Real-world Programming

Toshiyuki Masui

Sony Computer Science Laboratories, Inc.
3-14-13 Higashi-Gotanda
Shinagawa, Tokyo 141-0022, Japan
+81-3-5448-4380
masui@csl.sony.co.jp

ABSTRACT

Although more and more computing is performed away from desktop computers, most programs used in handheld computers, ubiquitous computers, and augmented-reality systems in the real world are still developed on desktop computers, and users of these systems cannot modify the behavior of the systems or make a new program for the systems without using desktop computers. Programs used in real-world environments should also be programmed in the real world, so we have developed a new programming paradigm, "Real-World Programming (RWP)," which enables users to make programs for handling real-world environments as well as data in computers. By combining simple hardware and software, users can specify actions and conditions and create programs in the real world without using desktop computers. In this paper we describe the features required for RWP, programming techniques for RWP, useful devices for RWP, and examples of RWP.

KEYWORDS: Real-world Programming, Real-world Interface, Augmented Reality, FieldMouse

INTRODUCTION

More and more computing is being performed away from desktop computers and in real-world environments, and various interaction techniques which support off-the-desktop computing have been developed. *Ubiquitous Computing*[10] is an early example of off-the-desktop computing, where users interact with many computers scattered in the real-world environment. *Augmented Reality*[2] or *Enhanced Reality* is an approach to add information in computers to real-world objects in order to enable users handle more information than they can see with their eyes. *Tangible Bits*[5] is a term for making all the information in the computer "tangible" and controlled by users using real-world objects.

The concept common to these techniques is that (1) interacting with real objects in the real-world environment is in many cases much more appropriate than handling information on a computer display using general-purpose input/output devices of desktop computers, and that (2) integration of real-world data and data in the computer is very important. Since there are so many approaches in this direction and it is difficult to classify existing approaches strictly into one of these categories, we call these techniques "*Real-World Interface* (RWIF)" systems as a whole. We expect that RWIF systems are going to become more and more popular, and in the near future, special terms like Ubiquitous Computing and Augmented Reality will not be used, since they will be so common.

Although users can handle real-world objects in current RWIF systems, programming them is usually performed on computers which are separate from the RWIF environment. For example, if a user wants to make a program to ring an alarm bell at 7:00, he has to use symbols both for the alarm and for the time, and issue a command like below:

```
% echo 'play beep.wav' | at 7:00am
```

The command names, file names, and device names are used for the computer's convenience, but not for the user's, even though controlling and programming real-world objects should be as easy as using RWIF systems.

On the other hand, people can program analog alarm clocks easily without using symbols or a computer. When programming an alarm clock, users can easily imagine the movement of the hands in the future and set the alarm hand to the position representing the future time. Users can also check if the program is correct or not just by moving the hands of the clock to their future position. Programming the alarm clock is much more direct and intuitive than programming it indirectly by using texts and symbols. In this way, programming an RWIF system by only using real-world objects is much more intuitive than making programs with texts and symbols which represent real-world objects.

We propose a programming paradigm, "*Real-World Programming* (RWP)," that enable users to create RWIF programs by only using real-world objects and handheld/ubiquitous devices.

REAL-WORLD PROGRAMMING

Listed below are some typical cases where RWP is suitable.

- Ring an alarm at 7:00.

- Turn off the TV when the telephone rings.
- Change the time zone automatically when traveling.
- Display a train timetable when the user approaches a train station at night.
- Open a text file in an editor when a user puts a paper document on the desk.
- If a packet is delivered when nobody is at home, tell the delivery person to take it next door.
- If the captured image of a mountain turns white, download the ski information of the area.
- Remind the user to post a letter when the user gets close to the post office.
- Remind the user to buy a present when the person's spouse's birthday is getting close.
- Print a message when an important e-mail message is received.

Characteristics of Real-world Programming

Matching between programming elements and programmed objects In text-based programming languages, most of the data handled in the program are represented as numbers or strings, and the representation of the program is close to the representation of the data handled in the program. A typical example of this kind of language is the famous C program shown below, where both the program text and the data are represented as a string.

```
printf("Hello, World!\n");
```

There is no problem when the representation of the program is the same as the representation of the data like the case shown above, but if the representation of the program is very different from that of the data, the mismatch between them causes problems.

For example, a program to draw a blue rectangle in a text-based programming language usually looks like this:

```
glColor3f(0.0,0.0,1.0);
glRectf(0,0,100,100);
```

There is a big difference between the representation of the program text shown above and the actual shape of a blue rectangle.

On the other hand, it is much easier to understand the behavior of the program if the blue rectangle is directly specified in the program by using a graphical editor or an "interface builder," because the representation of the program and the representation of the data are almost the same. Visual languages and 2D drawing tools are therefore much more suitable for handling graphical data.

Th ese observations indicate that using real-world objects for making RWIF programs is much better than making programs with text-based languages.

Concreteness Real-world objects can be handled directly in real-world programs, without assigning names or symbols to them. On the other hand, by using a text-based language to make a program for copying data from a VCR to a TV, programmers have to use symbols to represent them like below:

```
% videocopy VCR2 TV1
```

In this case, users have to remember that the name of the VCR is "`VCR2`," the name of the TV is "`TV1`," and the name of the command is "`videocopy`."

Users do not have to remember these symbols if TVs and VCRs are connected by cables and the directions of signals are apparent. RWP should support this kind of concreteness.

Resemblance to PBE Using real-world objects for programming is similar to making programs with "programming by example" (PBE) techniques. PBE is a technique to create programs only by specifying example data or operations, and a variety of researches on PBE have been done so far[4]. Some PBE systems only record conditions and actions from examples and define macros. Others infer a user's intentions from example data given by the user. Some systems even create programs with loops and condition statements only from various examples.

PBE resembles RWP in that only concrete data are used to create programs, instead of using abstract instances like control structures and variables. PBE techniques enable simple abstract program elements like loops, condition statements, and variables to sometimes be safely inferred from examples.

Using real-world objects to specify complex data Using real-world objects and surrogates is sometimes more convenient for specifying complex data and conditions than using text-based programming languages. For example, using a text-based programming language, it would be difficult to make a PDA program for displaying a train timetable when the user gets close to a station. This is because it is not easy to specify the condition "close to a station" by using conventional text-based programming languages. On the other hand, using a map to specify the area close to a station is much easier, because the area can be specified just by drawing a line around the station on the map. In this way, it is sometimes much easier to make programs to specify data and conditions by using real-world data and their surrogates.

REQUIREMENTS FOR REAL-WORLD PROGRAMMING

Various new interaction techniques and software techniques are important for RWP.

Interface Idioms for Real-world Programming

Many user interface "idioms" have been invented since the first computer was operated by a user. A user interface idiom is a set of control operations which may seem strange at a first glance but is easy to remember and hard to forget once users get used to it[3]. In command line languages, typing the return key to confirm the invocation of a command is a well-established idiom. On recent personal computers, many idioms like "clicking a button" and "dragging an icon" are remembered by users and are widely used in various graphical user interface (GUI) systems.

Since RWIFs are relatively newer than command language interfaces and GUIs, almost no idioms for them exist. In many RWIF systems, barcodes are used to represent data or initiate commands. However, unlike icons, a barcode looks only like a set of stripes and does not usually mean anything to the human eye. They should be properly designed so that they can afford users to perform actions which are appropriate to the object with the barcode.

In the IconSticker system[9], a graphical icon is printed on a sticker as well as a barcode so that users can use the barcode just like users can use icons on the computer desktop. This can be regarded as a way to reuse the idioms, but more idioms which fit to RWIF and RWP should also be investigated.

Getting data for RWP

A real-world program should consider the context of the environment (e.g., time and location) and know the status of real-world objects. To identify real-world objects, various sensors and recognition techniques can be used. The easiest way at the moment may be to use printed barcodes on the objects, but putting tags like RFID (radio-frequency identifier) tags and others are also possible. Using existing barcode IDs like ISBN and UPC codes is often very convenient because they are already ubiquitous.

To tell the location of the user, GPS and cellular phone can be used. A telephone number can also be used as a location ID, if a phone directory database is available.

Programming Elements

Programming elements like conditional statements and loops are very important in computer programs, but in most simple real-world programs, the programming elements shown below are also very important.

Using surrogates for real-world objects When a real-world object is difficult to handle in a real-world program, a surrogate can be used for the programming. For example, users can use a map to specify a location instead of actually going there, or use a picture or a namecard to specify a person instead of actually meeting him. Abstract programming elements like a variable or a loop can also be specified by using surrogates. Ishii is proposing the usage of "Phicons" (Physical Icons) as surrogates[5]. Phicons are miniatures of real objects, and they work as controllers for RWIF systems as well as representatives of real-world objects.

Real-world Pattern Matching Text pattern matching is one of the most frequently used functions in a program for handling texts. For example, "`/`*pattern*`/`", a pattern matching operator, is one of the most frequently used operators in Perl programs.

Similarly, real-world pattern matching is an essential part of real-world programs. Operators for matching real-world conditions include matching time, location, weather, etc. Real-world conditions can be specified either by using symbols (e.g., "133.2345E 35.3411N"), using the object itself (e.g., going to a place to specify the location), or using the surrogates (e.g., using a map to specify the location).

NEW DEVICES FOR REAL-WORLD PROGRAMMING

Various low-cost input/output devices can be used for RWIF. Input devices like mouses, video cameras, and barcode readers can be used as sensors to convert real-world data into computer data, and output devices like color printers can be used to convert data in computers into real-world objects. In this section, we introduce two new ways of using these devices for RWIF.

FieldMouse

A FieldMouse is a combination of an ID detection device and a motion detection device. The first device can be a barcode reader, an RFID tag reader, etc., and the second device can be a mouse, a gyroscope, an accelerometer, etc. Using a FieldMouse enables various GUI tools like buttons, menus, and sliders to be used on any surface and objects, just like using a mouse on a desktop computer. Users can control or program various information appliances as easily as using graphical terminals[8].

We have been developing various combinations of these devices. Figure 1 (FieldMouse#1) is a pen-type FieldMouse which consists of a barcode scanner and a pen-mouse. Figure 2 (FieldMouse#2) is a combination of a pen-type barcode scanner and a mouse with a gyroscope.

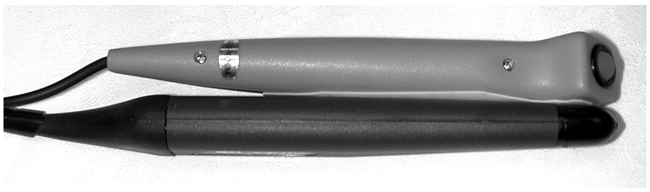

Figure 1: FieldMouse#1: Combination of a barcode reader and a pen-mouse.

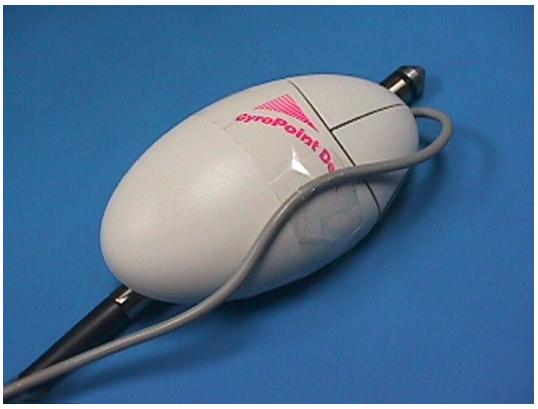

Figure 2: FieldMouse#2: Combination of a barcode reader and a gyro-mouse.

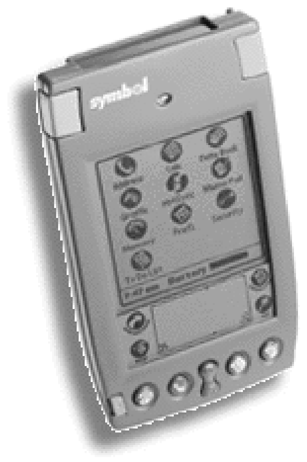

Figure 3: FieldMouse#3: PalmIII PDA including a laser scanner and a tilt sensor.

Figure 3 (FieldMouse#3) is a combination of a PDA, a laser scanner, and a tilt sensor. FieldMouse#3 is based on Symbol Technology's PDA SPT1500[1], which is a combination of a small laser barcode module and 3Com's PalmIII PDA. At the back of the print board of SPT1500, we put a tilt sensor chip ADXL202 by Analog Devices[2] and connected it to one of the unused pins of CPU[3], so that SPT1500 can detect its angle relative to the horizon. ADXL202 can detect two tilt directions, so the CPU on FieldMouse#3 can tell how much it is tilted from its horizontal position.

A FieldMouse can use the barcode reader to tell what it is pointing at and where it is pointing, and it can measure the relative movement of the device after detecting the barcode. Fortunately, many GUI widgets are based on point-and-drag operation and require only these information for interaction, and they can easily be simulated by the FieldMouse. For example, a barcode symbol can be used like a pulldown menu by using the amount of movement for selecting items. If the system interprets the amount of the relative movement as an analog value, it works just like a slider or a scroll bar.

RWGUI operations with a FieldMouse are very similar to GUI operations using a mouse. Table 1 compares using a mouse and using a FieldMouse when manipulating a menu. To use a menu or a slider, a user first moves the FieldMouse

FieldMouse	Mouse
Move the FieldMouse to a barcode	Move the mouse cursor to the menu title
Click a button to initiate recognition	Click the mouse button
Wait until the barcode is recognized	
Move the FieldMouse from the barcode	Drag the mouse cursor
Release the button	Release the mouse button

Table 1: Comparison of using a mouse and using a FieldMouse for selecting a menu item.

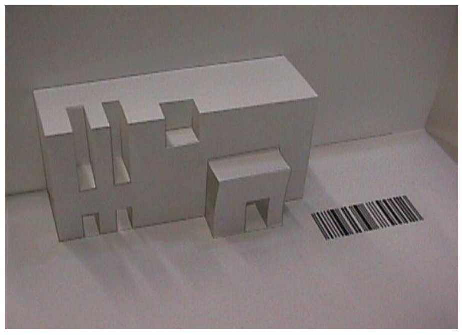

Figure 4: An example origami phicon.

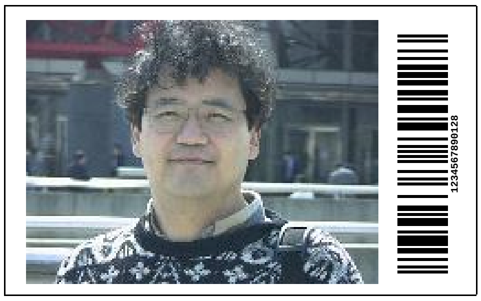

Figure 5: A picture with an ID.

to a barcode, clicks a button to initiate the scanner, waits until the barcode is recognized, moves the FieldMouse and releases the button. Since barcodes are usually recognized instantly, there is almost no time lag in the recognition step, and users feel little differences between using a mouse and using a FieldMouse to operate a menu or a slider.

Barcodes have been used for many years in industries, and small, reliable, and inexpensive barcode readers are widely available. Mouse and motion sensing devices are also widely available, so a FieldMouse can be very easily constructed.

Dynamic Creation of Surrogates

Using phicons is usually not very practical, because creating them usually takes time and there is no good way to save many phicons. However, if folded paper is used instead of plastic or wooden phicons, dynamic creation of phicons becomes possible.

The technique known as "Origamic Architecture" can be used to create papercraft phicons very easily just by cutting part of a printed paper and folding it. These "Origami Phicons" can very easily be unfolded back into a sheet of paper, so many phicons can be preserved just like papers can.

If surrogates do not have to be in a 3D shape, cards and stickers can be used as surrogates by simply printing icons or pictures on them (Figure 5).

[1] http://www.symbol.com/palm/

[2] http://products.analog.com/products/info.asp?product=ADXL202

[3] This technique and schematics are described in: http://www.ibr.cs.tu-bs.de/~ harbaum/pilot/adxl202.html

EXAMPLES OF REAL-WORLD PROGRAMMING

In this section, we show several examples of RWP using the software and hardware techniques described above.

Programming a context-aware PDA

A user can program a PDA to display the train timetable when he gets to the train station at night, by either of the following methods.

Specifying the condition explicitly We assume that the PDA has the macro definition feature and a condition/action pair can be programmed.

- Put the PDA into macro-definition mode.
- Specify the condition "at night" using the face of a clock.
- Specify the condition "close to the station" either by going to the station or using a map around the station.
- Open the timetable on the PDA to specify the action.
- Finish the macro definition mode.

In this way, a real-world macro can be programmed without using text-based programming languages, as easily as defining a keyboard macro in a text editor.

Programming a PDA by examples If a user always opens the timetable when he gets close to the station at night, the system can automatically infer the user's intent and make an appropriate real-world program from the examples. Various techniques for automatic program creation and macro definition have been proposed, and a very simple approach like Dynamic Macro[6] would work for the purpose.

Programming a VCR

Almost everyone can use a VCR to play a videotape fairly easily, but many people have problems programming VCRs. On the other hand, most people can "program" an alarm clock to ring the next morning.

One of the reasons why programming a VCR is difficult and programming an alarm clock is easy is that programming an alarm clock is much more direct than programming a VCR. When programming an alarm clock, users can easily imagine the movement of the hands of the clock in the future, and they can easily set the alarm hand to the future position to set the alarm. On the other hand, when programming a VCR, operations for selecting channels and specifying recording in the programming mode are different from the normal operations for selecting channels, and checking if the the VCR is programmed correctly is not straightforward.

Using the direct programming approach of real-world programming shown below makes programming a VCR as easy as setting an alarm clock.

- Put the clock of the VCR forward to a future time/date, thus automatically setting the VCR to the programming mode.
- Select a channel by normal operations.
- Push the REC button to specify start recording.
- Put the clock further forward to the time/date when the VCR should stop recording.
- Push the STOP button to specify the stop operation.
- Put the clock back to the current time, thus automatically setting the VCR to normal (non-programming) mode.

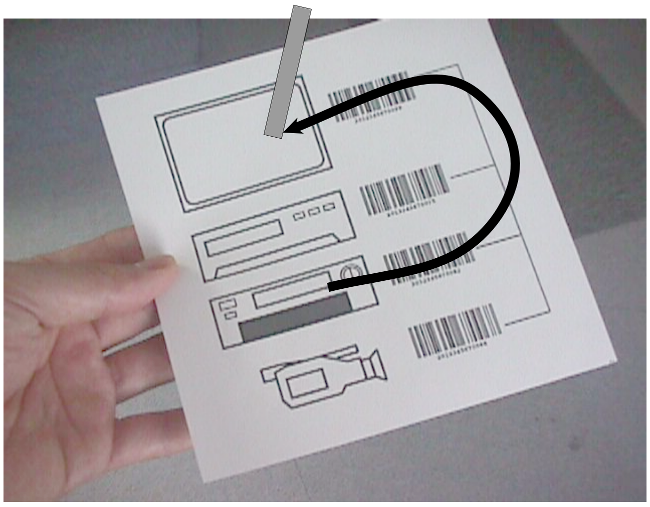
Figure 6: Controlling AV systems by using paper with printed pictures and barcodes.

This method allows users to specify the channel and record/stop operations by using the same buttons used for normal operations; in other words, they do not need to use special buttons for programming. Moreoer, users can check the programming status easily by simply putting the clock forward again.

Programming Data Transfer Between VCRs

In the near future, most audio/visual appliances will be connected by a single network cable instead of using many cables for each audio/video signal. This will reduce the number of cables, but will make setting and understanding the data flow difficult and confusing. For example, if you have two VCRs and want to copy data from one to the other, somehow you have to specify the source and the destination and invoke a command like below:

```
% videocopy VCR1 VCR2
```

This procedure is very cumbersome since you have to (1) use another device to initiate the copy operation, (2) assign names to each VCR, (3) remember the names of each VCR, (4) remember the name of the copy command, and (5) remember whether the first argument of the command is the source or the destination.

When programming a VCR, it is much easier to tell the intention of the user by using the VCRs themselves or their surrogates instead of by using names and symbols. Using the FieldMouse, VCRs can be programmed easily by the following operations.

- Scan the barcode on the source VCR and tilt the FieldMouse upward to specify that this VCR is the source of the data transfer.
- Scan the barcode on the destination VCR and tilt the FieldMouse downward to specify that this VCR is the destination of the data transfer.

Using a panel on which surrogates are printed can specify the same operation even easier than by actually scanning the VCRs (Figure 6).

FUTURE DIRECTIONS

Many problems must be solved before RWP is accepted by people and becomes widely used in RWIF systems.

RWP Idioms

Unlike GUI, there are no interface idioms for RWP at the moment. Many GUI idioms like sliders, pie menus, and drag-and-drop are also applicable to RWGUIs with the FieldMouse, but new varieties of idioms should be invented for RWP. As seen in the evolution of GUIs, once a set of idioms is established, it is difficult to change even if a better set of idioms appears. Good idioms for RWP should therefore be developed from the start.

Simple Authoring Methods

Real-world authoring is one of the most promising applications of RWP. It is possible to automatically create a fancy diary just by logging the time and location of a person's everyday activities, and if the user could add sounds, pictures, movies and URLs to the diary, it would become a more attractive "active diary."

For example, when a person attends a meeting, he can add to his diary various information like another participant's URL, minutes, and memos to his blank diary automatically created from the log of time and location. When a user visits a city and looks for a restaurant, he can add to his diary how he found the restaurant, what it was like, pictures of the food, how he felt about the restaurant, and the link to the information about the restaurant.

These diaries are fun to browse at a later time, but if it takes a great amount of time to author them, very few people would actually make such diaries. If the authoring can be done easily by using RWP techniques, meeting minutes will be ready at the end of the meeting, and fancy photo albums will be ready just when the user gets home. Methods for fast and intuitive real-world authoring should be investigated.

Methods for Browsing RWP Programs

Since real-world programs are not represented as texts, it is difficult to browse them or print them on paper. A simple augmented reality system for browsing and editing RWPs will thus be required.

Specification of Space and Time

Specification of space and time is very important in real-world programming. In the previous examples, maps and clock panels are used, but a more sophisticated way of specifying the combination of space and time will be required.

RELATED WORK

A variety of research is going on in the RWIF field, but most of the developed systems do not support programming or authoring using real-world data.

Arai's PaperLink system[1] is one of the first systems that enables users to define real-world objects like a printed text as a link to other information or a control button for initiating an action. It works very well for simple interaction with papers within limited application areas, but complicated programming is not possible.

Rekimoto's "Augment-able Reality" system[7] is an attempt to perform real-world authoring by using a wearable computer. This system can put various information from real and virtual worlds on a display that can be browsed easily. The system is useful for authoring static RWIF data, but more complex programming is not possible.

CONCLUSIONS

We have introduced a new programming paradigm "Real-World Programming (RWP)" for creating real-world interface (RWIF) programs using real-world objects. With RWP, users can control and program various information appliances and PDAs without using conventional programming languages on desktop computers. Now that many new devices and interaction techniques are widely available, we should develop new interface idioms for RWIF and RWP.

REFERENCES

1. Toshifumi Arai, Dietmar Aust, and Scott E. Huson. Paperlink: A technique for hyperlinking from real paper to electronic content. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'97)*, pages 327–334. Addison-Wesley, April 1997.
2. Ronald T. Azuma. A survey of augmented reality. *Presence: Teleoperators and Virtual Environments*, 6(4):355–385, August 1997.
3. Alan Cooper. *About Face – The Essentials of User Interface Design*. IDG Books, August 1995.
4. Allen Cypher, editor. *Watch What I Do – Programming by Demonstration*. The MIT Press, Cambridge, MA 02142, 1993.
5. Hiroshi Ishii and Brygg Ullmer. Tangible Bits: Towards seamless interfaces between people, bits and atoms. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'97)*, pages 234–241. Addison-Wesley, April 1997.
6. Toshiyuki Masui and Ken Nakayama. Repeat and predict – two keys to efficient text editing. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'94)*, pages 118–123. Addison-Wesley, April 1994.
7. Jun Rekimoto, Yuji Ayatsuka, and Kazuteru Hayashi. Augment-able reality: Situated communication through physical and digital spaces. In *Proceedings of ISWC'98*, 1998.
8. Itiro Siio, Toshiyuki Masui, and Kentaro Fukuchi. Real-world interaction using the FieldMouse. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST'99)*, page to appear. ACM Press, November 1999.
9. Itiro Siio and Yoshiaki Mima. IconStickers: Converting computer icons into real paper icons. In *Proceedings of HCI International'99*, August 1999. to appear.
10. Mark Weiser. Some computer science issues in ubiquitous computing. *Communications of the ACM*, 36(7):75–84, July 1993.