IBM Object-Oriented Technology Center and Böblingen Development Lab
invites to the

3rd European Symposium on Object-Oriented Software Development

Böblingen, Germany, October 12 - October 15, 1992

The focus of the Symposium will be the introduction of object-orientation into software development at the IBM laboratories in Europe.

Conference Themes

Papers submitted to the conference shall describe practical experiences with the introduction and usage of object-oriented techniques and can be in any area of object-orientation:

- Site reports
- Experience Reports

Topics: **Analysis, Design, Programming, Languages, Graphical User Interfaces, Databases, Reuse, Prototyping, Programming Environments, Case Tools, Development Process, Transition to OO, Coexistence of OO and non-OO, and other Organizational issues.**

Graphical user interfaces applied to map processing: an application within a C++ framework

G.Chiozzi - G.Ghezzi - M.Tucci

IBM SEMEA (*) s.p.a.
Centro Ricerche e Soluzioni Tecnico-Scientifiche
Segrate - Roma
Italy

October 1992

Abstract

Many tools and libraries are widely available for applications in computer graphics, under the AIX (*) operating system. However, they cannot be easily integrated into a coherent programming environment.

On the one hand, X-Windows toolkits are commonly used to build graphical user interfaces, with a standard look and feel, but they lack in pure graphical functions. On the other hand, advanced graphics is achieved by libraries (e.g. graPHIGS (*) , GL) that do not provide high-level primitives for the user interface, i.e. buttons, menus, etc. Such libraries and tools are not easy to be mastered in a uniform environment.

In this paper, we describe an Object Oriented (OO) approach to the construction of a framework, which provides both advanced graphic capabilities and a powerful environment for developing user interfaces.

The application framework is based on a class hierarchy, which encapsulates both AIX X-Windows Environment and graPHIGS, and is implemented in C++ , on a Risc System/6000 (*) machine.

Several advantages stem from our approach. First of all, the development times for new applications are drastically reduced. Secondly, the resulting code is smaller in size and easier to read and to maintain. At the same time, it can be easily extended and reused.

So far, the library focuses on 2D graphic programming, and is in use, as a development tool, by groups in Milan and Rome. As an example, we demonstrate a typical application in the field of map processing: an interactive tool to display and edit the results of automatic symbol detection in a map.

Background and needs

We have designed and implemented an Object Oriented library to answer the needs of the "technical maps" project at the Italian Scientific and Technical Solutions Centre (STCS hereafter). The library is written in C++ and runs under the AIX operating system.

The goal of the library was to provide colleagues involved in the automatic map recognition project with an easy tool to handle both the user interfaces and the graphic output. The processing of technical maps proceeds through several steps. We needed to build Graphical Users Interfaces (GUIs hereafter) with a standard look and feel to present these phases, which are usually interactive: the user dialogs with the application.

While providing our colleagues with the required library, we have tried to build a more general tool.

Objectives and constraints

There were several constraints imposed on the library. First of all, the knowledge of programming languages of the would_be users was very varied: they came from Fortran, C, graPHIGS and X-Windows experiences. Secondly, in the area of automatic recognition algorithms there was a lot of code already written, which we wanted to preserve and reuse.

One of the main needs in the project is the ability of fast prototyping: the algoritms implemented must be visualized as soon as possible to check their validity. Actually, visualization is the main debugging tool. To achieve this, we had to free our colleagues of the burden of learning high level graphic and interface languages, such as graPHIGS and X-Windows, wasting precious time in the task of visualizing the output of their programs and writing a GUI to interact with them. Besides, we wanted these GUIs to have a standard look and feel.

With these constraints and goals in mind, we decided that the solution was an Object Oriented library based on graPHIGS and X-Windows.

Architecture

The library we have implemented is composed by several blocks, represented in the following figure:

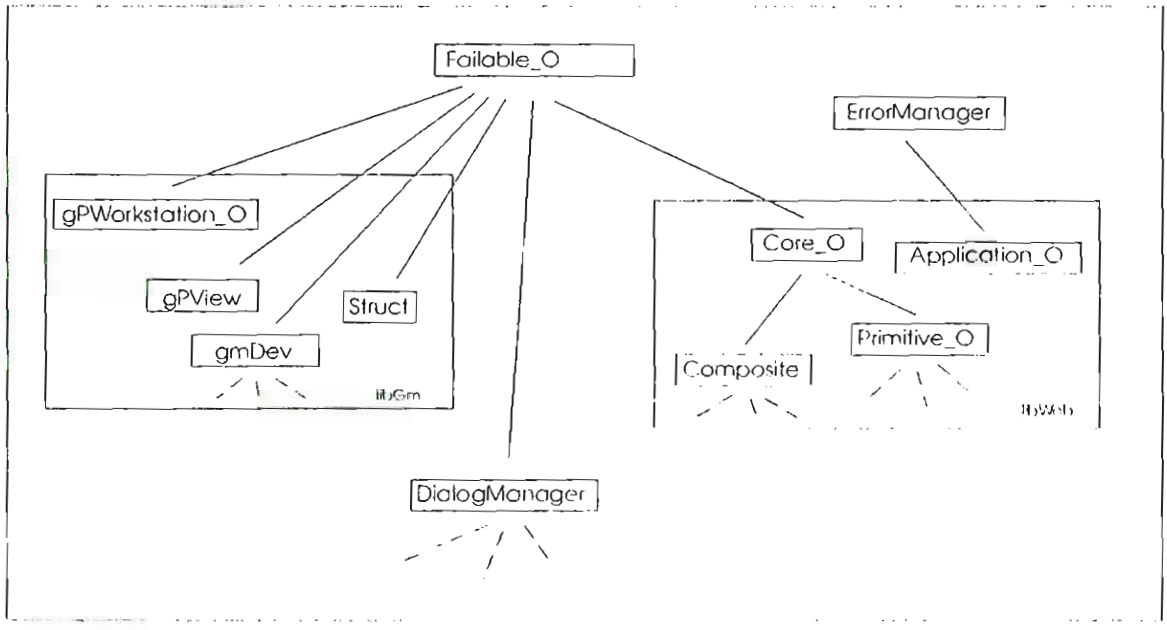

Figure 1. Main classes

Failable_O and ErrorManager

Most of the classes of the library are derived from the meta-class Failable_O. It provides error handling support and relies on the ErrorManager: a class built to collect error conditions at run time and to fill in detailed reports. It makes use of catalogs and the AIX message facility. Each block of the library has its own error message file, which can be easily updated. Moreover, each new class user derived, can have its own message file for new errors, but use the old ones for inherited error conditions. The error report is saved in a log file. It specifies the date and time of the error and, for objects, the instance name, method and error condition:

```
**************************************************
Error messages will be logged starting at  Mon Sep 14 17:51:16 1992

Mon Sep 14 17:51:15 1992:  Application:   Application_O() - cannot open the display
Mon Sep 14 17:51:16 1992:  Test me!:   AddCallback() - cannot add a NULL callback
Mon Sep 14 17:51:16 1992:  Test me!:   SetState() - state value is invalid: use TRUE or FALSE
GPPLCI AFM0092 COLOR INDEX < ZERO
Mon Sep 14 17:51:18 1992:  gmLineColor():  the error is described by the preceding graPHIGS message/s
GPDPL2 AFM0100 NUMBER OF POINTS < ZERO
Mon Sep 14 17:51:18 1992:  gmLine():  the error is described by the preceding graPHIGS message/s
```

The X-Windows wrapper

The X-Windows wrapper, which we have called libWob, is a tree of objects covering part of the Xtoolkit(2). They are derived from the Failable_O class.

XToolkit widgets, though very versatile, are not easy to learn and use. Besides, most GUIs require only a few of X-Windows functionalities. Mirroring it, we have selected the most useful, and transformed them into objects. LibWob objects are used to build the GUI: selection_boxes, buttons, labels, menus etc. are provided.

New classes are easily derived from existing widgets: a template file is supplied with the skeleton of every class and includes of the library. Of course, user defined classes can be derived from every class in the library trees.

The graPHIGS wrapper is linked to this block of the library through the gPWorkstation_O object. It is a graPHIGS workstation, and a child of Core_O, via a more complex hierarchy. It is a peculiar child, though: through it, we realized the shift in programming philosophy from graPHIGS to X-Windows. In fact, all input events from either of the two are driven in a single queue, and then dispatched to the destination object through the callback mechanism.

The graPHIGS wrapper

This part of the library was the most delicate to build. It covers 2D graphics (3). As we mentioned before, graphics is of the utmost importance in our automatic recognition project.

Many of our colleagues knew graPHIGS very well. Those who did not were scared by its complexity, and did not want to learn it. No one had any experience of OO programming, and the fear of wasting a lot of time with this new technique was very spread out. While, at least at the beginning, an OO GUI could be more or less inherited from a ready made skeleton, postponing the moment of learning OO techniques, graphics had to be dealt with directly.

The solution we found was the following: we built a few objects with their methods. These implement the graPHIGS concepts of workstation, view, structure and input devices. But we also wrote a set of "normal" functions, covering graPHIGS 2D primitives, which can be called as usual. Two of these functions deal with structure opening and closing. The functions are written in C++ and heavily exploit the possibility of overloading: in this way we get rid of a lot of parameters, and the user has to specify only the ones meaningful for his purpose. Plenty of defaults are supplied.

In this way we realized a smooth shift from traditional to Object Oriented programming. The users of our library do not have to learn how to derive a new class if they do not feel like. It is necessary to learn some C++ syntax to make an instance of an object, address class methods and make use of overloaded functions. But the main part of graphic programming is made with graphic primitives within structures. And this can be done using the set of overloaded functions provided. On the other hand, we compelled the users to become acquainted with objects and methods: the workstation initialization, view definition, and input processing are made through the corresponding objects. The more they become skilled in using them, the more they shift to the use of the structure object with its methods instead of the overloaded functions.

Moreover, they began to ask for new implementation of classes, and finally they began to build their own new ones.

The Application_O

Every user interface developed with our library requires at least an instance of the class Application_O: X-Windows initialization, display handling, and a number of standard procedures are hidden to the programmer in this way.

The Dialog Manager

This block of the library provides a set of useful tools to handle the communication with the application user. Help and message boxes are provided to display information or warn the user. Data can be entered interactively through panneled dialog boxes. For example, in this way it is possible to let the user choose paths of navigation through the application.

All these pannels and boxes, and methods to manage them, are readily available, saving a lot of time and avoiding repetitive tasks to the programmer.

The map editor

The map editor we are going to show in this symposium is a direct application of our library. The editor visualizes the output of our algorithms that automatically recognize symbols within a map.

By the term "symbol", we usually mean alpha_numeric strings with different orientations, or other signs with a semantic content attached. Starting from the planar graph representing the image (1), several algorithms identify subparts wich could be symbols. To perform this detection, a first screening is made on metrics considerations: we search all the subgraphs whose connected components lay within a certain radius. Then, other considerations arise from the topology of the graph. Besides, to achieve a very high standard of detection, we make use of some semantics too, even though a strong effort of generalization is made.

The map editor visualizes the symbols so detected and labelled.

The main benefit of such an editor is that it provides a powerful debugging tool: this is necessary to make the subsequent step, from symbol detection to symbol recognition. The editor allows a first check of the symbols detected, hence of the criteria used to achieve the detection. It allows the application developer to become acquainted with the graph, moving along its edges and vertices, and to study the tipology and occurrences of the various symbols.

In this way, the process of symbol recognition becomes an iterative one: from a first visualization of the output, hints are derived for a better detection. These are coded and then visualized again for debugging.

The editor allows to visualize the symbols from different points of view: various kinds of highlighting can be associated to graph nodes, graph edges, different ways of classification, different symbols.

Other important features of the map editor are the pick and search functionalities. These too are very useful for debugging purposes. They allow an interactive data retrieval. An element of the graph can be picked through the library object which masks the graPHIGS device, and information concerning it can be obtained. Vice versa, it is possible to look for a certain element entering the relative input via a dialog or selection box.

This map editor is a good example of mixed programming: in a standard C program the classes, objects and methods of the library are used. To build this editor, we used many detection algorithms which were already available at our Centre. Obviously, these parts have not been rewritten, and no classes have been used. Only the graphic display of these algorithms has been changed, using the graphic functions available. The whole user interface has been written with objects and methods of the library. Moreover, we derived new classes to tailor the available ones to our specific needs.

Conclusions

Our library covers a subset of X-Windows and graPHIGS. It is much less versatile than they are. On the other hand, it has proven to be much easier to learn and use.
No 3D graphic functionalities have been implemented, even though the library is easily extensible.
It is available under the AIX operating system only, while it could be ported under VM too.

The use of our library brought many advantages. First of all, the benefits derived from an OO approach: a cut off in the development time, a significant reduction of the code size thanks to the reduced number of instructions, formal neatness of the code. The code is easy to read and maintain, and it is extensible and reusable.
Secondly, the use of the library means a cut off in the learning curve, thanks to the higher level functionalities and tools available and to the simplification of use.
The possibility of "mixed" programming allows a gradual shift to Object Oriented Programming. The library has been written to provide a standard look and feel, but it is possible to generate new applications with different interfaces.

A problem, arisen while the library was under development, is that of documentation and support. Our group is made of several people, some of them working in Rome and some in Milan. Our development standards require that all our source code be well documented. This was sufficient within our group. But soon we realized that our library had become a more general tool. It can have, and has already had, other applications. The problem of a more official documentation arose. Besides, we cannot offer the support that an official product would require. Our management is considering the possible solutions to these problems.

(*) graPHIGS, AIX, Risc System/6000, and IBM SEMEA are trademarks of IBM corporation.

Bibliography

(1) Boatto L. et al., An Interpretation System for Land Register Maps, IEEE Computer, vol. 25, no.27, pp.25-33, 1992.

(2) Young D. The X-Windows System: programming and applications with Xt, 1990, Prentice Hall

(3) Foley J.D. et al., Computer graphics, principles and practice, 1990, Addison Wesley