THESEUS on X
A Graphical User Interface System

Matthias Muth
(muth@zgdvda.uucp)

ZGDV
Zentrum für Graphische Datenverarbeitung e.V.
Wilhelminenstr. 7
D-6100 Darmstadt
Federal Republic of Germany

Joaquim A. P. Jorge
(jaj@inesc.uucp)

INESC
Instituto de Engenharia de Sistemas e Computadores
Rua Alves Redol, 9
1000 Lisboa
Portugal

1. Introduction

This paper describes THESEUS, a system to provide graphical user interfaces for software tools, and highlight the implications of porting THESEUS to a multi-programming environment.

It will first detail on the major concepts of THESEUS:

- a windowing concept which hides most window management tasks, such as moving or sizing windows or doing window updates, from the application,
- a graphics output system based on creating and manipulating objects,
- an interactive dialogue system supporting direct manipulation, easily programmable multi-threaded dialogues, incremental dialogue specification in a mixed control architectural model,
- integration of a window bases Graphics Kernel System (GKS).

The portation of the THESEUS system from IBM-PC/ATs to UNIX workstations using Version 11 of the X Window System from MIT is under way as a joint project effort of ZGDV (Darmstadt, West Germany) and INESC (Lisbon, Portugal). This cooperation will result in THESEUS being available as a graphical user interface for tools in many areas, such as software engineering, expert systems, process automation.

A short introduction into the THESEUS concepts will be given. An overview of the structure of the THESEUS implementation on X11 will provide the opportunity to demonstrate the usefulness of combining the X11 Toolkit "widget" abstraction for user interface entities with the high level concepts of THESEUS.

It will be shown how THESEUS in combination with some X11 concepts such as resource management and defaulting mechanisms can be used to easily create graphics user interfaces and adapt them to the user's and the application's preferences.

The important issues raised by porting THESEUS from a single-user/ single-tasking environment to the multi-user/multi-tasking world of UNIX and X will be shown up. The questions involved are how THESEUS, incorporating window manager functionality in itself, and the window managers supplied with X11 will be brought to cooperation. Some ideas about how THESEUS can be extended to make better use of multiprocessing will be presented.

2. Major concepts of THESEUS

THESEUS is a User Interface Management System allowing to design and control graphical user interfaces in a multi window environment [Hübner87A], [Hübner87B]. THESEUS supports the use of graphics in designing man-machine interfaces and simplifies their implementation.

To the user, THESEUS offers the multi windowing concept to allow for working in different contexts at the same time. Within the windows, graphics is used to represent and visualize objects, serving as the communication means between user and machine. The objects can be manipulated interactively in a manner which is obvious to the user, using direct manipulation as a technique.

To the application programmer, THESEUS offers a high level abstraction of all user interface related tasks. Physical abstraction frees the programmer from device dependencies, such as display size and resolution. Logical abstraction is used to allow for creating and manipulating graphics objects and their attributes in an application- oriented manner. On the input side, the application is freed from specifics of the realization of interaction styles and interaction techniques: e.g., menus, known to the application only as a 1-out-of-n selection, can be implemented as pop-up menus triggered by pressing a certain mouse button or as a top-line drop-down menu or even as a command language or function key interface. The syntactical feedback is consistent throughout the system because it in concentrated in one single package.

Control model.

The THESEUS programming interface is based on a modified external control model (Fig. 1). This means that after an initialization phase in which the screen layout, initial graphics objects and the application's reactions to user input related to these objects are set up, the application gives control to THESEUS. From then on, THESEUS will call specific application functions whenever certain user actions occur.

From within these application functions, not only semantic and application-specific processing may be carried out, but THESEUS functions may be called for output and window control and for controlling further dialogue flow. This is done by creating, deleting and modifying input elements and input sets, and by activating and deactivating them to further control the application's reaction to user events. This will happen dynamically in the course of the running application.

As an application function will only be called as a reaction to a specific user input event, it is obvious that it will never have to deal with anything not in the semantic scope of this single dialogue thread. Thus, several working contexts can be programmed independently within the same application, allowing for multi-threaded dialogues by separating the event

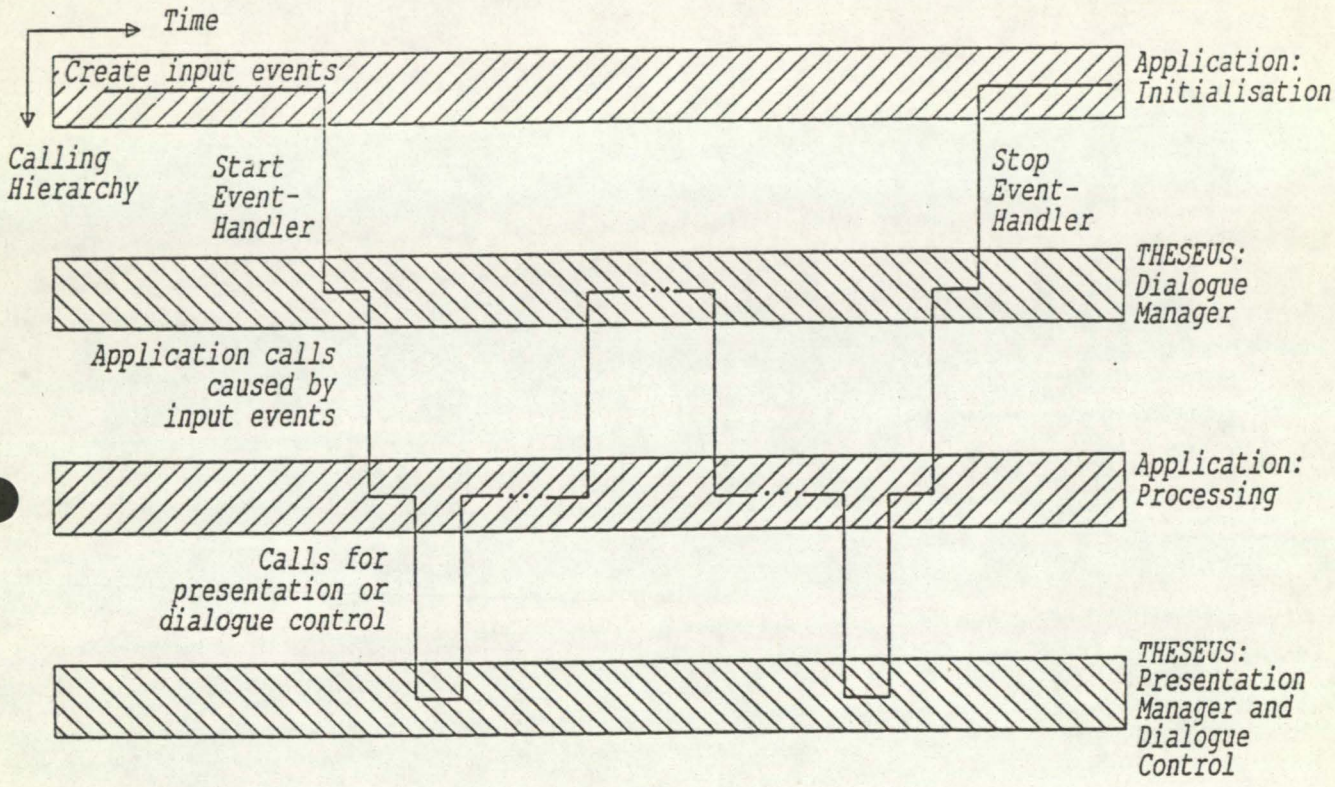

Fig. 1: THESEUS Control Model

dispatching mechanism from the application.

Object oriented graphics output.

THESEUS graphics output is based on the definition of output objects. These objects can be created and exist until they are deleted again, and their attributes may be changed dynamically. This can be contrasted to typical graphics systems, which are usually not object-based but operation-based.

THESEUS basic objects are simple geometric types like polygons, boxes, circles or text or specific raster symbols provided for applications of a special area, such as software engineering. They are defined in a two-dimensional world coordinate system of application dependent size. These basic objects may be structured to form so called complex objects, which can be manipulated exactly like any basic object. This way, applications can build up hierarchical objects and form tree- like structures. Such a structure can be referenced in its whole as a single object or its components can be worked on separately.

An attribute inheritance mechanisms allows do migrate attributes given to a complex object to all of its children objects. For each attribute of each child object it can be specified whether an inherited attribute should be used or if the object overrides the inheritance mechanism, using an attribute of its own.

Alphanumeric output.

THESEUS offers a special window type for alphanumeric output other than graphics text. The level of the programming interface corresponds to alphanumeric interfaces like VT100. Conceptually, all the characters that are made available as output in such a window

are positioned in a world coordinate system whose units of measure are rows and columns. THESEUS maintains a buffer of dynamic size to store the text output in alphanumeric windows in order to be able to do window redraws and to enable the user to scroll through the text. Window Management.

In the philosophy of the "user driven interface" incorporated in THESEUS, it is in the user's choice to rearrange the screen layout at his will.

In most window management systems, the application just gets notified when a typical window operation occurs (e.g. moving or sizing the window) and is fully responsible for redrawing the window contents when the need to do so arises (e.g. when an overlapping window is removed). THESEUS hides these tasks from the application. The object structure is used to redraw the window contents when the window gets unhidden, when it is resized or when the user uses the scrollbars to select another part of the world coordinate system to be displayed within the window.

The application may be totally unaware of these window operations happening. However, it may be useful for the application to take some additional action after certain window operations. The application therefore may be informed about what has happened by supplying THESEUS with a function that is to be called when a specifific window event occurs.

To further support uniformity of the user interface, standardized HELP- and UNDO-mechanisms are supplied using buttons in the border area of a window. Use of these buttons also results in calling an application function whose task it is to supply context related help or to undo operations the user initiated before.

Dialogue control.

User input consists of physical events generated by the user using input devices such as the mouse and the keyboard. It it the main task of THESEUS input and dialogue control to collect these physical events and process them, eventually resulting in an application function that is to be called. This is a multi-step process (see Fig. 2), in which care has to be taken to solve ambiguities caused by several input sets or input elements being active at the same time that are triggered by similar sequences of physical input events.

Physical input events are related to one of the THESEUS input classes:

- Menu selection
- Icon selection
- Object identification (pick)
- Object movement (dragging)
- Position area input
- Keyboard input

Instances of these classes are called input elements. They are grouped together to form input sets. E.g., a menu is represented by an input set whose input elements are the menu items. THESEUS provides functions not only to create and destroy input sets and input elements, but also to dynamically add or remove them from being managed by the THESEUS event handler, to temporarily enable or disable them, and to set or inquire their attributes.

Each input element has the address of an application function associated with it. So if THESEUS finally related a physical event or a sequence of such events to a specific input element, its corresponding application function is called by THESEUS, passing in some details of the recognition process as parameters.

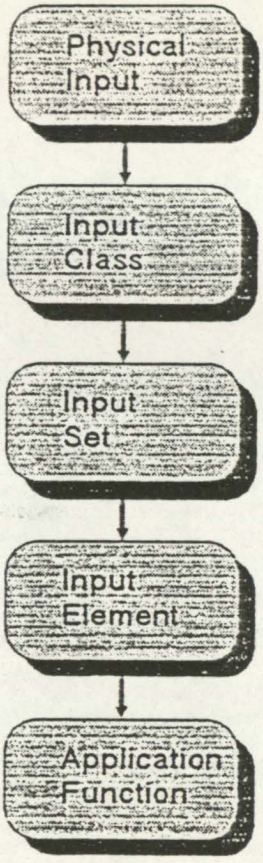

Fig. 2: Processing physical user input events

Several input elements can be connected to the same application function, and the correspondence between input elements and application functions may also be changed dynamically by calling THESEUS functions.

Forms.

A graphic forms system is integrated into THESEUS. It offers editing of multi-line scrollable text fields, buttons to implement 1:n and m:n choices and the ability to use all possible graphics objects within forms.

GKS*.

The functionality of Graphics Kernel System GKS [ISO86] has been implemented to work in THESEUS windows. Some extensions to the GKS standard specification have been developed to integrate the window management capabilities of a system like THESEUS and the standardized general purpose graphics capabilities of GKS [Lux-Mülders88]. The main idea is to use a window as sort of a dynamically changing GKS workstation. The major differences between the GKS standard and GKS* are:

- Some fixed parameters of GKS workstations are made more flexible in GKS* to incorporate dynamic resource distribution (for display surface and colour indices).
- User-invoked window-operations modify GKS* state list parameters directly, without going through the application program. E.g., window sizing will result in a modification of the GKS workstation transformation before redrawing the window.

– The input model of GKS* has been extended to allow context switching, i.e. to suspend a GKS input request as long as the input focus is bound to another window or another non-GKS* interaction.

3. THESEUS architecture

The THESEUS architecture is shown in Fig. 3.

The Basic Input/Output System is an implementation dependent part of the system. It is used to hide device dependencies from the other modules for output and input. In the current implementation, the Digital Research GEM product is used on the IBM-PC/AT to serve as an interface to the various graphics cards.

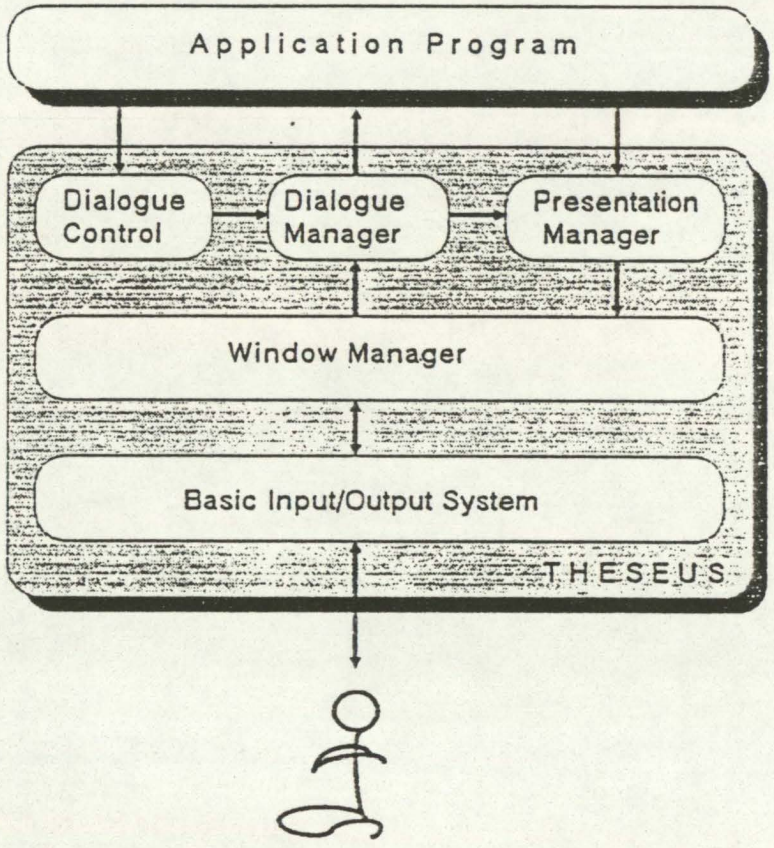

Fig. 3: THESEUS architecture

The Window Manager is responsible for all window operations. It offers an output interface which is capable of drawing output primitives into windows, providing clipping to window borders and hidden areas of windows and the transformation from window to screen coordinates. Parts of this layer can be realized using existing window management systems.

The Presentation Manager executes graphics and alphanumeric output and to provide access to window control functions to the application. Here, the output data structures are maintained.

The Dialogue Manager executes the transformation process from physical to logical input events described above, eventually calling application functions. Its central instance is the input class recognition mechanism, build as a finite state machine which is triggered by

physical input events. Derived from the currently active input classes and elements, it selects the physical events that it is capable of processing. It then will accept physical events to switch through basic interactions such as popping down menus, highlighting items, dragging objects etc. Reaching some of the states, application processing will be triggered, after which the automaton starts all over again, as a new interaction sequence can be started and there possibly are new input elements to be considered after the application function was called.

The Dialogue Control module incorporates the programming interface that allows the application to create, delete and control input sets and input elements.

4. Porting THESEUS to X

THESEUS has been implemented on IBM-PC/ATs using GEM. It has been made available to UNIX applications as a server process running on the PC that can be accessed as a graphics front end for UNIX programs using a remote procedure call mechanism.

Wider spread use of THESEUS can be made as soon as it is available on UNIX workstations in a version based on the X Window System, Version 11 [Gettys88].

X is a network transparent window system developed at MIT which runs under a wide variety of UNIX workstations and is supported by all major workstation vendors. The X Window System supports overlapping hierarchical subwindows and a basic set of graphics and text operations both on monochrome and colour displays. X is designed as a "policy-free" base mechanism that can be used by applications to implement their own style of graphics user interface.

Together with X, the X Toolkit is provided, which gives programmers the next layer of functionality [McCormack88], [Swick88]. The main concept of the X Toolkit is that of a "widget", which is an abstraction for a simple or compound user interface entity. A widget consists of an X window and its own associated semantics. Using an object oriented programming approach, standard mechanisms exist to construct new widget classes, to combine widgets into more complex ones, and to use a powerful defaulting mechanism.

Widgets are completely self contained entities. Each widget implements an event handling routine that will be called by the central X Toolkit event dispatching routine XtDispatchEvent only when an event happened that the widget had been waiting on. This is easily done in X, as the widgets are realized as X windows, and each input event is directly related to an X window (Fig. 4).

The input control model is very similar to that of THESEUS in that application functions are called whenever some semantic action has to take place ("client callbacks").

The X Toolkit supports combining widgets into composite widgets. A composite widget implements a layout routine called geometry handler which is responsible for arranging the positions and sizes of all the subwidgets, taking their respective wishes into respect or just overriding them. The geometry handler does not need to know what kind of widgets it performs its layout policy for.

In porting THESEUS to the X Window System, as much use as possible of the X Toolkit philosophy as well as of the sample widget implementations that come with the system will be made.

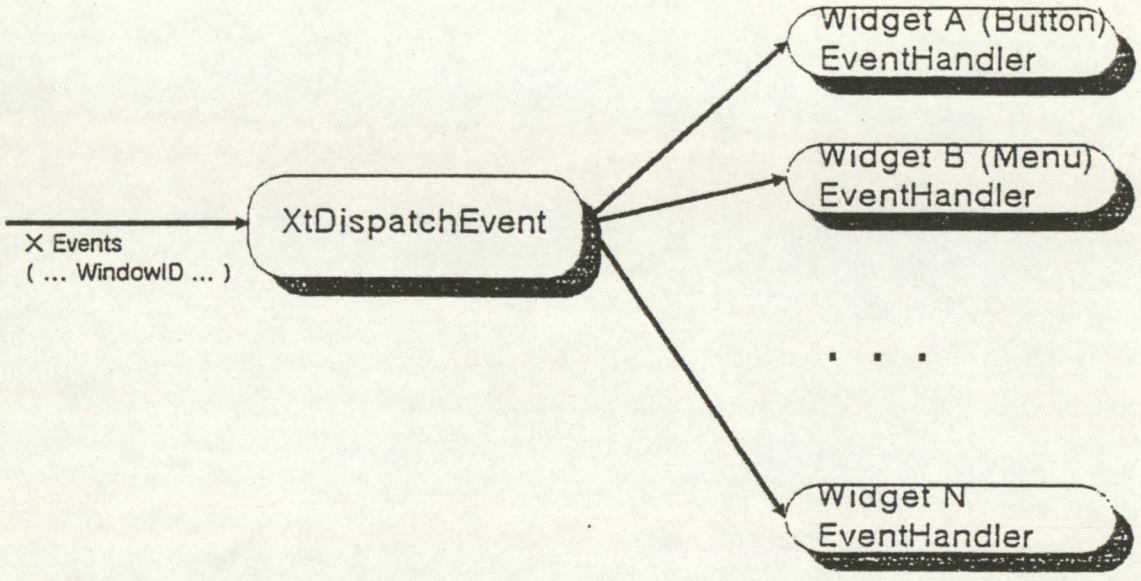

Fig. 4: Dispatching X Events to Widgets

4.1. Mapping THESEUS Concepts to X

In the following, it will be described how the concepts and architectural properties of THESEUS can be transformed into the X world.

Window Management.

The window management area probably is the one that is influenced most by the change of the underlying base system. Some special problems will be discussed in a later section.

In the X implementation of THESEUS, all the border interaction zones and the work area of the window itself will be implemented as widgets. A geometry manager has to be written that provides for the layout of a THESEUS window as seen in the current implementation. Many of the border zones will be instances of a predefined Button widget class.

For the work area, a "THESEUS work area widget" class will have to be created. The window redraw mechanism will be pretty much the same as in the current implementation. It will be possible to gather expose-events for the work area widget and to draw all output objects looping through the object data structure.

Output.

In the output area, an internal interface exists which makes a transformation of window related output requests into calls to the basic input/output system. This part has to be rewritten, which seems rather straightforward, since in X, output is done in window coordinates, so there is no need for a transformation, and X provides window border clipping automatically. Both this had to be done by THESEUS on GEM. Object Management itself as well as the implementation of the GKS* functionality probably can be done without major changes to existing code.

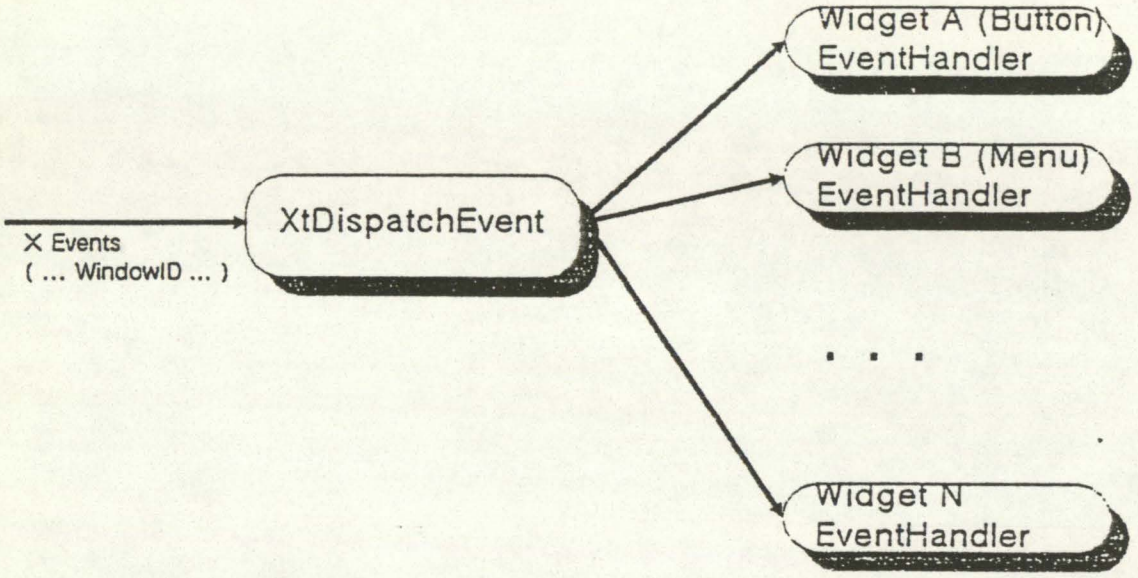

Fig. 4: Dispatching X Events to Widgets

4.1. Mapping THESEUS Concepts to X

In the following, it will be described how the concepts and architectural properties of THESEUS can be transformed into the X world.

Window Management.

The window management area probably is the one that is influenced most by the change of the underlying base system. Some special problems will be discussed in a later section.

In the X implementation of THESEUS, all the border interaction zones and the work area of the window itself will be implemented as widgets. A geometry manager has to be written that provides for the layout of a THESEUS window as seen in the current implementation. Many of the border zones will be instances of a predefined Button widget class.

For the work area, a "THESEUS work area widget" class will have to be created. The window redraw mechanism will be pretty much the same as in the current implementation. It will be possible to gather expose-events for the work area widget and to draw all output objects looping through the object data structure.

Output.

In the output area, an internal interface exists which makes a transformation of window related output requests into calls to the basic input/output system. This part has to be rewritten, which seems rather straightforward, since in X, output is done in window coordinates, so there is no need for a transformation, and X provides window border clipping automatically. Both this had to be done by THESEUS on GEM. Object Management itself as well as the implementation of the GKS* functionality probably can be done without major changes to existing code.

Input.

On the input side, the rather large central Dialogue Manager which is responsible for processing all input generated by the user will be split up into several parts. Menu and Icon input classes can be broken out into (composite) widget classes of their own. This reduces the class recognition finite state machine, which will have to handle only input events that happen within the work area of the THESEUS window and which correspond to one of the THESEUS Object Identification, Object Movement, Position Area Input or Keyboard Input classes. Thus, the finite state machine will be the basis of the WorkArea widget's Event Handler.

An input transformation from window to world coordinates is still necessary, as the work area may be scrolled using the scrollbars.

The WorkArea Widget Class.

Once THESEUS is ported to X as described above, one can think of extracting the object oriented output system and the input mechanisms related to these objects to implement a version of the WorkArea widget class that fits smoothly into the X Toolkit set of widgets, ready to be integrated into other X Toolkit client applications. The abstraction of widgets is restricted to an X window as the smallest instance for which input may be processed. The WorkArea widget will give applications the ability to create objects, to manipulate them and to process input related to these objects within a widget window. Currently there is no widget class offering this. In order to make the WorkArea widget available for X Toolkit applications, the programming interface would have to be totally redesigned to fit into the X Toolkit concepts. While this is not intended currently, it is at least worth being thought about.

4.2. Resource management

In the X Window system, resource management provides a means to specify the user's preferences of the user interface at runtime. Attributes like the width and height of single named widgets, preferred colour for window background, thickness and colour of window border, and even how user input will be translated into which syntactical or semantical feedback can be controlled by specifying X Resources. They override the default values and actions built into the widget implementations. E.g., creating a command button widget without changing any of the defaults will result in a standard text popping up in a little window which is just as large to hold the text, entering and leaving the window with the mouse cursor will highlite and dehighlight the button changing its border thickness, pressing a mouse button within the button will invert the text, and releasing it will call an application function. All this can be changed at runtime.

For THESEUS, specifying X resources can be used to change some values which are fixed in the current implementation (like the background colour of the work area), to tune some of the systems limits depending on the applications foreseen needs, and maybe even to expand the set of output objects, especially the predefined raster objects.

Not overriding the defaults will result in a well defined standard behaviour of the system.

4.3. Multitasking issues

THESEUS originally was originally designed without any implication of its implementation being based on a single- or multi-tasking system. The programming interface is free of assumptions about other applications running or not running at the same time.

However, there are several areas where the straightforward projection of the current implementation on MS-DOS into a multi-user/multi-tasking environment may raise some problems which will be discribed below.

Window Managers.

As the main value of THESEUS lies in the way that the programmer's interface supports output objects and input abstractions to an application, not in the specialty and flexibility of its user interface, it was decided that THESEUS on X should not replace other window managers running on the workstation. THESEUS is linked to an application as a set of library functions, providing the user interface for this application only. Several THESEUS applications may run at the same time without interfering each other.

Screen Layout.

In the current implementation, icons are being drawn on the screen background. As there is only one application, it is obvious that the icons belong to this application. This is not the case in a multitasking environment. Visual hints have to be added to make an icon's belonging to a certain application apparent to the user. These visual hints may range from giving the same border colour to the icons and to the windows of one application up to presenting a "virtual screen background", i.e. a window that contains all the icons and THESEUS windows of one application. The latter approach relieves the problem that occurs when an application window is raised to the top by means of the system's window manager, possibly leaving this application's other windows and its icons hidden behind some other processes' windows. Moreover, THESEUS applications may want to have finer control over its windows' initial placement when the window contents belong closely together. This, too, would be made easier using a common background window as a reference to which all icons and windows could be positioned.

A problem with this is how to select the size of the background window. For porting already existing applications clearly a background window having the size of a PC screen would be ideal. This would make THESEUS on X look more like a PC emulator window. This technique clearly may not be acceptable for applications originally designed for the UNIX environment or for applications that just use one single window. A solution could be to use an initialization parameter to tell THESEUS about which technique should be used. Initialization parameter were built into the THESEUS programming interface as reserved but so far unused, foreseeing purposes like this one. Another possible way would be to use the X resource mechanism, giving a virtual background screen of fixed size as default if the application or the user does not explicitly specify that another size or no background window at all is desired.

Inter-application communication.

In a multi-programming environment like UNIX, especially on a workstation, where many application are visible to the user and interacting with him at the same time, it becomes apparent that there is a need for applications communicating with each other to share information and data. One technique typically used in text windows, but also in graphics programs

is to cut out information from one application's display into a special memory pool and to reinsert this data into another application. Mechanisms to do so are often supported by the operating system (e.g. UNIX message queues or named pipes) or by the windowing software (clip areas or cut buffers). This is not very complicated for text, but if applied to other data, conventions have to be set up about the data types to be transmitted this way and about the format the applications use to store and recall special types of data from the buffers.

While applications are free to communicate using operating system facilities or a common data base, there currently is no mechanism in THESEUS to share user interface data. Therefore, the application interface has to be enriched at least by functions for saving selected user interface data (of both dialog state and output objects) in a file, which could then also be used to suspend a running application and resume it at a later time.

Another issue is how applications can share common user interface facilities at runtime. It is imaginable to have a window in which different objects are controlled by different applications. This would have to result in a totally different structure of the THESEUS implementation where THESEUS runs as a centralized server, executing all output and dialogue control requests of all applications, and dispatching input events to the applications not only on a per-window, but on a per-object or a per-input-element basis.

Once THESEUS is running as a separate process, however, one can experiment to extend THESEUS from the fully synchronous manner of input processing to dispatching events asynchronously while the application may do other tasks at the same time. A mechanism for the application to find out or be informed about the current user activity or inactivity would have to supplied.

Another advantage of separating THESEUS into a process of its own is that there is no code duplication of the user interface any more, which makes it easier to control version updates and to ensure the uniformity of the user interface after changes to THESEUS.

5. Conclusion

THESEUS is a system that provides graphical user interfaces, offering an application-oriented programming interface. Porting THESEUS to the X Window System will make it available on a large number of workstations. It has been shown how the X and X Toolkit concepts support the tasks of THESEUS and reduce complexity in its implementation. Some issues raised by switching over into a multi-tasking environment have been pointed out and some ideas have been presented of how future extensions to THESEUS or new versions could look like.

Literature

Gettys88:

Jim Gettys, Ron Newman, Robert W. Scheifler. *Xlib - C Language X Interface: X Window System, X Version 11, Release 2*. Cambridge; Maynard, 1988.

Hübner87A:

Wolfgang Hübner, Gregor Lux-Mülders, Matthias Muth. *THESEUS: Die Benutzungsoberfläche der UniBase-Softwareentwicklungsumgebung*, Beiträge zur Graphischen Datenverarbeitung. Berlin [etc.]: Springer, 1987.

Hübner87B:

Wolfgang Hübner, Gregor Lux-Mülders, Matthias Muth. "Designing a System to Provide Graphical User Interfaces: The THESEUS Approach". *Eurographics'87*, G. Maréchal (ed.). Amsterdam [etc.]: Elsevier, 1987. pp. 309-321.

Lux-Mülders88:

Gregor Lux-Mülders, Wolfgang Hübner, Matthias Muth, Udo Brand, Thomas Nötling. "An approach for the integration of general purpose graphics systems and window management", *The Visual Computer*, (4)1988, pp. 159-171.

McCormack88:

Joel McCormack, Paul Asente, Ralph R. Swick. *X Toolkit Intrinsics - C Language X Interface: X Window System, X Version 11, Release 2*. Cambridge; Maynard, 1988.

Swick88:

Ralph R. Swick, Terry Weissman. *X Toolkit Widgets - C Language X Interface: X Window System, X Version 11, Release 2*. Cambridge; Maynard, 1988.