REFERENCES


X Windowing for Embedded Systems

Larry Mittag
Ramona, Calif.

Larry Mittag is an independent consultant specializing in real-time embedded systems. He has 12 years' experience in the field and is a contributing editor to Embedded Systems Programming.
INTRODUCTION

The X windowing system is rapidly becoming the standard interface for the UNIX workstation world. There are continuing arguments over the particular flavor (MOTIF or Open Look), but both these systems are based on X. This brings a degree of interoperability previously unheard of to the workstation environment.

This is a very nice situation for the UNIX hackers, but are there opportunities in this for the embedded systems programmer? At first glance, it doesn’t appear so. The standardization of X is achieved at the cost of significant overhead, due to the necessary generality of the code. This has traditionally been the bane of embedded systems. Also, embedded systems often don’t call for much of a user interface at all, much less a sophisticated CRT windowing system. Can the embedded systems programmer safely ignore the X wave as irrelevant to his speciality?

I maintain that if he does, he may be missing a good bet. A standardized graphics interface opens up some interesting possibilities for all phases of an embedded system development. I will attempt to describe a few of them in this paper.

OVERVIEW OF X

The X windowing system is essentially a standardized way for an application program to interact with a graphics terminal, just as ASCII is a standardized way of sending characters to a text device. There are other standards that allow some degree of interaction, but the “smarts” (and therefore the overhead) are always located in the computer, as opposed to the terminal. In X, the terminal is a computer in its own right and assumes a large part of the load.

The formal terminology has a host and a display server. These may be the same machine, as in the case of an application running on the same workstation that is being used to generate the window displays, but they can also be completely different types of machines connected by a LAN.

The interface between the host and server is based on notification of significant events. The host will notify the server that a window should be opened, and then selects the occurrences that it is interested in hearing about. If the application requires that the host program be notified that the mouse cursor has entered its window, a message will be sent when that happens. Events which have not been selected will pass unnoticed by the host.

EMBEDDED SYSTEMS APPLICATIONS

Given that this capability is available, what good is it to the developer of an embedded system? Here are some examples from all phases of a system.

Prototyping

Prototyping is becoming popular as a tool for concept demonstration in the world of PC programming. Why not prototype the user interface for a VCR or a microwave oven? A virtual version of the control buttons and displays could be presented in a window, allowing early user testing. This also tends to help the situation where engineers are working from the same written specification, but each has a different picture of the system in their head. A picture can literally be worth a thousand words.

System Test

A common sight during test of embedded software is an ASCII terminal endlessly scrolling a series of characters. It is usually either checking off points in the code (Starting up, Initializing hardware, Interrupt received, DEAD) or regularly displaying the value of a variable that is going awry at some point. These techniques were good enough for our grandfathers (well, maybe older brothers) and oughtta be good enough for us. Right?

Maybe not. Picture instead a checklist on the screen. As each step in the program is reached, a checkmark appears beside the item. Suddenly, that missed step becomes glaringly apparent. And picture that stream of numbers replaced by a strip chart, showing that every ten seconds it goes a little more crazy, finally blowing up. The point is, graphics and other new ways of looking at what your program is doing can reap significant rewards.

Final Product

Remember that fake user interface you generated during prototyping? Maybe it will suffice instead of a real one. The VXI followup to the VME bus standard allows creation of high-powered test equipment mounted on PC boards. The resulting measurements have to be reported to be useful, but who says they have to be shown on a traditional panel? Based on some ads I have seen recently, some major names in the test equipment world are picking up on this idea.

PROGRAMMING IN X

The usual reactions to ideas like those above are things like “we don’t have time for that”, “our budget is limited and this is only a small, simple job”, and “those software guys are at it again”. It is certainly possible to spend more time on tools than on “getting the job done”. The key to avoiding this is being able to whip up these radical items quickly. I once spent a couple of lunch hours putting a full-screen front end on a robotic inspection station.
INTRODUCTION

The X windowing system is rapidly becoming the standard interface for the UNIX workstation world. There are continuing arguments over the particular flavor (MOTIF or Open Look), but both these systems are based on X. This brings a degree of interoperability previously unheard of to the workstation environment.

This is a very nice situation for the UNIX hackers, but are there opportunities in this for the embedded systems programmer? At first glance, it doesn’t appear so. The standardization of X is achieved at the cost of significant overhead, due to the necessary generality of the code. This has traditionally been the bane of embedded systems. Also, embedded systems often don’t call for much of a user interface at all, much less a sophisticated CRT windowing system. Can the embedded systems programmer safely ignore the X wave as irrelevant to his speciality?

I maintain that if he does, he may be missing a good bet. A standardized graphics interface opens up some interesting possibilities for all phases of an embedded system development. I will attempt to describe a few of them in this paper.

OVERVIEW OF X

The X windowing system is essentially a standardized way for an application program to interact with a graphics terminal, just as ASCII is a standardized way of sending characters to a text device. There are other standards that allow some degree of interaction, but the “smarts” (and therefore the overhead) are always located in the computer, as opposed to the terminal. In X, the terminal is a computer in its own right and assumes a large part of the load.

The formal terminology has a host and a display server. These may be the same machine, as in the case of an application running on the same workstation that is being used to generate the window displays, but they can also be completely different types of machines connected by a LAN.

The interface between the host and server is based on notification of significant events. The host will notify the server that a window should be opened, and then selects the occurrences that it is interested in hearing about. If the application requires that the host program be notified that the mouse cursor has entered its window, a message will be sent when that happens. Events which have not been selected will pass unnoticed by the host.

EMBEDDED SYSTEMS APPLICATIONS

Given that this capability is available, what good is it to the developer of an embedded system? Here are some examples from all phases of a system.

Prototyping

Prototyping is becoming popular as a tool for concept demonstration in the world of PC programming. Why not prototype the user interface for a VCR or a microwave oven? A virtual version of the control buttons and displays could be presented in a window, allowing early user testing. This also tends to help the situation where engineers are working from the same written specification, but each has a different picture of the system in their head. A picture can literally be worth a thousand words.

System Test

A common sight during test of embedded software is an ASCII terminal endlessly scrolling a series of characters. It is usually either checking off points in the code (Starting up, Initializing hardware, Interrupt received, DEAD) or regularly displaying the value of a variable that is going awry at some point. These techniques were good enough for our grandparents (well, maybe older brothers) and ought to be good enough for us. Right?

Maybe not. Picture instead a checklist on the screen. As each step in the program is reached, a checkmark appears beside the item. Suddenly, that missed step becomes glaringly apparent. And picture that stream of numbers replaced by a strip chart, showing that every ten seconds it goes a little more crazy, finally blowing up. The point is, graphics and other new ways of looking at what your program is doing can reap significant rewards.

If a Lissajous figure of your code were generated, would it tell you anything? How do you know unless you think to try?

Final Product

Remember that fake user interface you generated during prototyping? Maybe it will suffice instead of a real one. The VXI followup to the VME bus standard allows creation of high-powered test equipment mounted on PC boards. The resulting measurements have to be reported to be useful, but who says they have to be shown on a traditional panel? Based on some ads I have seen recently, some major names in the test equipment world are picking up on this idea.

PROGRAMMING IN X

The usual reactions to ideas like those above are things like “we don’t have time for that”, “our budget is limited and this is only a small, simple job”, and “those software guys are at it again”. It is certainly possible to spend more time on tools than on “getting the job done”. The key to avoiding this is being able to whip up these radical items quickly. I once spent a couple of lunch hours putting a full-screen front end on a robotic inspection station.

Mitag X-Windowing 231
Management had maintained that this was a serious application, and time should be spent improving the basic functionality rather than wasted on making it look pretty. After seeing the results, they changed their standard presentation of the system to highlight it.

So can X programming be done quickly? A look at the documentation can be discouraging. The standard documentation included with X is intimidating. It is a reference set, make no mistake, with few if any examples. The source code included with X can provide useful information, but it is generally not well enough documented to explain how X works. An excellent manual set is available from O'Reilly and Associates, but it consists of seven volumes (and counting). Is it necessary to read all of them? Where should you start?

The fact is, there is a significant learning curve before one becomes a competent X programmer. There is a lot of background material to cover. There are, however, a few hints that can save time and frustration.

**Use Toolkits**

This is a significant savings in and of itself. The tendency is to go directly to the XLib (lowest) level, because the calls are somewhat familiar to people who have done graphics-oriented programming in the past. The problem with this approach is the sheer effort involved in reproducing the functionality of currently available high-level toolkits. The exception to this rule might be a systems programmer, who would be doing low-level X programming for a group of high-level programmers. It often helps to have somebody around who understands the underlying system, but it would be difficult to justify the time to bring an entire programming group up to speed.

**Write Event-Driven Programs**

The natural tendency is to attempt to use traditional programming techniques, forcing the X system to do what the program wants when it wants it done, usually in an orderly, sequential manner. X applications work best when they are structured as a series of small, self-enclosed routines that react to events. These routines should have minimal interdependence, so that a change in the order or timing of events doesn’t affect their functionality. This programming model is very object-oriented (isn’t everything these days?), and thereby can gain the advantages promised by that crowd, notably resuability, maintainability, and all those other wonderful -ilities. Interestingly enough, I think some of the best training for this model of programming is working on interrupt-driven device drivers, giving many embedded systems programmers an advantage.

**CONCLUSION**

X Windows is a significant event in the programming world, cutting across a wide range of fields. Embedded systems is potentially one of the areas that can benefit, but that depends on whether the practitioners of the field will use the new tools that are available. It is often difficult to break away from the day-to-day grind of short deadlines, but X and other new technologies have much to offer in return for the investment.