. In the Macintosh environment, this is a program that ships with every Macintosh, but in the DOS/Windows environment, this program is not very widely used yet.

HyperCard is an 'object oriented' system that would be ideal for developing an entire system for this type of high level communications. It also has the ability to be changed dynamically which leads to even more flexible types of systems. HyperCard is the system I would choose to use if I decide to develop an entire system without relying on already developed software such as MacWorkStation.

HyperCard is only available on the Macintosh and DOS/Windows. It would not be feasible to implement on any other computers.

CONCLUSIONS

I would like to see more done for packet radio in the areas of local intelligence, ease of use, and specifically, Graphical User Interfaces. The method I have used is not the only way to implement a GUI system over packet radio. Considering the amount of software that would have had to be written in any other alternative I could think of, MacWorkStation turned out to be the easiest method by far. I will continue to pursue this method and other methods of achieving a Graphical User Interface over packet radio. I would like to see some of the popular BBS systems start to think about adding this type of capability. The author of one of the popular packet BBS packages is currently looking into this type of capability. Hopefully, other people will get interested in this aspect of packet radio and eventually will have more than just the old command line interface.

REFERENCES

[1] *MacWorkStation Programmer's Guide,* Apple Programmers and Developers Association

[2] Dewayne Hendricks and Robert Taylor, "Application Software for Packet Radio" in *8th Computer Networking Conference,* Newington, CT, **ARRL,** 1989, pp **210-215** (Colorado Springs, Colorado, October *7,* 1989)

SIDEBAR

BRIEF OVERVIEW OF APPLE MACWORKSTATION

MacWorkStation is a complete system for allowing host computers to talk to remote terminals using a full Graphical User Interface without having to write any graphics software on the end user computers. It has a very comprehensive command set that covers almost all bases. Listed bebw are the main categories of commands with a brief overview of each.

- Alert Director The Alert Director allows the client application to report error conditions or otherwise alert the user about something by displaying an alert box. Alerts range from a simple beep to a dialog box that can ask the user for further instructions, such as to cancel the current command or to continue.

- Cursor Director The Cursor Director controls the behavior of the cursor on the screen. The cursor's appearance may be changed or the cursor may be hidden. Changing the cursor's appearance can be an effective means of communication with the user.

- Dialog Director The Dialog Director takes care of displaying and using Dialog boxes on the computer screen.

- File Director The File Director controls the exchange of information between the host computer and the end users computer. It allows the host to create files, open and close files, read and write data to files, and get or set information about files.

- Graphics Director The Graphics Director handles the creation of graphic images. The host application can draw lines, shapes, text, icons, and complicated pictures in a window by using simple Graphics Director commands.

- List Director The List Director allows the host application to display text in a variety of list formats. It also provides limited editing facilities for the user. Examples of things that can be done with lists would include spread sheet type applications, lists of messages on a PBBS, or lists of files that are available for downloading.

- Menu Director The Menu Director allows the host computer to tell the MWS to display any menus that are needed by the current application. It then only reports back to the host computer when a menu item was actually selected.

- Process Director The Process Director handles program control and administration These tasks include protocol handshaking, ascertaining machine and software version numbers, determining what graphics devices are available, setting user wait states, and exiting the MacWorkStation program.

- Text Director The Text Director allows the user to view, scroll, edit, save, and print text in a text windows.

- Window Director The Window Director handles the creation and manipulation of dtfferent types of windows. There can be LIST windows, TEXT windows, and GRAPHICS windows. The window director also takes care of multiple windows and simultaneous displaying of windows of different types.

- Exec Director The Exec Director allows you to extend the MacWorkStation application's functionality by adding custom programs of your own to the MWS environment.

As you can see, this is a quite comprehensive list of command groups. Writing a system to handle a full graphics user interface would be quite an undertaking. This particular system has a bt more in it than would be required for a minimum functional system but even writing a minimal system would be a major undertaking. That is why the authors choose MacWorkStation.