336

N90-20652

TAE PLUS: Transportable Applications Environment Plus

Tools for Building Graphic-oriented Applications

Martha R. Szczur
NASA/Goddard Space Flight Center
Greenbelt, Maryland 20771
[MSZCZUR/GSFCMAIL] TELEMAIL/USA
Marti@DSTL86.span.nasa.gov
(301) 286-8609 FTS 888-8609

INTRODUCTION

The Transportable Applications Environment Plus (TAE Plus™), developed by NASA's Goddard Space Flight Center, is a portable User Interface Management System (UIMS), which provides (1) an intuitive WYSIWYG WorkBench for prototyping and designing an application's user interface, integrated with (2) tools for efficiently implementing the designed user interface and (3) effective management of the user interface during an application's active domain. During the development of TAE Plus, many design and implementation decisions were based on the state-of-the-art within graphics workstations, windowing system and object-oriented programming languages, and this paper shares some of the problems and issues experienced during implementation. The paper concludes with open issues and a description of the next development steps planned for TAE Plus.

TAE PLUS AS A UIMS

Before presenting TAE Plus as a UIMS it is first necessary to define what a UIMS is. The definition by Betts et al [1] which is defined in terms of activities and purposes best describes the objectives of TAE Plus:

"A User Interface Management System (UIMS) is a tool (or tool set) designed to encourage interdisciplinary cooperation in the rapid development, tailoring and management (control) of the interaction in an application domain across varying devices, interaction techniques and user interface styles. A UIMS tailors and manages (controls) user interaction in an application domain to allow for rapid and consistent development. A UIMS can be viewed as a tool for increasing programmer productivity."

TAE Plus is a tool for designing, building and tailoring an application's user interface (UI) and for controlling the designed UI throughout the application's execution. The main component of TAE Plus is a WYSIWYG user interface designers' "WorkBench" that allows an application developer to interactively construct the look and feel of an application screen by arranging and manipulating "interaction objects" (e.g., radio buttons, menus, icons, stretchers, rotators, etc.).

Once the application's screen has been designed, the WorkBench saves the user interface details in a resource file. TAE Plus includes runtime services, Window Programming Tools (WPTs), which are used by application programs to display and control the user interfaces designed with the WorkBench. Since the WPTs access the resource file during execution, the user interface details remain independent from the application code, allowing changes to be easily made to the look and feel of an application without recompiling or relinking the software. To change the user interface, the designer returns to the WorkBench, dynamically makes the modifications, and the resource files are automatically updated. The next time the application is run, the modifications will be in effect. Figure 1 illustrates the TAE Plus structure.

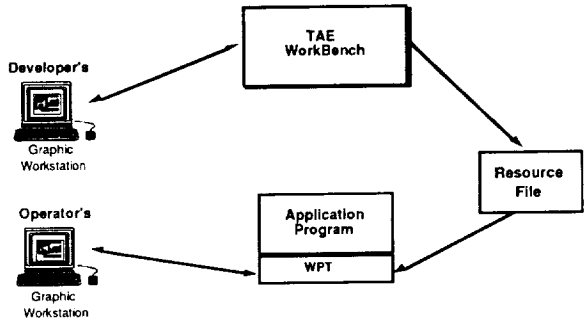


Figure 1. TAE Plus Plus Structure

In addition to providing the WPT runtime subroutines, TAE Plus also offers control of interaction objects from the interpreted TAE Command Language (TCL). This capability provides an extremely powerful means to quickly prototype an application's use of TAE Plus interaction objects and add programming logic without the requirement to compile or link.

INTERACTION OBJECTS AS BUILDING BLOCKS

The basic building blocks for developing an application's user interface are a set of interaction objects. All visually distinct elements of a display that are created and managed using TAE Plus are considered to be interaction objects. Within TAE Plus, interaction objects fall into three categories: user-entry objects, information objects and data-driven objects. User-entry objects are mechanisms by which an application can acquire information and directives from the end use, and include radio buttons, text entry fields, scrolling text lists, pulldown menus, and push buttons. Information objects are used by an application to instruct or notify the user, such as contextual online help information displayed in a scrollable static text object or brief status/error messages displayed in a bother box. Data-driven objects are vector-drawn graphic objects which have been "connected" to an application data variable, and elements of their view change as the data values change. Examples are dials, thermometers, and strip charts, The real-time data-driven objects are the most recent addition to the TAE Plus interaction object collection and currently, the types supported include rotators, stretchers, discretes, text and realtime graphs. Figure 2 illustrates the current set of TAE Plus interaction objects (which are referred to as *items* in the WorkBench). For advanced screen designs, these items can be grouped or composed into larger interaction objects, called *panels* by the WorkBench.

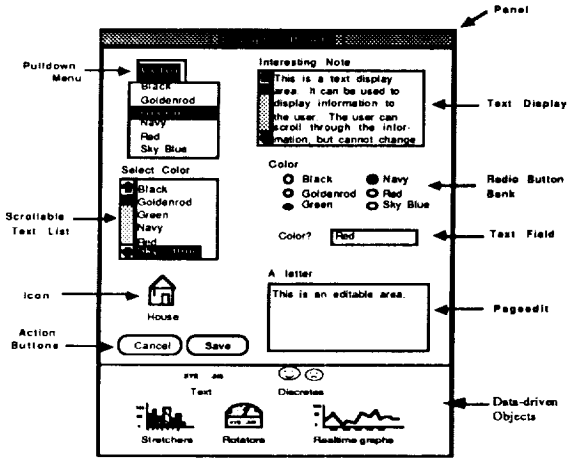


Figure 2. Current set of TAE Plus interaction objects

The use of interaction objects offers the application designer/programmer a number of benefits with the expected payoff of an increase in programmer productivity. [2]

• The interaction objects work together both visually and behaviorally to provide a consistent look and feel for the application's user interface. This consistency translates into reduced end-user training time, more attractive (from a graphic design point of view) screens, and an application which is easier to use.

• Interaction objects provide a common framework for diverse sets of application programs, and serve as a base set of well-documented standards for user interfaces in systems composed of many separate application programs.

• The set of user entry interaction objects covers most common data entry and manipulation needs, allowing the application programmer to spend more time on the content of the application program. The data driven interaction objects provide a standard means of displaying realtime data graphically. The object architecture also enables quick development and addition of new interaction objects into the TAE Plus object library.

• The interaction objects have been thoroughly tested and debugged, allowing the programmer to spend more time testing the application, and less time verifying that the user interface behaves correctly. This is particularly important considering the complexity of some of the objects, and the programming effort it would take to code them scratch.

WORKBENCH SCENARIO

The WorkBench provides an intuitive environment for defining, testing, and communicating the look and feel of an application system. As a designer tool, it provides the following key features:

- Customization and direct manipulation of user interaction objects
- Application code generator
- Capability to dynamically define "connections" between interaction objects
- Rehearsal capability to "try out" sequencing of the user interface design
- Icon editor and support for raster objects
- Undo capability
- Help icon/button on-line support
- Capability to dynamically draw and define data-driven graphic objects

Let's walk through a simple design scenario to get a feel for how the WorkBench operates. The application is a hardware monitoring task for a satellite data handling facility and the designer is going to layout the user interaction in which the

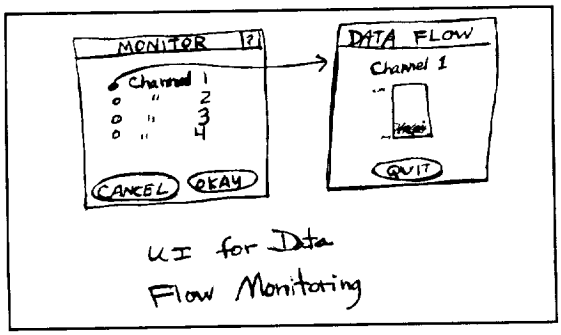


Figure 3. Hand-drawn sketch of application's user interface to be created with WorkBench

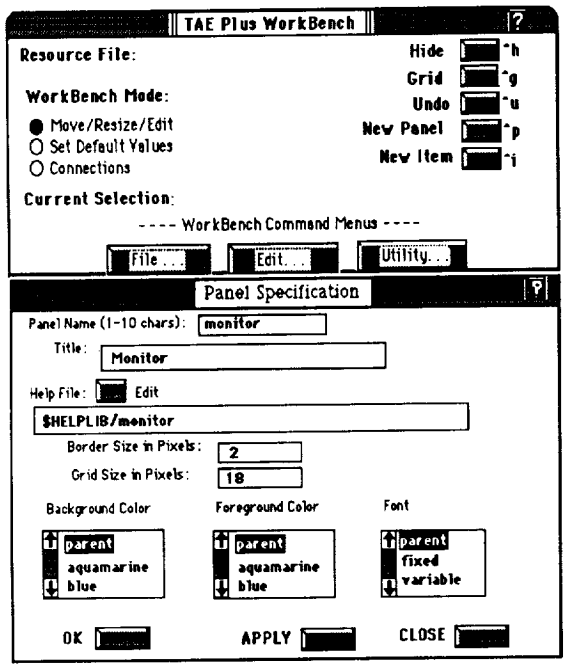


Figure 4. WorkBench's Main Menu and Panel Specification Panel

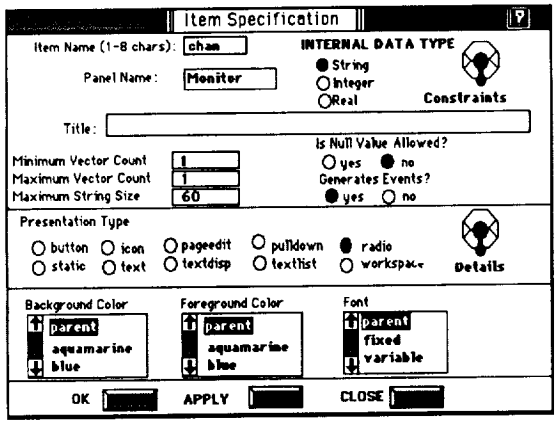


Figure 5. Defining the interaction objects to reside in a panel.

operator is prompted for a hardware channel number. Once the operator selects a channel, a new panel appears with a realtime sliding bar object displaying the amount of data flowing through the channel. Figure 3 shows a rough sketch of the two panels which are to be designed with the WorkBench in this scenario.

Functionally, the WorkBench allows an application designer to dynamically lay out an application screen, defining its static and dynamic areas. The tool provides the designer with a choice of predesigned interaction objects and allows for tailoring, combining and rearranging of the objects. To begin the session, the designer needs to create the base window into which interaction objects will be specified. He/she selects **New Panel** from the main WorkBench menu, which displays the panel specification panel (Refer to Figure 4) where the designer specifies presentation information, such as the title, scroll option, font, color, and optional on-line help for the panel *Monitor*.

The designer is now ready to define the interaction items to reside in the panel. He/she selects **New Item** from the WorkBench main menu and is presented with the item specification window. The designer defines both the presentation information and the context information. The item specification window has an associated *Constraints* (i.e., context) window within which the labels for each entry of a radio button bank object are specified (refer to Figure 5). For the scenario we are following here, the designer has created a radio button bank for the channel numbers, a *cancel* and *okay* button and a panel help icon. For icon support, the WorkBench has an Icon Editor, within which an icon can be drawn, edited and saved.

The designer also has the option of retrieving a "palette" of items (by selecting **File....Include** from the WorkBench menu). From this collection of previously created items, the designer can select and copy appropriate objects. The ability to reuse items saves programming time, facilitates trying out different combinations of items in the prototyping process, and contributes to standardization of the application's "look and feel". If an application system manager wanted to ensure consistency and uniformity across an entire application's UI, all developers could be informed to use only items from the application's palette of common items.

The designer goes through the same process to build the realtime display panel, *DataFlow*. This simple panel is made up of a data-driven stretcher item, selected from a pre-defined pallete of "output objects", and a *quit* button. The WorkBench provides a drawing tool [5] within which the static background and dynamic foreground of a data-driven object can be drawn, edited and saved. Once the object is created, the designer identifies presentation attributes for the object (i.e. the color

thresholds, maximum/minimum, delta).

Most often an application's UI will be made up of a number of related panels, sequenced in a meaningful fashion. Through the WorkBench, the designer defines the interface "connections". These links determine what happens when the user selects a button or a menu entry. The designer attaches "events" to interaction items and thereby designates what panel appears and what program executes when an event is triggered. Events are triggered by user-controlled I/O peripherals (e.g., point and click devices or keyboard input). In Figure 6, the designer has specified links causing the *Dataflow* panel to appear when the end user selects the option marked Channel 1 and the process *Flowcompute* to be executed. In turn, *Flowcompute* is the application process containing the data variable that drives the variations in appearance of the item *BarSlide* .

TAE Plus also offers an optional help feature which provides a consistent mechanism for supplying application specific information about a panel and any interaction items within the panel. In a typical session, the designer elects to edit a help file after all the panel items have been designed. Clicking on the edit help option brings up a text editor window in which the appropriate information can be entered. The designer can then define any button item or icon item to be " the" help item for the panel (in the scenario we are following, it would be the Help icon in the panel *Monitor*). During the application operation, when the end-user clicks on the question mark item, the cursor changes to a "?". The end-user then clicks on the panel itself or any item in the panel to bring up a help panel containing the associated help text.

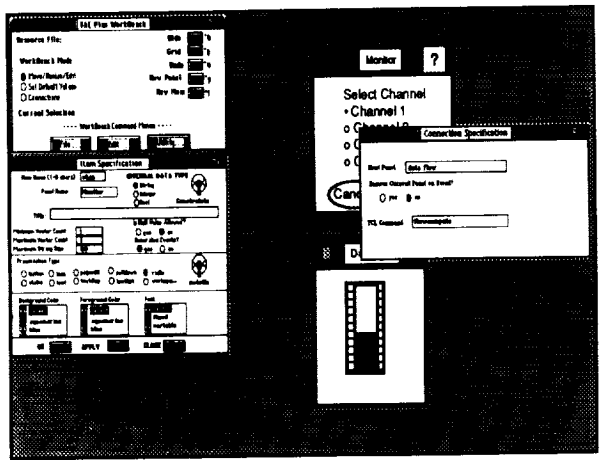


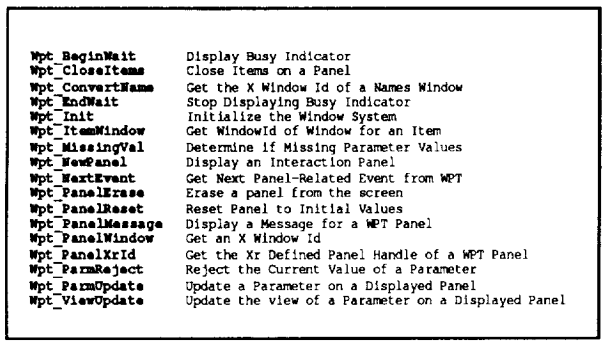
Figure 6.
Using the WorkBench to define "connections".

Having designed the layout of panels and their attendant items and having threaded the panel and items according to their interaction scenario, the designer must then be able to preview (i.e., to rehearse) the interface's operation. With this potential to "test drive" an interface, to make changes, and to dry-run again, iterative design becomes part of the prototyping process. When the designer selects the rehearse option (by selecting **Utility....Rehearse** from the WorkBench Menu), the screen is cleared and the WorkBench goes through the entire sequence as if the application were executing. With the rehearsal feature, the designer can evaluate and refine both the functionality and the aesthetics of a proposed interface. After the rehearsal, control is returned to wherever the designer left off in the WorkBench and he/she can either continue with the design process or save the defined UI in a resource file (by selecting **File....Save** from the WorkBench Menu).

Developing software with sophisticated user interfaces is a complex process, mandating the support of varied talents, including human factors experts and application program specialists. Once the UI designer (who may have limited experience with actual code development) has finished the UI, he/she can turn the saved UI resource file over to an experienced programmer. As a further aid to the application programmer, the WorkBench's "generate" feature (**Utility....Generate**) produces a fully annotated and operational body of code which will display and manage the entire WorkBench designed UI. Currently, source code generation of C, Ada and TCL are supported, with bindings for Fortran and C++ expected in later TAE Plus releases. The programmer can now add additional code to this template and make a fully functional application. Providing these code "stubs" helps in establishing uniform programming method and style across large applications or a family of interrelated software applications.

WINDOW PROGRAMMING TOOLS (WPTs)

The Window Programming Tools (WPTs) are a package of application program callable subroutines used to control an application's user interface. Using these routines, applications can define, display, receive information from, update and/or delete TAE Plus panels and interaction objects (refer to Figure 7 for a current list of WPT). WPTs support a modeless user interface, meaning a user can interact with one of a number of interaction objects within any one of a number of displayed panels. In contrast to sequential mode-oriented programming, modeless programming accepts, at any instance, a number of user inputs, or*events*. Because these multiple events must be handled by the application program, event-driven programming can be more complex than traditional programming. TheWorkBench's auto-generation of the WPT event loop reduces the risk of programmer error within the UI portion of an applications' implementation.



Wpt_BeginWait	Display Busy Indicator
Wpt_CloseItems	Close Items on a Panel
Wpt_ConvertName	Get the X Window Id of a Names Window
Wpt_EndWait	Stop Displaying Busy Indicator
Wpt_Init	Initialize the Window System
Wpt_ItemWindow	Get WindowId of Window for an Item
Wpt_MissingVal	Determine if Missing Parameter Values
Wpt_NewPanel	Display an Interaction Panel
Wpt_NextEvent	Get Next Panel-Related Event from WPT
Wpt_PanelErase	Erase a panel from the screen
Wpt_PanelReset	Reset Panel to Initial Values
Wpt_PanelMessage	Display a Message for a WPT Panel
Wpt_PanelWindow	Get an X Window Id
Wpt_PanelXrId	Get the Xr Defined Panel Handle of a WPT Panel
Wpt_ParmReject	Reject the Current Value of a Parameter
Wpt_ParmUpdate	Update a Parameter on a Displayed Panel
Wpt_ViewUpdate	Update the view of a Parameter on a Displayed Panel

Figure 7. The Window Programming Tools (WPTs)

The WPTs utilize the X Window System™ [10] as its base windowing system. One of the strengths of X is the concept of providing a low-level abstraction of windowing support (Xlib), which becomes the base standard, and a high-level abstraction (X toolkits), which has a set of interaction objects (called "widgets" in the X world) that define elements of a UI's look and feel. The current version of TAE Plus (V3.2) is implemented with the latest release of X (X11.3) using the Xray toolkit, which was distributed with earlier versions of X. We are rewriting our WPTs to utilize the X Toolkit, which is becoming a de facto toolkit standard. The initial approach is to base our default set of interaction objects on the HP widget set delivered with the generic M.I.T. delivery of X (and which is in the public domain) while supporting an open architecture that allows adding to the widget set. A "cookbook" explaining the steps to be taken to replace/add widgets and update the WorkBench is in progress. This will enable TAE Plus to be used for designing and managing the user interface that adheres to whatever UI style is defined by an application group to be their preferred widget set.

The WPTs also provide a buffer between the application program and the X Window System services. For instance, to display a WorkBench-designed panel, an application makes a single call to Wpt_NextPanel. This single call translates into a function that consists of about 2800 lines of C code and makes about 50 calls to X Window System routines. For the majority of applications, the WPT services and objects supported by the WorkBench provide the necessary user interface tools and save the programmer from having to learn the complexities of programming directly with X. This can be a significant advantage, especially when considering that the full set of 17 Wpt routines consist of 5800 lines of C code and make a total of between 300-400 X calls.

PROTOTYPING IN TAE COMMAND LANGUAGE (TCL)

To provide an easy method for displaying and manipulating the newly designed user interface, we created a simple set of commands ("WPT" commands) within the TAE Command Language (TCL).

TCL offers a high-level set of commands used to invoke and manage application functions. Commands can be invoked dynamically during an interactive session or used to build command procedures.An advantage TAE Plus has over some other UIMS is that it does not just support the user interface component of an application, but has a full set of integrated tools to fully support an application, either a prototype or an operational version. These services include parameter manipulation, message logging, logon/logoff procedure, data file I/O, operating system services, scripting capability, session logging, procedure building capability, on-line help, and user-site tailoring of TAE Plus commands. Because user interface tools are integrated with general purpose application management services, the application need not be tightly tied to a particular operating system or computer.

Since TCL is an interpreted language, the commands can be used to prototype an application without having to recompile or relink every time a change is made. Just as with WPT routines used by application programs, the WPT commands can be used to directly define panels and items, or they can be used to access WorkBench-generated resource files that contain pre-defined panels and items. While the intended use of these commands is for prototyping, if the overhead performance of executing TCL commands is acceptable, then command procedures using WPT commands would be appropriate for operational systems.

TAE PLUS ARCHITECTURE

The TAE Plus architecture is based on a total separation of the user interaction management from the application-specific software. The current implementation is a result of having gone through several prototyped versions of a WorkBench and graphic support development during the 1986-87 period, as well as building on an exisiting application management system, the original TAE (affectionately referred to as "TAE Classic"). [9] TAE Classic architecture, which was designed in 1980, was based on a total separation of the user interaction in a much stricter sense than the TAE Plus implementation. All user dialogue was directed through a *terminal monitor*, including dialogues initiated from within an application. This central control of the UI easily facilitated the goal of providing a consistent look and feel across an applica-

tion, but was limited to an ASCII terminal.

The advent of the graphic workstation inspires more elaborate user interfaces and a closer interrelationship between the application program and the UI. The TAE architecture was enhanced to allow for an application to directly control the user interactions, while still maintaining presentation independence (i.e., an application doesn't need to know any of the details as to how a request for data is actually being presented to the user, only what the data is). Figure 8 illustrates how the TAE Plus structure maintains UI/application independence while providing run-time services to control and manipulate the user interactions from within an application.

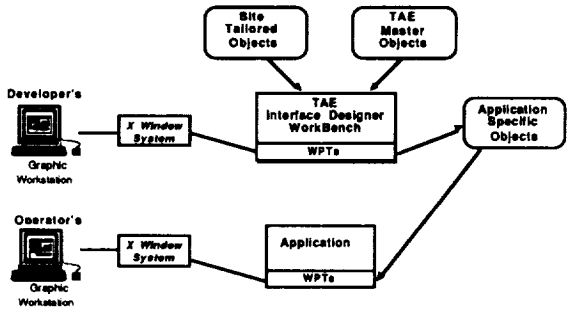


Figure 8.
TAE Plus architecture maintains separation of UI and application elements

SELECTION OF AN IMPLEMENTATION LANGUAGE

TAE Classic is implemented in the C programming language, which has proven to be an efficient and standard language across different hardware platforms, thus allowing for the porting of TAE source code with reasonable ease. However, we felt a "true" object-oriented language would provide us with the optimum environment for implementing the TAE Plus graphical user interface capabilities. (See Chapter 9 of Cox [3] for a discussion on the suitability of object-oriented languages for graphical user interfaces.)

In early 1987, before committing to an object-oriented language and as a means of demonstrating the utility of the X Window System in our UIMS concept, we built a rapid prototype of TAE Plus, using Smalltalk™ to implement the WorkBench. This proved to be a beneficial learning experience. The prototype demonstrated that object-oriented programming is a productive and effective method for building user interfaces. Although Smalltalk enabled us to generate a prototype in a timely manner, several concerns did surface during the implementation. For instance, at the time of the prototyping effort, Smalltalk was not based on the X Window System, which meant the WorkBench and the WPTs had different implementations of the interaction object functions. Another concern with the Smalltalk implementation was that the designer had to have some understanding of Smalltalk's interface conventions -- not a desirable feature since the user interface for applications operating in the TAE Plus environment would have a different set of conventions imposed by an X-based Window Manager. The issue of distribution of TAE Plus with a Smalltalk application was also a problem. With TAE, distribution only involves acquiring a license from COSMIC™ (NASA's distribution center), but for a site to run the WorkBench, they would also need a Smalltalk license. The limited use of Smalltalk in our user community made this undesirable. For these and other reasons [15] we looked at other languages for the operational implementation of the WorkBench .

Though the X Window System is written in C, we did not want to constrain ourselves to a procedure-based language, especially in light of the power of C++ and Objective C, and the fact that interfaces from these object-based languages exist to the X runtime library. For the past several years, we have closely followed the C++ versus Objective C debate. The Objective C argument is strong -- the language is a marriage of two powerful languages (Smalltalk and C), and provides much of the Smalltalk elegance without severe performance penalties. We selected C++, however, for several reasons [15]. For one, C++ seems to be a "cleaner" language (i.e., it is a conceptually strong expansion of C) and is becoming increasing popular within the object -oriented programming community. Another strong argument for using C++ is the growing availability of existing public domain X-based object class libraries. Utilizing an existing object library is not only a cost saver, but also serves as a learning tool, both for object-oriented programming and for C++. Delivered with the X Window System is the *InterViews* C++ class library and a drawing utility, *idraw*, both of which were developed at Stanford University. [4,5] The *InterViews* C++ class library has many attractive features. The class structure has gone through several major iterations and the current design is clean. The *idraw* utility is a sophisticated direct manipulation C++ application, which allows the WorkBench to create, edit and save the graphical data-driven interaction objects.

Many of the current implementations of C++ compilers are pre-processors generating standard C code, thus enabling the operational TAE Plus code to be delivered in C code and allowing for ease in porting. With this option and by utilizing sophisticated public domain software packages (X Window System, InterViews, and idraw) we avoid requiring our user community to purchase any additional software licenses or compilers.

Because of NASA's commitment to use Ada™ for all Space Station software development, the question arises "why not Ada"? We do not consider Ada a purely object-oriented language. [3,11,12,17] As mentioned earlier, we felt that the TAE Plus development would be better served by a "pure" object-oriented language -- one that supports data encapsulation, inheritance and polymorphism. These are the features associated with the type of object-oriented programming supported by Smalltalk and C++. Since TAE Plus software services can be accessed by Ada applications, we feel that implementing the TAE Plus environment in a pure object-oriented language is the most effective approach at this particular time.

PORTABILITY and MAINTAINABILITY

TAE is designed to be portable. At present, TAE Classic is successfully operating on 14 Unix-based computers, VAX/VMS and the IBM/VM environment. TAE Plus base development is being done on a Sun workstation under Unix. As of February 1989, it is also operational under Unix on the Apollo, VaxStation II (Ultrix), HP9000, Masscomp and the Macintosh II (A/UX). Ports are in progress for the IBM RT and IBM PS/2 under AIX and the VAX under VMS. TAE Classic has over 230 installations, of which 64 are NASA. The current beta version of TAE Plus is located at over 100 world-wide beta sites, including at least 30 NASA installations.

Every system is maintainable; *how easy it is to maintain* is the issue. When a UIMS is used as a tool to build and support an application's user interface, there is a legitimate concern about the application's dependency on a "black box". (Since an application program's UI control is isolated in the UIMS, it is frequently perceived by application programmers as a "black box".[6]) The UIMS architecture assure developers that corrections and upgrades to itself will have a minimal impact within the application domain. We knew when we began that TAE Plus was targeted for wide application utilization and for different machines, so ease of maintenance has always been important. By providing the application callable WPTs and WPT function commands, applications are isolated from the windowing system, and thus, if in a few years a newer, faster, fancier windowing standard shows up, only the WPTs require updating or rewriting; the application code is not affected. In effect, this is what we're doing with the rewrite of the WPTs to use the X11 Xtoolkit intrinsics. All applications, as well as the WorkBench, will get enhanced capability and performance without making any changes to themselves.

User support is another facet of maintainability. Since the first release of TAE Classic in 1981, we have provided user support through a fully staffed Support Office. This service has been one of the primary reasons for the success of TAE. Through the Support Office, users receive answers to technical questions, report problems, and make suggestions for improvements. In turn, the Support Office keeps users up-to-date on new releases, provides training sessions, and sponsors user workshops and conferences. This exchange of information enables the Project Office to keep the TAE software and documentation "in working order" and, perhaps most importantly, take advantage of user feedback to help direct our future development.

NEXT STEPS

The current TAE Plus provides a powerful and much needed tool for the continuum of software engineering -- from the initial design phases of a highly interactive prototype to the fully operational application package. However, there is still a long list of enhancements and new capabilities that we will be adding to TAE Plus in future releases. Features included on the "Wanted List" are extensions to the interaction objects, particularly in the data-driven object category; integration with the Open Software Foundation's (OSF) User Environment Component (UEC); direct manipulation support for application programs; ports to new workstation platforms; on-line tutorial and training tools; introduction of hypermedia technology; integration of expert system technology to aid in making user interface design decision; and implementation of additional user interface designer tools, such as a WYSIWYG graph builder.

CONCLUSION

Building large scale interactive systems has been a regular activity at NASA/Goddard Space Flight Center (GSFC) since the transition from card readers to interactive terminals. Although the applications vary from on-board flight instrument command and control to scientific data analysis, they have all required software to support the communication between the human user and the application tasks. In the early 1980's, GSFC sought to capitalize on common requirements in human-computer interaction by building TAE Plus Classic, a powerful tool for quickly and easily building consistent, portable user interfaces in an interactive alphanumeric terminal environment. With the emergence of sophisticated graphic workstations and the subsequent demands for highly interactive systems, the user interface becomes more complex and includes multiple window displays, the use of color, graphical objects and icons, and various selection techniques. Traditional UI paradigms give us only improvished models and

guidelines; they are inadequate for what can be accomplished with the new technology. Prototyping of different user interface designs, thus, becomes an increasingly important method for stabilizing concepts and requirements for an application. At GSFC, we had the requirement to provide a tool for prototyping a visual representation of a user interface, as well as establish an integrated development environment that allows prototyped user interfaces to evolve into operational applications. We feel TAE Plus is fulfilling this role by providing a usable, generalized, portable and maintainable package of development tools. TAE Plus is an evolving system and its development will continue to be guided by user-defined requirements. To date, each phase of TAE Plus's evolution has taken into account advances in virtual operating systems, human factors research, command language design, standardization efforts and software portability. With TAE Plus's flexibility and functionality, we believe it can contribute to both more advances and more standardization in user interface management system technology.

ACKNOWLEDGEMENTS

TAE Plus is a NASA software product being developed by the NASA/Goddard Space Flight Center and by Century Computing, Inc. The work is sponsored by the NASA Office of Space Science and Applications and the Office of Space Operations. Special thanks to Dr. Patricia Carlson for her quality editing and to the TAE Plus Support Office staff for their tireless service to the TAE Project.

TAE is a registered trademark of National Aeronautics and Space Administration (NASA). It is distributed through NASA's distribution center, COSMIC. For further information, contact the TAE Support Office at GSFC, (301) 286-6034.

REFERENCES

1. Betts, B., Burlingame, D., Fischer, G., Foley, J., Green, M., Kasik, D., Kerr, S., Olsen, D., Thomas, J., "Goals and Objectives for User Interface Software", COMPUTER GRAPHICS, 21:2, 1987.

2 .Bleser, Teresa, "TAE Plus Style Guide", NASA Contractor Document, February 1989.

3. Cox, Brad J., OBJECT ORIENTED PROGRAMMING, AN EVOLUTIONARY APPROACH, Addison-Wesley Publishing Company, Reading, Massachusetts, 1986.

4. Linton, Mark, Calder, Paul R., "The Design and Implementation of InterViews", Proceedings of the C++ Workshop, USENIX, November, 1987, pp.256-273.

5. Linton, Mark A., Vlissides, John M., Calder, Paul R., "Composing User Interfaces with Interviews", IEEE COMPUTER, February, 1989

6. Lowgren, Jonas, "History, State and Future of User Interface Management Systems", SIGCHI BULLETIN, 20:2, 1988.

7. Myers, B., "Gaining General Acceptance for UIMS", COMPUTER GRAPHICS 21:2, 1987.

8. Olsen, D., " Larger Issues in User Interface Man agement",COMPUTER GRAPHICS 21:2, 1987.

9. Perkins, D.C., Howell, D.R., Szczur, M.R., "The Transportable Applications Executive -- an interactive design-to-production development system", DIGITAL IMAGE PROCESSING IN REMOTE SENSING, edited by J-P Muller, Taylor & Francis Publishers, London, 1988.

10. Scheifler, Robert W., Gettys, Jim., "The X Window System", MIT Laboratory for Computer Science, Cambridge, MA., October 1986

11. Schmucker, Kurt J., OBJECT-ORIENTED PROGRAMMING FOR THE MACINTOSH, Hayden Book Company, Hasbrouck Heights, New Jersey 1986.

12. Seidewitz, Ed.,"Object-Oriented Programming in Smalltalk and Ada", Proceedings of the Object-Oriented Programming Systems, Languages and Applications (OOPSLA) Conference, October, 1987.

13. Space Station Program Office, "Space Station Information System User Support Environment Functional Requirements", Final Draft, JSC 30497, April, 1987

14. Stroustrup, Bjarne, THE C++ PROGRAMMING LANGUAGE, Addison-Wesley Publishing Company, Reading, Massachusetts, 1987.

15. Szczur, Martha R., Miller, Philip, "Transportable Applications Environment (TAE) Plus: Experiences in "Object"ively Modernizing a User Interface Environment", Proceedings of the Object-Oriented Programming Systems, Languages and Applications (OOPSLA) Conference, September 1988

16. TAE Plus V3.2 Documentation Set, Century Computing, Inc., NASA Contractor Documentation, January 1989

17. Wegner, Peter, "Dimensions of Object-Based Language Design", Proceedings of the Object-Oriented Programming Systems, Languages and Applications (OOPSLA) Conference, October, 1987.