The InterViews Sponsors Group: A Proposal to the MIT X Consortium

Charles Brauer
Fujitsu America

Mark Linton
Silicon Graphics

John Vlissides
Stanford University

September 17, 1990

Abstract

Although C++ offers the opportunity to create powerful class libraries for application developers, these libraries need to be standard across multiple platforms and must evolve over time to support new technologies. We propose the formation of a sponsors group within the MIT X Consortium to define and implement standard C++ class interfaces for windowing applications, and to evolve these interfaces over time. The group will concentrate on native C++ interfaces, rather than interfaces to the Xt and Xlib C APIs.

We propose to use the InterViews toolkit developed at Stanford as the starting point for this effort, from which we will refine, modify, and extend the class interfaces into a standard that is acceptable to the Consortium as a whole. Sponsors will contribute an additional 30% above their Consortium membership fee, to be allocated for development, documentation, and distribution expenses.

1 Introduction

Many application developers are moving to C++ as an implementation language in an attempt to improve productivity through shared software components. One of the keys to achieving this goal is a set of standard C++ class libraries. Standards require a technology that has been tested successfully, a mechanism for achieving consensus, and an evolution path for integrating new technology as it emerges.

C++ is a nearly 10-year-old technology that has seen rapid growth in the past several years. Several C++ class libraries have been built for X applications, including InterViews at Stanford University, ET++ at the University of Zurich, and OI by Solbourne Computer. The experience from these class libraries provides a reasonable foundation to begin defining standard C++ classes for X applications.

The MIT X Consortium is the ideal forum in which to come to a consensus and to provide an evolution path for the future. We propose the formation of an "InterViews Sponsors" group for Consortium members interested in participating in this process. We propose the name "InterViews Sponsors" to indicate that the group will work on native C++ classes, not C++ interfaces to the existing Xt and Xlib C libraries, and to acknowledge that InterViews will be the starting point for the work. However, we are open to the alternate strategy of a "C++ Sponsors" group that encompasses all activities relating to C++ and X.

The reason for working on native C++ classes is to specify class interfaces and implementations using object-oriented programming language constructs. C++ supports data abstraction through protected and private access to class members, and access functions that hide whether data is computed or stored. Virtual functions provide efficient run-time binding of function calls in a way that is easy to program. In general, C++ allows a programmer to work at a higher level, focusing on class design and implementation and not worrying about the implementation of abstraction and inheritance.

The reason for starting with InterViews is that it represents an advanced yet practical technology. InterViews is technically advanced in three important areas:

Flyweight objects InterViews objects are cheap enough to use for individual characters in a text editor. The availability of inexpensive objects is critical if object-oriented techniques are to help application writers construct user interfaces to an application's data.

Graphical objects InterViews defines a set of classes for graphical objects, with support for grouping and connectivity.

Layout InterViews provides a small, powerful set of mechanisms for composing objects into a variety of automatically-controlled layouts.

InterViews is in use at hundreds of sites throughout the world, and has been incorporated into a number of commercial products. The combination of innovative technology and practical experience makes InterViews the best starting point for work on a standard set of C++ classes. To be successful, however, we recognize that is is necessary to evolve InterViews into a standard and to be able to integrate ideas and implementations from other toolkits where appropriate.

2 Goals

The sponsors group will be successful if we produce *a* standard C++ programming interface that becomes widely-available commercially. To achieve this, the group will need to

- Establish priorities
- Coordinate class definitions
- Donate sample implementations
- Propose standards
- Support a core set of classes
- Evolve classes over time

It is important that the sponsors group avoid the slow pace and bureaucracy normally associated with standards efforts. At the same time, the group must carefully produce a first-rate technology that will satisfy the demands of current and future C++ developers. The most important resource is manpower—the individuals who participate in this activity are critical to its success. Thus, the goal underlying all others for the sponsors will be to choose and manage the team that designs, implements, and supports the standard set of classes.

3 Technical focus

The technical goal of the sponsors group is to produce a set of C++ classes that simplify the construction of interactive applications. We divide these classes into two groups: "intrinsics" that define mechanisms for constructing user interfaces, and "GUI classes" that define concrete components for a particular style of graphical user interface. To demonstrate these classes are appropriate, it will be necessary to implement several applications.

3.1 Intrinsics

The ability to apply objects at a fine granularity is necessary for a simple and natural implementation of the user interface to application data. The best example of this effect is in a text editor where each character on the screen is a distinct object. This approach allows orthogonal composition of different fonts, character sets, and formatting algorithms within a single text window.

One effect of constructing a user interface from objects at a fine granularity is that immediate-mode rendering is less often necessary within the application itself. Furthermore, while C++ supports objects much better than C, functions are similar in both languages. Thus, there is little motivation for the sponsors group to be concerned with rendering operations when several rendering libraries already exist for C.

3.1.1 Graphical objects

The object structure representing an output area will have visible objects such as characters, labels, polygons, stencils, and raster images as its leaves. This structure will also contain objects for changing graphics state such as coordinate transformation, clipping, font, color, and fill pattern. In addition, an object may be connected to another object so that it moves or changes appearance whenever the second object moves.

3.1.2 Layout objects

Aggregate objects can tile (side-by-side), stack (only top-most visible), or overlay (see-though) a list of component objects. Composition objects use a compositor object to determine how to split a list of components across several areas, such as line breaks in a paragraph or page breaks in a document.

3.1.3 Window objects

At the top-level, window objects encapsulate window manager conventions and coordinate graph-

ics context to allow sharing among the objects in the window. We assume that applications built with standard C++ classes will automatically follow standard application-level conventions, such as window manager communication and inter-application selections.

3.1.4 Input handling

With fine-grain objects, it is impractical to presume that individual objects detect events (imagine every character in a document being mouse-sensitive!). Our approach is to have a coarser-grain object detect input events and use hit detection to determine what object or objects were referenced by the input event.

3.1.5 Multiple views

User interface objects often present information from an associated data object. In this case, we refer to the data object as the "subject" and the user interface object as the "view". In general, a view defines a many-to-one dependency on the subject and must be notified when the subject is modified.

3.1.6 Customization

As with input handling, customization of object attributes is not appropriate at a fine level of granularity but can be handled by a specific subclass of output object. It is important to support user customization and we assume that the C++ classes will automatically support standard customization mechanisms such as X resources.

3.1.7 Direct manipulation

Direct manipulation of an object may present a different appearance to the user. For example, when moving a complex picture it might be shown as a simple outline. A separate set of classes are necessary to support direct manipulation and isolate the application from hardware-dependent implementation choices, such as overlay planes or xor drawing.

3.2 GUI classes

Classes that define some graphical user interface, including buttons, menus, and scrollbars, are necessary to build demonstration applications. Unlike Xt, which was developed before any standard GUI emerged, both the Motif and OpenLook GUIs are now widely-available. Any toolkit standards effort beginning now must be able to support multiple GUIs and must have implementations available for OpenLook and Motif.

It is difficult in general for a library to support multiple GUIs for a single application, i.e., change an arbitrary application from Motif to OpenLook style without any source modifications. However, one simple feature is a "subclass switch" that allows an application to defer binding of a class for an object to an implementation for a particular GUI. For example, an application could create a scrollbar object that later creates a Motif or OpenLook scrollbar (perhaps depending on an X resource value) and delegates all operations to this style-specific object. Such a switch is similar to what is provided by the Solbourne OI toolkit.

3.3 Applications

A builder of software tools must never lose sight of the end product—making applications easier to build. The best guarantee for success is to build applications in parallel with tool development. At a minimum, the group will use a drawing editor to demonstrate graphical objects and a document editor to demonstrate text and layout objects. A modest user interface builder is also necessary to allow rapid experimentation with the set of classes that are provided.

3.4 Outside of focus

The implementation of the standard classes and the demonstration applications will encounter many problems that are tempting to try to solve as part of the standard. Examples include general-purpose data structures, persistent objects, IPC to remote objects, and tasking. Attempting to standardize these kinds of classes is both inappropriate for the X Consortium and would be very difficult for the sponsors group to accomplish in a timely manner.

The exclusion of this functionality from the group's focus does not exclude the possibility that work will be necessary in these areas. It might be important, for example, to experiment with multi-threaded applications that rely the user interface to run as a separate task. The sponsor's group

sample implementation might therefore include a tasking package that is not required to implement the standard class interfaces.

3.5 Making objects lightweight

One of the most important features of InterViews is lightweight output objects. The key to making objects lightweight is to define operations that *pass* information instead of relying on objects that *store* information. For example, the base class, Glyph, defines a "draw" operation that has parameters for the drawing area (such as a specific window and graphics context) and the position of the glyph in the area.

Glyphs need not store the drawing area and therefore can be smaller and simpler. Positional information, if it needs to be stored to avoid the cost of recomputation, is better located in the container object than the object itself. With this approach, glyphs do not need to store a pointer to a "parent" glyph and can appear several places in a layout. This feature is particularly useful for implementing a table, where an element appears in both a row and a column.

To prove that glyphs are practical, Paul Calder (a Ph.D. student at Stanford) has implemented a simple document editor that stores a glyph for every character. The editor implements real-time TeX-quality formatting by using aggregate glyphs to layout text and figures. The approach greatly simplifies the implementation of multi-font text, international text, and mixed figures and text. The editor's performance is acceptable on current generation workstations.

4 Plan

The sponsors group will be created formally through the normal Consortium mechanism, in a manner analogous to the PEX sponsors group. The sponsors group will immediately create a steering committee that is responsible for directing the group's activities.

4.1 Steering committee

Each sponsor will appoint one person to the committee. The committee will meet approximately every other month to discuss both technical issues and budget matters. The first task for the committee will be to choose a development team that will be directly responsible for class library design, implementation, and support. This team need not be located centrally; indeed, it is likely the team will include people from several different sites.

4.2 Budget

Each sponsor will contribute an additional sum of money equal to 30% of their Consortium membership fee per year to augment support for the group's work. For example, a company paying $50K per year to the Consortium would pay an additional $15K per year as a sponsor. These funds may be budgeted to contract technical writing work, support development at Stanford and/or MIT, contract additional development or testing, and cover distribution expenses.

4.3 Schedule

The first milestone is to create the sponsors group itself. Once the group is formed, one possible schedule is as follows:

Distribute InterViews 3.0	12/90
Propose standard	
Distribute InterViews 3.1	12/91
Adopt standard	
Distribute InterViews 3.2	12/92

We assume that the group will exist for at least two years. The role of the group beyond that depends on the future of the Consortium after 1992.

5 Summary

We have proposed a new sponsors group within the MIT X Consortium to help create a standard set of C++ class interfaces for X applications. The group will concentrate on native C++ classes for fine-grain user interface objects rather than be concerned with C++ interfaces to existing Xt and Xlib functions. We propose to use InterViews 3.0 as the starting point for future design and implementation. The sponsors will contribute an additional 30% above their Consortium membership fees to support the group's efforts.