S10-61

57690

N92-16578

P- 8

CB609194

A CLIPS/X-WINDOW INTERFACE

Kym Jason Pohl

CAD Research Unit
California Polytechnic State University, San Luis Obispo

Abstract. This paper describes the design and implementation of an interface between the CLIPS expert system development environment and the graphic user interface development tools of the X-Window system.

The underlying basis of the CLIPS/X-Window interface is a client-server model in which multiple clients can attach to a single server that interprets, executes and returns operation results, in response to client action requests. Implemented in an AIX (Unix) operating system environment, the interface has been successfully applied in the development of graphics interfaces for production rule cooperating agents in a knowledge-based CAD system. Initial findings suggest that the client-server model is particularly well suited to a distributed parallel processing operational mode in a networked workstation environment.

INTRODUCTION

Graphic user interfaces are gaining in importance in all computer application areas. Once the almost exclusive domain of CAD users such as architects and engineers, who have traditionally used drawings to visualize design solutions, they are today the preferred medium for virtually all computer-user interactions. A recent study of experienced and novice workstation users in business offices indicated a significant increase in productivity and quality of tasks performed with graphical user interfaces, for both groups (Temple et al. 1990).

Clearly, navigation through applications and the selection of functional options in a window environment with pointing devices is far superior to typed commands and keyed data entry. However, an equally important advantage of graphical user interfaces is the greatly increased 2-D and 3-D visualization capabilities that can be integrated into the application environment. Particularly with the emergence of more complex application systems involving expert systems, distributed databases and integrated parallel processing in networked workstation environments, the need for complex data display capabilities has become no less important than operational efficiency.

Several considerations drove the development of the GXI graphics interface builder. First, it was recognized that data displays should convey not only values but also the context in which the data values exist (Tufte 1990, Abler 1989). Unfortunately, the majority of data displays seen today, such as tables and point line graphs, are two-dimensional in character. This does not recognize the fact that the human user lives in a three-dimensional world and has excellent facilities for perceiving and reasoning about complex solid images. With the decidedly higher level of computer-based graphics capabilities available today data displays should no longer suffer from these limitations. Utilizing advanced graphics programming tools, such as MIT's X-Window system, data can now be expressed in three-dimensional graphs or real world solid objects.

Second, recent advances in computer hardware, examplified by IBM's RS/6000 and Hewlett Packard's HP-700 workstations, provide support for more sophisticated applications software systems. Representing a third generation of workstations with greatly improved reduced

PAGE 114 INTENTIONALLY BLANK

PRECEDING PAGE BLANK NOT FILMED

instruction set computing (RISC) technology, these computers provide speeds in excess of 50 million instructions per second (MIPS) and more than 64 million bytes (MB) of fast memory. Combined with a multi-tasking operating system, such as UNIX, these workstations are capable of supporting applications software packages in which multiple processes interact with each other. For example, knowledge-based design systems in which several expert systems interact with each other through a blackboard control mechanism, while they evaluate the evolving design solution in real-time (Myers et al. 1991). While multi-tasking has opened a wide range of new applications opportunities, it has also added a new perspective to the object-oriented software design approach. Software objects can be treated as semi-autonomous processes, executing concurrently on one or more workstations and communicating with each other through sockets or similar inter-process communication facilities.

Third, an increasing involvement of applications experts with limited computer science knowledge and skills in software development is establishing a demand for higher level programming tools. It can be argued that since this trend is likely to lead to more useful and effective applications software, every effort should be made to provide software tools that are relatively easy to apply and yet do not unnecessarily constrain the applications developer within simplistic structures and paradigms.

The fourth motivating factor for the development of GXI deals with a crucial limitation of many AI language environments. In the current state of AI, most high level programming environments such as the CLIPS expert system shell are limited in that they have virtually no graphics facilities. Therefore, the programmer is considerably restricted in the quality of the interface that can be presented to the user. To solve this limitation, a set of routines may be developed to extend the CLIPS language to include the interface facitilies offered in such graphical environments as X-Window. Such an extension would allow for considerably more robust and interactive applications in an AI environment.

Within the context of these considerations the development of the GXI interface builder was undertaken by the author in response to several needs that arose in the CAD Research Unit of the School of Architecture and Environmental Design at Cal Poly. The interdisciplinary nature of the various project teams established the need for a set of higher level tools that could be used by architects and engineers for prototyping software modules. Typically, this work includes expert systems, databases and procedural programs. Any meaningful evaluation of these prototype models requires the involvement of practising architects and engineers, who would be unduly influenced in their assessments by operational complexities and unrealistic user interfaces. For this reason the availability of a graphics interface builder, serving as a high level development tool for applications experts with limited software engineering background, became a high priority requirement.

ALTERNATIVE APPROACHES

Two main approaches were considered during for the design of the GXI interface builder. The first was based on the concept of providing a set of client graphics calls along with their implementation in one physical process. This approach brings with it some distinct advantages. It allows the client requestor and the graphics server to exist as one cohesive process. Accordingly, all graphics actions can be centralized on a single machine alleviating the need for the software developer to deal with the complexities of a networked environment. The second advantage is also related to programming simplicity. The single process approach allows the programmer to link directly to a library of robust graphics routines, for creating interactive menuing systems and graphical objects.

However, the single process approach also has several inherent disadvantages. As mentioned earlier, both the graphics requestor and the graphics server reside in the same process. Therefore, several graphics applications running concurrently cannot be physically or logically connected. Each exists as an independant entity completely insulated from the other. This must inevitably lead to duplication of code and sequential processing of graphics requests.

A second deficiency arises when the application system resides in a networked environment. The single process approach makes provision only for dealing with the local domain environment. In view of the numerous advantages of telecommunication networks, the requirement for interprocess communication across a network has become a high priority consideration. The single process approach provides no facilities for interprocess communication on the current machine let alone across a network. This limitation simply becomes too costly when application systems of larger size are considered.

The third, and perhaps most serious disadvantage of the single process approach is related to the architecture of the X-Window graphics system, which was mandated as a precondition for the targeted user environment. X-Window provides a collection of graphics primitives to the application programmer. However, these primitives exist in basic form and require extensive programming knowledge of the X-Window graphics environment. While the GXI interface builder was expected to utilize these tools extensively, it was considered important that the complexities of the low level tools be hidden from the software developer.

It is relevent, at this point, to briefly discuss the method used by X-Window to accept, perform and reply to application graphics requests. When the application makes a request of the X-Window server an event is placed on an output queue located on the server side. This output queue is unique to each application client and can be readily accessed by its owner. Event structures contain all of the information required by the X-Window server to bring into existance the particular request or event. When the application wishes to execute an event, it simply removes this prepackaged event from the queue and dispatches it through a series of calls to the X-Window server. To simplify this process even further, the application has the option of entering into an event loop which monitors the application's output event queue. When an event is loaded onto the queue, it is immediately dispatched. At any given time if there are no more events left on the queue, the application may choose to block (ie., sleep) until another event is posted to the queue. Such events can be sent as the result of either a client graphics request or a mouse interaction by the user with an active menu button.

Adhering to this single cohesive unit approach means that the X-Window server must give up control to the application once it has fulfilled the clients request. This has two serious shortcommings. The first is related to a peculiarity common to most large scale graphic tools systems. As mentioned earlier, the X-Window system is a complex system consisting of several processes which have the ability to communicate with each other across a network. Using the single process approach, control moves from the client application to the request server upon the issuence of a graphics request. With control delegated to the server side, the client is put to sleep while the server attempts to carry out the particular request. This is analogous to the situation which arises when a program wishes to read a number from the keyboard. Once the program has issued the appropriate system input call, control moves from the program to the operating system. The client program is simply blocked until the user enters a character and presses the 'return' key. As soon as the user has completed the entry sequence, control is returned to the program. However, during the interval the operating system is able to capture other requests or events independent of the particular client application. This is due to the fact that the operating system itself consists of several processes. It is therefore apparent that in the case of an operating system, control actually resides in several places at the same time. The ability of the operating system to be receptive to multiple client application requests is typical of most client-server relationships.

The X-Window environment is also based on the client-server model. In this environment it is by no means a trivial matter to remove control from the X-Window system. Any attempt to artificially break the event catching and dispatching loop of X-Window can cause serious synchronization problems between the client and the server. For example, it is certainly possible for a series of queued menu creation events to be dispatched in non-chronological order resulting in the display of a menu without buttons. The ability of X-Window to block itself and subsequently wake itself up again, is typical of many large scale graphics tool systems. It was considered important that the GXI interface-builder be compatible with this kind of control environment.

This concept of movement of control throughout the system illustrates the second shortcomming of the single process approach. Once the graphics server has carried out the particular client request, there is no other option but for the server to relinquish control to the client. This means that the server has been literally put to sleep, and remains in this dormant state until the the client makes another request to the server. Since the very essence of an interactive interface is predicated on continuous receptiveness to I/O activity, the single process approach is highly undesirable.

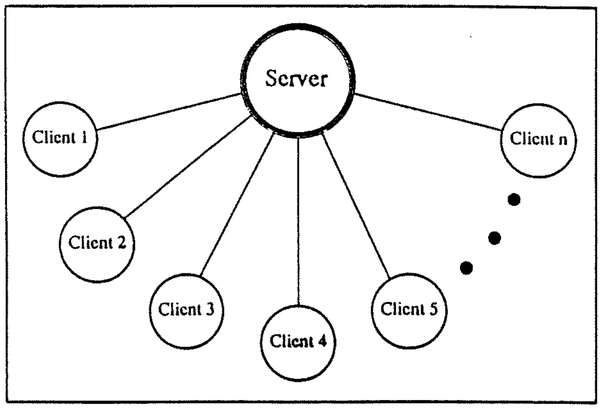

Figure 1.
Client-Server Model

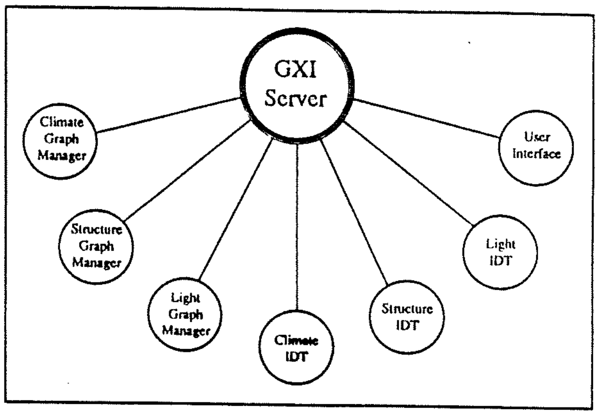

Figure 2.
GXI Application

A second, and considerably more favorable, approach to solving the design problem posed by the GXI interface-builder involves multiple processes (Fig.1). One of these processes is a graphics request server. Similar to the previous models, client applications make graphics requests of the server which in turn performs the necessary work and returns the results (ie., a handle to the graphics object). The clients making the requests need not be concerned how their graphics requests are being carried out. Thus, the client application is removed from the complexities of internal graphics processing and representation.

The advantages of the client server approach are threefold. The first advantage deals with the problem of control. As mentioned earlier, serious synchronization conflicts can arise when control is forced upon the system. Using a client-server design, the server along with each of its client applications resides in its own process. Therefore, similar to the operating system case discussed previously, control resides in several places concurrently. These independent entities communicate with each other through a message passing procedure, thus allowing the server the freedom of blocking and subsequently waking itself. Therefore, the server can be receptive to any interactive activity taking place under the direction of the user, independantly of the current state of the clients.

Second, the logical and physical connection which was lacking in the single process server design is now present in an organized and complete fashion. There exists only one graphics server which performs all of the graphics work requested by each client. This is true no matter which client on which networked machine is actually making the request. The server can be thought of as an invisible workhorse running in background somewhere on the system. Each client simply requests a connection to the server at the beginning of the session. Since all requests are made to a centralized server, there now exists a logical and physical connection between each of the server's clients. Even though all information is channeled through the server, graphics data can be easily passed from one client to another via the server.

The third advantage deals with the potential for distributing the work load generated by a sizable applications system over several networked workstations. The distributed processing model allows the client processes and displays to be assigned independently of each other to any machine/monitor on the network.

IMPLEMENTATION OF THE GXI INTERFACE-BUILDER

The GXI server is designed to accept any number of client applications written in either the 'C' programming language or the CLIPS expert system shell (Fig.2). Since the design of GXI is based on a client-server model, it adheres closely to the guidelines commonly set forth for such environments; namely, multiplicity of clients, parallel request handling, management of the client environment and a common communication protocol (Stevens 1990, Tanenbaum 1987). The two principal entities of the client-server model, the server and the client, must be designed to support a cooperative environment in which requests from its various clients are satisfied with minimum delay.

Typically the server exists in an endless loop performing three basic functions: accepting a client request; performing the requested work; and, then returning the results of the operation to the client. To allow parallel request handling, these three functions are divided between a mother server and a client request handler.

The first function, to accept requests from any of the clients, is accomplished through the use of internet sockets. The server simply waits on the socket for the next request. Once a prospective client has requested a connection to the server, the mother server forks an identical image of itself to handle the request of that particular client. This child process is referred to as a client request handler.

In the GXI implementation, as soon as the client request handler has been created, the mother server is free to return to the top of the loop where it waits for the next client connection request. The child server performs the same three functions as the mother server, however, dealing exclusively with its own client. This request handler will repeat these functions until the client requests termination of its GXI session. At this point the child server terminates itself.

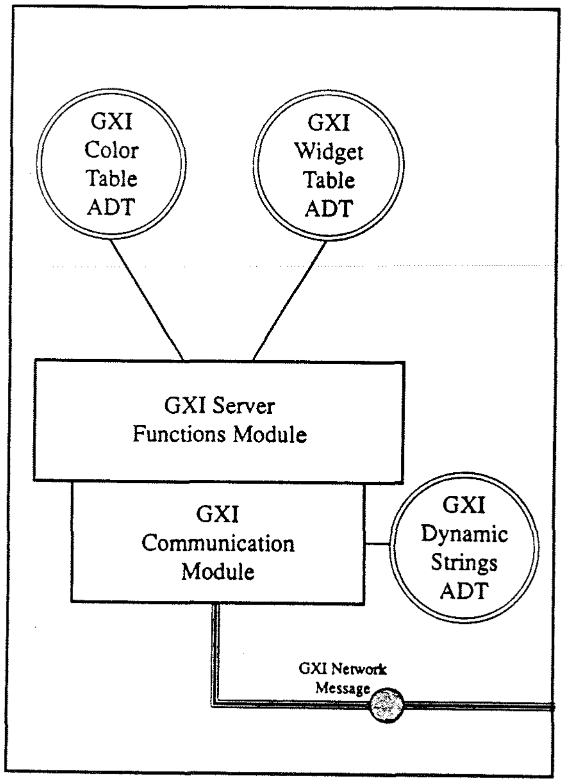

Figure 3.
GXI Server Architecture

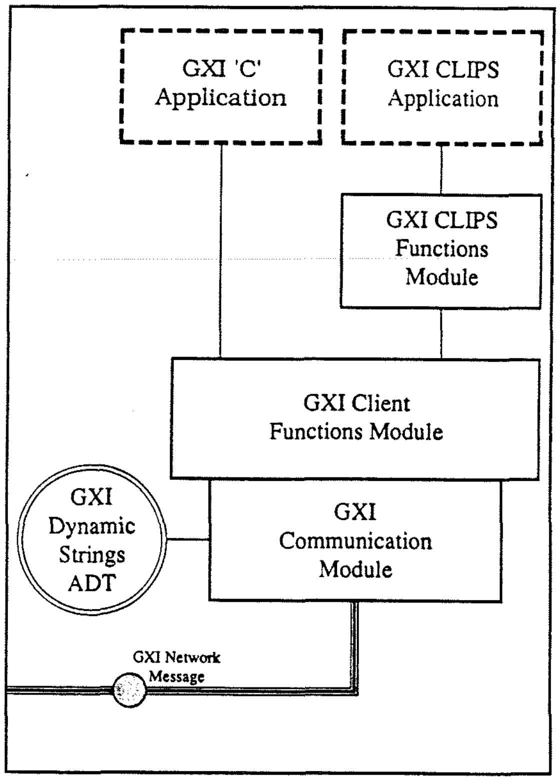

Figure 4.
GXI Client Architecture

To aid the child server in managing the particular environment its client is creating, it keeps an environment table. Among other information this environment table keeps track of the relationship of the graphic objects the client has created and the specific handle of each object that was returned to the client upon creation (Fig.3). To be more specific, each graphic object, such as a menu or a graph, has an associated handle which is used by the client to identify that particular object at a later time.

In addition to these tasks the client request handler also performs maintenance on the client's graphic environment. This may take the form of redrawing a newly exposed region of an object, or maintaining a color table. By caching the color values in a hash table a significant increase in performance is achieved.

The tasks described above for the child server are performed in a manner which is transparent to the client. A communication module which handles the dialogue between the client and the server is attached to each client (Fig.4). When the client issues a request to the server, a message is formulated according to a common communication protocol. This protocol is designed to allow for the transfer of both static and dynamic information. For example, client requests dealing with the number of buttons to be displayed in a menu or the number of points to be used for defining a polygon are not predefined from the point of view of the server. They require a dynamic data transfer capability that must be accommodated by the communication protocol. Once the message has been sent to the server, the client assumes a sleeping state until the results of the requested operation are returned.

The following is an example of an interractive session between a client and the GXI server. In this example the client creates a simple user interface in the GXI environment.

```
(deffacts buttons (Buttons      "Access Database"
                                "File System"
                                "Help"
                                "Exit")
)

(defrule CreateThermalInterface
        (Buttons $?MenuButtons)
=>
        ; ********* Request a Connection to GXI    *************

        (bind ?Sheet  ( XCInit          ?Client ?ColorFile ?XPos ?YPos
                                        ?Height ?Width ?BorderWidth
                                        ?BorderColor ?BkgColor
                      )
)
        ; ********* Create an Interface Banner    *************

        (bind ?Banner ( XCBanner        ?Sheet ?BannerText ?BorderWidth
                                        ?Length ?TextColor ?BkgColor
                                        ?FontStyle
                      )
)

        ; ********* Create the Main Menu         *************

        (bind ?Menu   ( XCMenu          ?Sheet NULL ?Banner ?BorderWidth
                                        ?BorderColor ?TextColor ?BkgColor
                                        ?Horizontal ?NumButtons
                                        $?MenuButtons
                      )
)
        ; *********  Assert the New Environment     *************

        (assert(ThermalSheet  ?Sheet))
        (assert(ThermalBanner ?Banner))
        (assert(ThermalMenu   ?Menu)
)
```

At this point the client may choose to read from the menu it has just created. After the user has made his or her menu selection, GXI utilizes the extensive pattern matching capability offered in the CLIPS environment. The client rule having the appropriate button pattern will fire thus performing the action corresponding to the selected button. Each menu button should have an associated action rule adhering to the following format:

```
(defrule AccessDatabase

        ; *********  Match on the Database Button    *************

        ?Selection <- (Thermal "Access Database")
=>

        ; *********  Perform Thermal Database Access *************
        ;    ...
        ;    ...

        (retract ?Selection)
)
```

TYPICAL APPLICATION

The GXI interface builder was first applied in the computer-aided design field, in the domain of architectural design. In the building design process it is useful for the designer to examine hourly temperature data for an average year for the site. By comparing these values with the range of human comfort, the designer can establish which times during the year require heating, and which require cooling. This information is usually presented in a table with rows of months, and columns of hours in the day.

However, using the client-server model of GXI, these climatic data are passed from an expert system, to a graphic display program connected as a client to the GXI server. Using the lines, polygons, and other primitives provided by GXI, the data are represented graphically as a 3-D contour model, with the x-axis representing months, the y-axis representing hours of the day, and the z-axis representing temperature. By filling each polygon in the contour model with a color relating to its level of comfort (shades of blue for cold areas and red for hot areas), the relationship of temperature to human comfort is also incorporated. More detailed information about a certain month or time of day, can be displayed by taking sections of the contour model. These sections, selected by the user, are displayed as 2-D graphs, and multiple sections can be chosen and superimposed on each other to compare different months or times of day. Each graph can be resized and rotated, and by saving data from the expert system in a file, multiple climates can be displayed side by side, allowing comparison of different sites.

The entire client is mouse driven, using buttons, dialog boxes, and pull-down menus provided by the GXI server. By using GXI to display climatic data graphically, the user is able to examine hundreds of points of data, define custom views, and quickly evaluate the climatic conditions of the site.

A second application involved the generation of space layouts during the earliest stages of building design. GXI was used to build a graphic display facility for a layout advisor written in CLIPS (NASA 1989). In this case, GXI provided a rich selection of graphic object manipulation tools embedded as user defined external functions in the CLIPS programming environment.

CONCLUSION

The GXI graphics interface builder responds to the needs of an increasing number of domain experts who wish to build technically sophisticated applications software, incorporating high quality graphical user interfaces, without having to deal with the complexities of low level procedural languages. The client-server implementation model was found to be particularly suitable to multi-process applications in which the performance of the application system as a whole depends largely on the efficacy of interprocess communications.

REFERENCES

Abler F.(1989); 'Metashapes: Voxel Data Analysis for Computer Aided Design'; Design Methods and Theories, 23(4) (pp.1088-99).

Byrd L.(1991); 'PROLOG and Client/Server Information Systems'; Computer Language, March (pp.37-43).

Myers L., J.Pohl and A.Chapman (1991); 'Computer-Based Intelligent Design Assistance: Concepts and Strategies'; First International Conference on Artificial Intelligence in Design, Edinburgh, Scotland, June 25-27.

NASA (1989); 'CLIPS Reference Manual (Version 4.3)'; Artificial Intelligence Section, Lyndon B.Johnson Space Center, NASA, May.

Stevens W.(1990); 'UNIX Network Programming'; Prentice Hall (pp.137-169, 258-339).

Tanenbaum A.(1987); 'Operating Systems: Design and Implement- ation'; Prentice-Hall (pp.40-2, 51-75).

Temple, Barker and Sloan Inc.(1990); 'Smile when you say GUI'; Trends to Watch, Computer-Aided Engineering, September (pp.23).

Tufte E.(1990); 'Envisioning Information'; Graphics Press.