A MOTIF-BASED INSTRUCTOR INTERFACE FOR A MAINTENANCE TRAINER

by
M. Catherine Moan*
Sara Talley
Steve Ford
Frontier Engineering, Inc.
Stillwater, Oklahoma

1. Abstract

Many maintenance trainers of the past have provided a crude and tedious way of entering lesson data, inserting malfunctions, and changing parameters.

This paper describes an instructor interface which uses the Motif Graphical User Interface (GUI) to provide a fast, easy means to accomplish lesson planning and training.

2. Introduction

The F-16 Engine Start System Maintenance Trainer (ESSMT) provides maintenance training for the General Electric (GE) F100-GE-100 and -129 engined F-16C Aircraft. The trainer provides students with realistic training for operating and maintaining the Engine Start System and accomplishes the training objectives identified below:

- Major component identification
- Major component location
- Operational checkout
- Fault isolation/correction

The ESSMT is composed of three major components: instructor station, student station, and computational system as shown in Figure 1. The instructor station consists of a 17" color X-terminal with a Surface Acoustical Wave (SAW) touchscreen and keyboard interfaced with the computational system.

The instructor station provides the following capabilities:

- Instructional feature controls (including log-in, lesson selection, system self-test, etc.)
- Parameter selection
- Malfunction insertion and sequencing
- Freeze control
- Rate control

©1994 ACM 0-89791-684-0/94/0006 3.50

The student station consist of an interactive video display system, simulated sound system, and a simulation panel representing the F-16C Engine Start System with associated components and support equipment. This station provides support for all student actions throughout the exercise interactions.

The computational system is housed in a VME chassis containing a 25 MHz 68030 computational processor unit (CPU), a analog-to-digital/digital-to-analog board, and several digital I/O boards.

The ESSMT software is executed under the Wind River Systems VxWorks real-time operating system. VX-Windows from VisiCom Laboratories, Inc., along with Transportable Applications Environment (TAE) libraries are used to provide an X-window, Motif GUI for the instructor station. The student station is controlled by application-specific software. All newly developed application software is written in the Ada programming language in accordance with the ANSI/MIL-STD-1815A.

2. Rapid Prototyping

The ESSMT development environment consisted of a Sun SPARCstation operating under the Sun OS operating system with X-terminals connected via Ethernet. VADSWorks, a tight integration of the Verdix Ada compiler/linker and Wind River VxWorks real-time executive, along with TAE was used during development of the ESSMT software.

All the instructor menu screens were conceptually drawn up on paper, and were then quickly developed on the development system using TAE. The menus at this stage contained an exact visual replication of the final product, and some limited interaction between menus. An example of one of these menus is shown in Figure 2. This rapid prototyping allowed the instructor to easily identify proficiencies and deficiencies in the design of the menus. When deficiencies were identified, the menu was then quickly corrected. After a few iterations during this prototyping phase, all the menus and the interaction between them were approved before the coding phase began.

* Currently with Hughes Training, Inc. in Arlington, Texas.

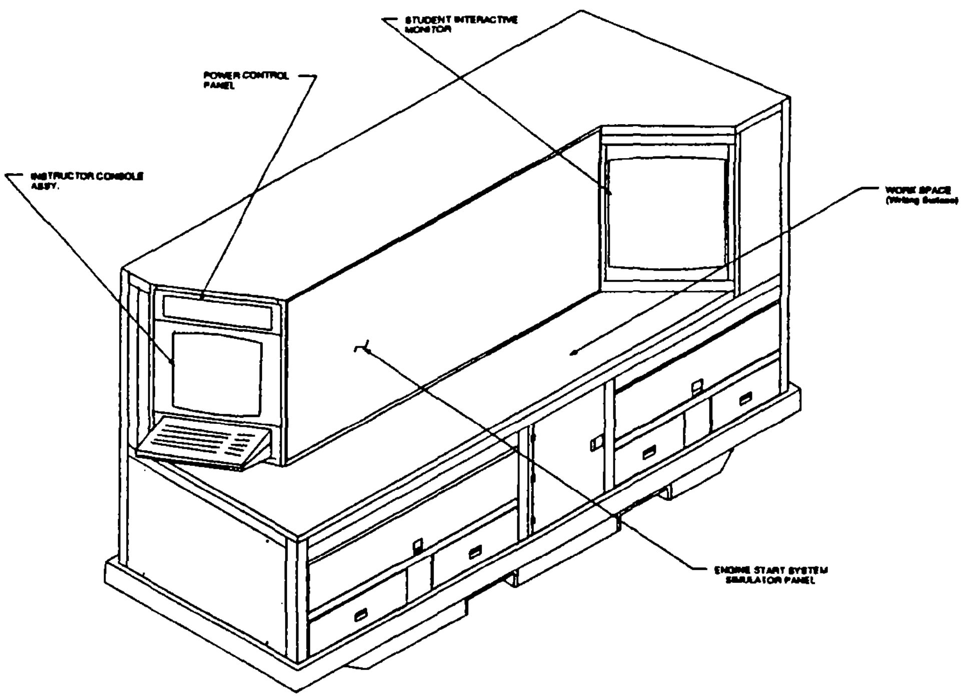
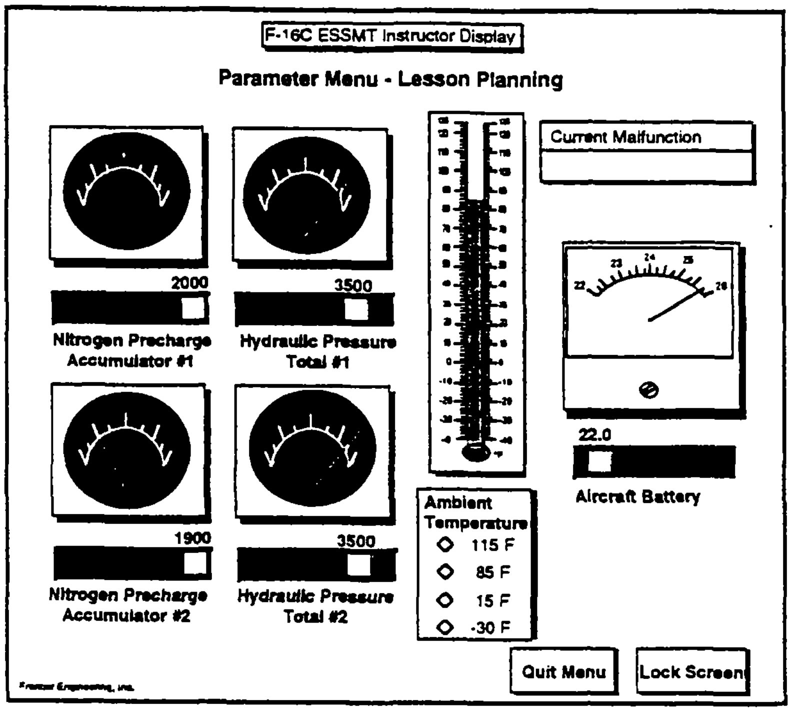

STUDENT INTERACTIVE MONITOR
POWER CONTROL PANEL
INSTRUCTOR CONSOLE ASSY.
WORK SPACE (Writing Surface)
ENGINE START SYSTEM SIMULATOR PANEL

Figure 1. Engine Start System Maintenance Trainer (ESSMT)


F-16C ESSMT Instructor Display

Parameter Menu - Lesson Planning

2000
Nitrogen Precharge Accumulator #1

3500
Hydraulic Pressure Total #1

1900
Nitrogen Precharge Accumulator #2

3500
Hydraulic Pressure Total #2

Current Malfunction

22.0
Aircraft Battery

Ambient Temperature
◇ 115 F
◇ 85 F
◇ 15 F
◇ -30 F

Figure 2. Instructor Parameters Menu

Once the menus were approved, TAE was used to generate Ada "skeleton" code. This code contained all the procedures necessary to display each menu, and the code for limited interaction between the menus.

3. Problems Encountered

The following problems were encountered by using TAE to create the menus and to automatically generate the Ada code:

1. The 68030 processor is no longer supported with TAE.
2. The Motif objects available in TAE did not directly correspond to what was needed, and at times could not be modified with TAE code.
3. Some of the TAE code uses stream I/O for reading files, and this is not available under VxWorks.
4. Certain Unix calls are not supported under VxWorks.

The following paragraphs will describe each of these problems in more detail and the solution that was implemented.

3.1. 68030 TAE Object Code

TAE Version 5.2 was used because of its keyboard traversal capability; however, the Motorola 68030 is not supported with this version. Basically, this means that the 68030 object code was not delivered with version 5.2; however, TAE does provide source code, and makefiles. All of the source code was transferred to a sun3 machine in order to use the default C compiler and all the TAE environment variables were changed to correspond to the new directory structure. The makefiles were then modified for the ESSMT environment. Some excerpts from the makefile Makefile.Xm are listed here with the areas in bold indicating the changed areas.

```
# Makefile.XM

#
# Generic make variables
#

        SHELL = /bin/sh
        AR = ar cq
        BOOTSTRAPCFLAGS =
        ASSEMBLE = false
        HAVETERMLIB = yes
        CC = cc
        CCOPTS =
        C++ =g++
        CPP = /lib/cpp $(STD_CPP_DEFINES)
        LD++ = CC -pipe
        PREPROCESSCMD =
                cc -E $(STD_CPP_DEFINES)
        LD = cc
        LN = ln -s
        RANLIB = ranlib
        RM = rm -f
        STD_CPP_DEFINES =
        STD_C++DEFINES = -DXTFUNCPROTO
        STD_CDEFINES = -D_NO_PROTO
        EXTRA_C++FLAGS =
                -I/home/kathy/tae2/MOTIF
        EXTRA_CFLAGS =
                -I/home/kathy/tae2/MOTIF
        EXTRA_LOAD_FLAGS =
        EXTRALIBS =
        APP_C++FLAGS =
        APP_CFLAGS =
        APP_LOAD_FLAGS =
        APP_LINKLIBS =
        SYSLIBS = $(TERMLIB) $(MATHLIB) $(CLIB)
        C++SYSLIBS = $(TERMLIB) $(MATHLIB)
                $(CLIB) $(C++LIB)
        C++DEBUGFLAGS =
        CDEBUGFLAGS =
        LDDEBUGFLAGS =
        C++FLAGS = $(C++DEBUGFLAGS)
                $(C++INCLUDES)
                $(APP_C++INCLUDES)
                $(EXTRA_C++FLAGS)
                $(APP_C++FLAGS)
                $(STD_C++DEFINES)
        CFLAGS = $(CDEBUGFLAGS) $(CINCLUDES)
                $(APP_CINCLUDES) \
                $(EXTRA_CFLAGS) $(APP_CFLAGS)
                $(STD_CDEFINES) $(CCOPTS)
        LDFLAGS = $(LDDEBUGFLAGS)
                $(EXTRA_LOAD_FLAGS)
                $(APP_LOAD_FLAGS)

.
.
.
        TAECONFIGDIR = $$TAE/config
        WPTDEPLIB = $(TAELIB)/libwpt.a
        XTERMDEPLIB = $(TAELIB)/libxterm.a
        DDODEPLIB = $(TAELIB)/libddo.a
        WMWDEPLIB = $(TAELIB)/libwmw.a
        TAECDEPLIB = $(TAELIB)/libtaec.a
        TAEDEPLIB = $(TAELIB)/libtae.a
        WPTDEPLIBS = $(WPTDEPLIB)
                $(XTERMDEPLIB) $(DDODEPLIB)
                $(WMWDEPLIB)

        TAEDEPLIBS = $(TAECDEPLIB)
                $(TAEDEPLIB)
        DEPLIBS = $(WPTDEPLIBS) $(TAEDEPLIBS)

.SUFFIXES: .cc .C
.C.o:
        $(C++) -c $(C++FLAGS) $<
.cc.o:
        $(C++) -c $(C++FLAGS) $<
```

```
#
#       Make file for the wpt widget library 'libwmw.a'.
#
# 03-jan-92     New -I entries (see PR1257); new depend
target...ljn

LIBRARY = wmw

CINCLUDES = \
        -I/home/kathy/tae2/XMMAK \
        -I$(TAEXM) \
        -I$(TAEINC) \
        -I$(TAEINCXM) \
        -I$(TAETOP)/lib

SRCS = $(TAEXM)/*.c

OBJS = \
        Box.o \
        Icon.o \
        PageEdit.o \
        Pulldown.o \
        Radio.o \
        TextEdit.o \
        TextList.o

all::
        @echo " "
        @echo "   Creating library $(LIBRARY)..."

all::
        @$(MAKE) -e lib$(LIBRARY).a

lib$(LIBRARY).a::
        @echo "   Building library $(LIBRARY)..."

lib$(LIBRARY).a:: $(OBJS)
        $(RM) $@
        $(AR) $@ $(OBJS)
        $(RANLIB) $@
        @echo "   Finished building library
$(LIBRARY)."
```

3.2. TAE Motif Objects

TAE has several pre-defined Motif objects to enable quick development of displays. This was very useful in the rapid prototyping phase; however, during actual implementation, some of the objects needed slight modification. Several of these modifications occurred easily through changing or adding attributes to the objects as shown in the examples listed:

a. **XtSetArg(al[ac], XmNlistSpacing,5); ac++;** was added to TextList.c to adjust the spacing in the textlist object allowing list selection with the touchscreen.

b. **SetTarget (void*) tempstring, 0)** was added to textlistRt.cc to allow deselection of all entries in a textlist.

c. Added check for **keysym == XK_Return** to buttonRt.cc to allow for <CR> to select the widget along with the space bar.

However, the size of the scroll bars on the textlist had to be changed in order to be used with a touchscreen. The following code depicts how the correct attributes were set and the widget was unmapped and mapped to reflect the change.

```
XtSetArg(al[0],XmNverticalScrollBar,&vsb);
XtSetArg(al[1],XmNhorizontalScrollBar, &hsb);
XtGetValues (XtParent(sel->textlist.list),al,2);

if (vsb) {
    XtSetArg (al[0], XmNwidth, 40);
    XtSetValues (vsb, al, 1);
}

if (hsb) {
    XtSetArg (al[0], XmNheight, 40);
    XtSetValues(hsb, al, 1);
}

XtUnMapWidget (sel->textlist.list);
XtMapWidget (sel->textlist.list);
```

3.3. Stream I/O

TAE uses objects called DDOs for implementing picture-type objects. This code was implemented using stream I/O calls in Unix. This presented a problem when porting the code to VxWorks. VxWorks, version 5.0.2 does not support these stream I/O calls. This problem was solved by modifying the TAE source code that contained stream I/O calls and replacing these calls with file I/O calls. A cross-section of the changes made to selection.c, illustrating the types of changes made to all the TAE files containing stream I/O, are shown here.

```
45a46,47
> #include <stdio.h>
> #include <unistd.h>
215,216c217,218
< void Selection::Skip (ivistream& from) {
<   while (from >> buf && strcmp(buf, startdata) !=0) {
-----
> void Selection::Skip (FILE* from) {
>     while (fscanf(from, "%s", buf)
>        && strcmp(buf,startdata) != 0 {
```

```
227c229
< void Selection::ReadVersion (ivistream& from) {
-----
> void Selection::ReadVersion (FILE* from) {
229c231
<   from >> buf;
-----
>    fscanf (from, "%s", buf);
232c233
<         from >> versionnumber;
-----
>         fscan(from, "%d", &versionnumber);
251c252
< void Selection::ReadGridSpacing (ivistream& from,
State* state) {
-----
> void Selection::ReadGridSpacing
        (FILE* from, State* state) {
258c259
<         from >> buf;
-----
>         fscanf(from, "%s", buf);
260c261
<         from >> g;
-----
>         fscanf (from, "%lf", &g);
376,377c376,378
<         from >> lookahead;
<         from.putback(lookahead);
-----
>         fscanf(from, "%c", &lookahead);
>           fscanf(from, "%c", &lookahead);
>            fseek (from, -(sizeof(lookahead)),
                 SEEK_CUR);
386c387,390
<         from>>p>>w>>l>>r;
-----
>         fscanf (from, "%u", &p);
>         fscanf (from, "%u", &w);
>         fscanf (from, "%u", &l);
>         fscanf (from, "%u", &r);
390c394
<         if (undefined || !from.good()) {
-----
>         if (undefined || ferror(from) {
```

3.4. Unsupported Unix Calls

Many of the routines included in the TAE source code were Unix calls that were not supported by VxWorks. Most of these were not used in the ESSMT application; therefore, "dummy" files were created for these procedures. However, there were several procedures that were needed. When this occurred, the procedures was written in Ada and became a part of the ESSMT application. The subroutine memchr() was one of these procedures. The Ada package specification and body are shown here.

```
with system;
with C_strings;

 ---------------------------------------------------------------------
 -
 -- Unit Name:  TAE_VADSWORKS_SUPPORT_PKG
 -- Unit Type:  Package Specification
 -
 -- Description:
 -
 -- The TAE_VADSWORKS_SUPPORT_PKG package
 -- provides support routine neccessary
 -- to support TAE in the VADSWORKS environment.
 -
 -- Unit Function:
 -
 -- function MEMCHR
 -- searches memory for a specified character
 -
 ---------------------------------------------------------------------

 package TAE_VADSWORKS_SUPPORT_PKG is

 ---------------------------------------------------------------------
 -
 -- Unit Name:          MEMCHR
 -- Unit Type:          Package Function
 ---------------------------------------------------------------------

 function MEMCHR(SEARCH_STRING : in
                 C_strings.c_string;
                 SEARCH_CHAR : in  integer;
                 BYTE_COUNT : in  integer) return
                 system.address;

 private

 pragma external(C, MEMCHR);
 pragma external_name (MEMCHR, "_memchr");

 end TAE_VADSWORKS_SUPPORT_PKG;

 ---------------------------------------------------------------------
 -
 -- Unit Name:  TAE_VADSWORKS_SUPPORT_PKG
 -- Unit Type:  Package Specification
 -
 -- Description:
 -
 -- The TAE_VADSWORKS_SUPPORT_PKG package
 -- provides support routine neccessary
 -- to support TAE in the VADSWORKS environment.
 -
 -- Unit Function:
 -
 -- function MEMCHR
-- searches memory for a specified character
-
---------------------------------------------------------------------------


package body TAE_VADSWORKS_SUPPORT_PKG is
---------------------------------------------------------------------------
-
-- Unit Name:          MEMCHR
-- Unit Type:          Package Function
-
---------------------------------------------------------------------------


function MEMCHR(SEARCH_STRING : in
                C_strings.c_string;
                SEARCH_CHAR : in  integer;
                BYTE_COUNT : in  integer) return
                system.address is

---------------------------------------------------------------------------
-- Variable used to index into string being searched
---------------------------------------------------------------------------

CHAR_INDEX : integer;

begin  -- MEMCHR

  begin

  ------------------------------------------------------------------
  -- Set index into string to start at beginning.
  ------------------------------------------------------------------

      CHAR_INDEX := 1;

  ------------------------------------------------------------------
  -- Look through memory BYTE_COUNT number
  -- of bytes to determine SEARCH_CHAR appears.
  -- If it does, return the address of it.  If it doesn't,
  -- return 0.
  ------------------------------------------------------------------

  while (CHAR_INDEX < BYTE_COUNT) loop

   if (SEARCH_STRING(CHAR_INDEX) =
     CHARACTER'VAL(SEARCH_CHAR)) then
       return(system.ADDRESS(SEARCH_STRING
            (CHAR_INDEX)'ADDRESS));
   end if;

   CHAR_INDEX := CHAR_INDEX + 1;
  end loop;

  return (system.ADDRESS'REF(0));
 end;

end MEMCHR;

begin  -- TAE_VADSWORKS_SUPPORT_PKG
  null;
end TAE_VADSWORKS_SUPPORT_PKG;
```

4. Conclusion

With some additional effort, a very impressive and easy-to-use instructor interface for a maintenance trainer was created. With very little improvement in the tools used during development, this method would not require any more effort than the previous instructor interfaces implemented.

REFERENCES:

Heller, Dan, *Motif Programming Manual*, O'Reilly & Associates, Inc., Sebastopol, CA, 1991.

Release Notes for the Transportable Applications Environment (TAE Plus), Version 5.2, (Unix Implementation), Century Computing, Incorporated, Laurel, Maryland, December 1992.

VxWorks 5.0 - Programmer's Guide, Wind River Systems, Inc.