A Modification of Oberon TextFrames for Narrower Displays

Paul Reed, Padded Cell Software Ltd, London
www.paddedcell.com
30th March 2000

Summary: The Oberon System [1] is a graphical operating system and programming environment for a personal workstation computer, ideal for use on machines 'outdated' by the power-hungry operating systems currently popular. However, a high-resolution (1024x768 or larger) display is a necessary hardware requirement for Oberon as originally designed. We propose a crude but simple modification of menu texts to remove repeated module names in commands. This facilitates the use of narrower displays (e.g. 800x600 or even IBM PC VGA resolution, 640x480), and is brought about by making only a few small alterations to the TextFrames module.

1. Introduction

One of the barriers to experimentation with, and adoption of the original Oberon System as a programming and user environment is its hardware requirements. This is unfortunate because Oberon does not require a particularly powerful processor, or huge amounts of memory or disk space (stumbling blocks for most commercially-available systems), but for rewarding use it does require a high-resolution graphical display (1024 pixels across or greater)

All text visible on the screen in Oberon has the same properties - there is no special area where commands are typed (and where output may be subsequently lost). Clicking on any word of the form *Module.Procedure*, such as *System.ShowModules*, will cause the procedure *Procedure* from module *Module* to be called, whereupon typically a new viewer with the desired information will be opened.

An Oberon screen is usually split vertically into a wide User Track on the left, for user activities such as editing documents, and a narrower System Track on the right, intended for the output of system commands like a directory of files, and for the system log.

A consequence of this design is that the menu bar at the top of each viewer (which is normally a single line of text) can contain only a few commands, such as:

```
Demo.Txt  |  System.Close System.Copy System.Grow Edit.Search Edit.Store
```

When the above menu is displayed in the narrower system track on a display which has a width of less than around a thousand pixels, the text (rendered in a reasonably readable font) disappears off the right-hand side of the screen.

Displays of more than a thousand pixels wide are now quite common in the workplace, but are by no means ubiquitous, and many common laptop computers still have 800x600 resolution or less. Besides, the lightweight processor and memory requirements of Oberon represent an excellent opportunity to re-employ older hardware seemingly outdated by bloated and inefficient commercial software. There may be just such a suitable machine holding your office door open at this moment.

Many solutions to this restricted display-width problem have been proposed and used over the years, but most complicate the design of the system considerably. This paper describes a crude but effective technique for shortening the menu text just enough to make the system entirely usable on 600-700 pixel-width displays, which covers virtually every personal computer ever made. Any reader with an original IBM PC with its text-only Monochrome Display Adapter may apply to Padded Cell Software for a FREE Hercules adapter (while stocks last).

For example, under our modifications, the above menu becomes:

Demo.Txt | System.Close .Copy .Grow Edit.Search .Store

2. Shortening Menu Texts

First, menu texts must be shortened when text frames are created. When a text frame is created, the procedure *TextFrames.NewMenu* is called. This in turn calls the procedure *Menu*, internal to TextFrames, which takes the name of the viewer and the list of commands as its two string parameters. *Menu* then returns a Text consisting of these two strings separated by a vertical bar character.

We simply insert some lines at the beginning of this procedure to shorten the local copy of the *commands* parameter if necessary (new code in bold):

```
PROCEDURE Menu(name, commands: ARRAY OF CHAR): Texts.Text;
  VAR T: Texts.Text; mb, me, m, s, d, i: INTEGER; ch: CHAR;
BEGIN
  IF Display.Width < 1024 THEN (*adjust commands*)
    mb := 0; me := 0; s := 0; d := 0;
    LOOP
      (*position to beginning of word*)
      LOOP
        IF s >= LEN(commands) THEN EXIT END;
        ch := commands[s]; INC(s);
        IF ch = 0X THEN EXIT END;
        commands[d] := ch; INC(d);
        IF ("A" <= CAP(ch)) & (CAP(ch) <= "Z") THEN EXIT END
      END;
      IF (s >= LEN(commands)) OR (ch = 0X) THEN EXIT END;
      (*scan first part of word, checking against last module name*)
      m := mb; i := 0;
      LOOP
        INC(i); IF (m < me) & (commands[m] = ch) THEN INC(m) END;
        IF s >= LEN(commands) THEN EXIT END;
        ch := commands[s]; INC(s);
        IF ch = 0X THEN EXIT END;
        commands[d] := ch; INC(d);
        IF (ch < "0") OR ("9" < ch) & (CAP(ch) < "A")
            OR (CAP(ch) > "Z") THEN EXIT
        END
```

```
      END;
      (*check for match*)
      IF (s < LEN(commands)) & (ch = ".") THEN
        IF m = mb + i THEN (*matches previous module name - compress*)
          DEC(d, i); commands[d-1] := "."
        ELSE (*set new module name*)
          mb := d - (i+1); me := d-1
        END;
        (*and do last part of word*)
        LOOP
          IF s >= LEN(commands) THEN EXIT END;
          ch := commands[s]; INC(s);
          IF ch = 0X THEN EXIT END;
          commands[d] := ch; INC(d);
          IF (ch < "0") OR ("9" < ch) & (CAP(ch) < "A")
              OR (CAP(ch) > "Z") THEN EXIT
          END
        END
      END
    END;
    IF d < LEN(commands) THEN commands[d] := 0X END
  END;
  NEW(T); T.notify := NotifyDisplay; Texts.Open(T, "");
  Texts.WriteString(W, name); Texts.WriteString(W, " | ");
  Texts.WriteString(W, commands);
  Texts.Append(T, W.buf);
  RETURN T
END Menu;
```

3. Saving a 'Hint' for Command Execution

Having shortened the menu text, we need to be able to scan the previous commands on a line, in order to infer the module name in the case of a shortened command. We achieve this by saving a 'hint' position when tracking words with the middle (command) mouse button, for use when the button is lifted and the command is executed.

First, a single variable is added at the beginning of the module:

```
VAR
  menuH*, barW*, left*, right*, top*, bot*, lsp*: INTEGER; (*standard sizes*)
  asr, dsr, selH, markW, eolW: INTEGER;
  par: Oberon.ParList; nextCh: CHAR;
  W, KW: Texts.Writer; (*keyboard writer*)
  hintPos: LONGINT;
```

Next, the hint position is saved when tracking the mouse:

```
PROCEDURE TrackWord*(F: Frame; X, Y: INTEGER; VAR pos: LONGINT;
          VAR keysum: SET);
```

```
  VAR old, new: Location; keys: SET;
BEGIN
  IF F.trailer.next # F.trailer THEN
    LocateString(F, X - F.X, Y - F.Y, old);
    ReplConst(Display.white, F, F.X + old.x, F.Y + old.y - dsr, old.dx, 2, 2);
    keysum := {};
    REPEAT
      Input.Mouse(keys, X, Y);
      keysum := keysum + keys;
      Oberon.DrawCursor(Oberon.Mouse, Oberon.Mouse.marker, X, Y);
      LocateString(F, X - F.X, Y - F.Y, new);
      IF new.pos # old.pos THEN
        ReplConst(Display.white,
          F, F.X + old.x, F.Y + old.y - dsr, old.dx, 2, 2);
        ReplConst(Display.white,
          F, F.X + new.x, F.Y + new.y - dsr, new.dx, 2, 2);
        old := new
      END
    UNTIL keys = {};
    ReplConst(Display.white, F, F.X + new.x, F.Y + new.y - dsr, new.dx, 2, 2);
    pos := new.pos; hintPos := new.org
  ELSE pos := -1
  END
END TrackWord;
```

And finally, the hint is used when a command is selected by the user, by detecting whether the command begins with a dot:

```
PROCEDURE Call*(F: Frame; pos: LONGINT; new: BOOLEAN);
  VAR S: Texts.Scanner; res, i, h: INTEGER;
    hint: ARRAY 32 OF CHAR; ch: CHAR;
BEGIN
  Texts.OpenScanner(S, F.text, pos); Texts.Scan(S); h := 0;
  IF ((S.class = Texts.Char) & (S.c = "."))
        OR ((S.class = Texts.Name) & (S.s[0] = ".")) THEN
    (*find hint*)
    Texts.OpenScanner(S, F.text, hintPos);
    REPEAT
      Texts.Scan(S);
      IF S.class = Texts.Name THEN
        i := 0; WHILE (i < S.len) & (S.s[i] # ".") DO INC(i) END;
        IF (0 < i) & (i < S.len) THEN
          h := 0; REPEAT hint[h] := S.s[h]; INC(h) UNTIL h > i
        END
      END
    UNTIL Texts.Pos(S) > pos;
    IF (S.class = Texts.Char) & (S.c = ".") THEN Texts.Scan(S) END
  END;
  IF S.class = Texts.Name THEN
```

```
    IF h > 1 THEN (*use hint*)
      i := S.len; IF S.s[0] = "." THEN DEC(h) END;
      REPEAT S.s[i+h] := S.s[i]; DEC(i) UNTIL (i = -1) OR (S.s[i+1] = ".");
      REPEAT DEC(h); S.s[h] := hint[h] UNTIL h = 0
    END;
    par.vwr := Viewers.This(F.X, F.Y);
    par.frame := F; par.text := F.text; par.pos := pos + S.len;
    Oberon.Call(S.s, par, new, res);
    IF res > 1 THEN
      Texts.WriteString(W, "Call error: ");
      ...
      (error handling)
      ...
      Texts.WriteLn(W); Texts.Append(Oberon.Log, W.buf)
    END
  END
END Call;
```

4. Conclusion

At Padded Cell Software Ltd, we produce custom software using an adaptation of the Oberon system and compiler as our development environment. When working at clients' sites, we have avoided having to install complicated development tools before we can begin work. The entire Oberon system can be used from a single bootable floppy disk or CD-ROM, and with modifications such the one above, can be used on almost any hardware configuration.

The original Oberon system is a shining example of economical software engineering resulting in a highly useful, flexible and reliable software environment. We have shown that, with the help of the lucid explanations of its source code provided in the Project Oberon book, even quite simple modifications can be used to adapt the system to whole new classes of hardware.

References

[1] Wirth, N. and Gutknecht, J. (1992) Project Oberon: The Design of an Operating System and Compiler. Wokingham, England: Addison Wesley