Graphical User Interface Tools[*]

G. Cosmo *CERN-CN/ASD, Geneva, Switzerland*

Abstract

The use of GUI builder tools in developing modern software applications has nowadays become a requirement. In this paper, a summary of some evaluations tests performed on commercial GUI tools and C++ graphics class libraries is presented. The evaluations have been performed in order to choose a suitable GUI tool to be used for designing the GEANT4 [1] graphical user interface.

Keywords: GUI; Object-oriented design; Widget; Gadget; Notifiers

1 Motivations & evaluation criteria

The *user interface* (UI) software is often large, complex and difficult to implement, debug and modify. A study found that about 50% of the implementation time is devoted to implementing the UI portion and that 48% of the code of applications is devoted to the user interface[2]. In the last few years, there has been significant progress in software tools to help with creating user interfaces, and today almost all user interface software is created using tools that make the implementation easier, make it possible to do fast prototyping, to organise the development process into working-groups, to assure code-rules integrity, and to reuse UI layouts already developed. Moreover, several tools integrate also more or less complete graphic class libraries.
The advantages of using GUI tools in a software development process can be classified into two main groups:

- The quality of the interfaces will be higher;
- The user interface code will be easier and more economical to create and maintain.

1.1 GUI Builders

GUI builders are interactive tools for building graphical user interfaces. Widgets used as building blocks can come from different graphics standards (X, OSF/Motif, Athena, *etc.*). The most important evaluation's criteria we considered in order also to fit with our project's requirements are:

- simplicity of use;
- compatibility and standards;
- completeness and extensibility;
- code generation and C++ solutions.

1.2 Graphics class libraries

Graphics class libraries provide a set of graphic primitives, resources, graphic objects and methods to handle them, which should be used and integrated to compose the interface in the final application. These objects share many common traits and form a natural hierarchy and therefore can be considered as the units of an object-oriented design. These are the criteria we took into account in our evaluation:

- Compatibility and Portability across APIs
- Extensibility and Completeness
  - primitive drawing functions
  - geometrical functions
  - transformation functions
  - events handling and interactors
  - graphics objects/gadgets, persistency and efficiency

[*] Evaluations of three commercial GUI tools

- Object-oriented design
  - polymorphism
  - class hierarchy - Protocols and Containers
  - external objects encapsulation
  - active values
- Maintenance and other more technical features ...

2 Evaluations

The commercial software packages evaluated and presented in this paper are the following:
- X-Designer 3.2
- ILOG Views 2.1
- XVT Development Solutions C++ 3.2

2.1 X-Designer 3.2

X-Designer is an interactive tool for building graphical user interfaces (GUIs), using the widgets of the standard OSF/Motif toolkit as building blocks.
Using X-Designer it is possible to design a hierarchy of widgets simply by clicking on icons and obtaining simultaneously on the screen the tree structure of the hierarchy and an active prototype as dynamic display, showing what the current interface looks like and how it behaves.
How to create an X-Designer application:
1. Building the widget hierarchy;
2. Settings resources;
3. Adjusting the interface-layout using the built-in Layout Editor;
4. Designating callbacks to be associated with individual widgets;
5. Generating code (C, C++ or UIL): code-files for each hierarchy, header files and stubs files containing the function declarations necessary to connect the interface code to the application code;
6. Linking, running and testing.

2.1.1 Features

The tool offers a very clean and intuitive Design Hierachy representation. There's the possibility to fold/unfold subsets of widgets, to highlight with colors those widgets defined as C++ classes, structures or functions, to split the design in more than one shell.
The Standard OSF/Motif widgets are directly represented on a *widget palette* and the user can add directly into the palette user-defined widgets or set of widgets. It is possible to integrate class widgets different from standard OSF/Motif, like *XRT* or *Athena* widgets. X-Designer offers a very powerful control of widgets' resources. The use of inheritance is exploited for managing common resources and methods among widgets. The tool provides built-in Pixmap Editor, Compound String Editor, Layout Editor and a Dynamic display. It is possible to map, unmap, declare, create, destroy callbacks and user-defined methods, adding user-defined method declarations directly at design-level. It is also possible to apply same callbacks to different widgets.
The tool automatically generates code specifications for callbacks and puts the code to be filled by the user in a separated file (*stub-file*). It can generate C, C++ or UIL source code, XPM and UIL Pixmap files, X resource files, Makefile and a *main* procedure. The code is well structured and commented where needed.
The user always has an exact control of resources for each widget (resources can be inserted or not inside the code). It is possible to import pixmap data from external files. X-Designer can be integrated with other packages (e.g.*XRunner*) for testing the generated interface.

2.1.2 X-Designer 3.2, globally

X-Designer 3.2 globally gives a positive impression. It appears to be reliable and complete. Its strong points are:

- Good assembly and intuitive grouping of command types;
- Complete on-line help and text documentation even in matter of error recovery;

The support from the company is good and quite fast. On the other side:

- No possibility to load more than one design at a time;
- Not very easy to understand how to setup and manage inherited resources and methods for widgets defined as C++ classes;
- Compatibility: supporting only X or X/Motif;
- It's nothing more than a GUI-builder !

2.2 ILOG Views 2.1

ILOG Views is an advanced C++ graphics class library; it handles graphics tasks, ranging from basic graphical user interfaces to complex real-time vectorial applications. Interfaces built with ILOG Views are portable across systems including X, X with Motif, Windows, and Presentation Manager. The package is supplied with an auxiliary editing tool called *ilvedit* to help the user designing his interface. The editor itself is written making use of the ILOG Views class library and its source code is delivered with the package, so it is useful to the user for possible customizations or coding templates. Everything is written in pure exemplary C++ style emphasizing code reuse through inheritance.
Classes can be grouped into 7 different domains more or less linked to each other through the use of derived objects or inherited methods:

- *Primitives*, a collection of drawing member functions realizing a sort of connection layer enabling the user to communicate transparently with different display systems (X, Motif or MS-Windows);
- *Resources*, whose class directly interacts with most of the primitives cited above;
- *Views*, the objects directly associated to windows on the screen;
- *Graphics Objects* and *Containers*, a class hierarchy to create and manage graphics objects and containers where to put them in;
- *Gadgets*, the set of classes enabling the user to create interface graphics objects, their interactors and their special containers;
- *Charts, Scales & Gauges*, graphics objects designed to represent data;
- *Management*, the main class which is responsible of the interactions between graphics objects, views and storage places (layers).

2.2.1 Features

In ILOG-Views, the drawing member primitives handle every aspect of the connection with the used display system. Elementary drawing classes includes: point, point with fp coordinates, vector, rectangle, region (list of rectangles), curves, strings, arcs, arrows, polylines, Bezier curves, labels, bitmaps ... . There are primitives dealing with resources: colors, line styles, patt., fonts. and Mouse & keyboard events handling primitives. It is possible to manage any bitmap format (GIF, BPM, BMP), to record events, store them and play back at any speed, to redirect the output on screen or PS file. The package includes 2D transformation, strings, timer and error handling.
A *View* is the real place where graphics objects are drawn on the screen, a window is an associated set of one or several views; visibility, scrolling, drawing, sizing are managed at this level. A graphics object is displayed associating its coordinates with the coordinate system of a particular *container*. The Containers coordinate the storage and the display of graphics objects inside a window; it is a kind of view with predefined Callbacks handling system/user events and the automatic refresh of

graphics objects it stores. Functionalities provided by Containers are: object management, drawing functions, geometrical transformations, buffering, I/O operations, accelerators.
It's possible to set look & feel of graphics objects. Object *Interactors* are defined to manage user events on objects. For each gadget there's a predefined object interactor for containers. At *manager* level the user can handle large sets of objects. The Manager coordinates the interactions between the display of graphics objects in multiple views and the organization of graphics objects in multiple layers.

2.2.2 The graphics editor

ILOG Views provides *Ilvedit* as a tool to build graphical user interfaces using ILOG Views gadgets. Ilvedit is an extensible GUI builder, it provides special panels displaying predefined gadgets and graphics objects; the user must simply pick and drag the object from the panel to the work space to graphically configure his own panel.
It gives the possibility to handle all graphics objects and gadgets supported with the library in a graphic and intuitive way. Attachment and alignment of objects is very easy and practical. It's very easy to extend the editor with user-defined gadgets and graphics resources can be set in a practical way. The tool generates C++ code, defining a class for each panel being built. Generated code is clean, well structured and commented where needed; it is possible to generate callbacks as virtual member functions (just redefine them in a derived class). Resource data can be included or not inside the code. The tool can also generate a *main* procedure.

2.2.3 ILOG Views 2.1, globally

ILOG Views 2.1 is provided as a C++ class library where classes are complete, efficient and extensible. Its power is that it is based on a true object-oriented architecture separating visual aspects from objects' behavior. Benefits of this are class reusability and programming flexibility.
Predefined drawing primitives assure portability accross different window systems (X, X/Motif, Microsoft Windows) making the system independent of platform and graphics standards. Documentation and tutorial on classes and code description are good and sufficiently deep. The possibility to handle the gadget editor source code, which is freely delivered within the package, is very useful.
The GUI builder or gadget editor is provided as a tool. By means of the editor it is possible to handle the most important features and tools provided within ILOG Views. Some solutions adopted with the editor are really successful such as the way to set attachments and alignment for objects and the easiness to extend the editor itself.
On the other hand:

- There's no on-line help or error-recovery;
- *Ilvedit* lacks a sort of dynamic-display;
- A hierarchy display is missing.

2.3 XVT DSC++ 3.2

XVT Development Solution for C++ (DSC++) is a set of products whose aim is to form a productive C++ development workbench. The software components of DSC++ are:

- *XVT-Power++* (release 3.2) - a portable C++ application framework, including the *Rogue Wave Tools.h++* class library;
- *XVT-Portability Toolkit* (release 4.0) - a portable API library that enables XVT-Power++ to easily move across platforms.
- *XVT-Architect* (release 1.0) - a visual object-oriented GUI development tool and application generator.

XVT DSC++ is portable across most major GUI environments including: Motif/X11R5, OPEN LOOK, MS-Windows, MS-Windows NT, OS/2 and Macintosh.

2.3.1 Application Framework

XVT-Power++ incorporates an application framework and encapsulates several hierarchies of object classes to feature an object-oriented design within the field of GUI programming. These objects are used by the designer as building blocks to assemble into the application. The application framework consists of three levels:

- *Application* - program control, application startup and cleanup handling, providing global objects/data and getting access to them, finding out global definitions and creating documents, message propagation between classes.
- *Document* - The *document* is the link between the application level and the view level. *CDocument* is the abstract class for accessing and managing files and internal data, creating windows or variations.
- *View* - Views allow the user to interact with the application and reflect the state of the application. Views draw themselves on the screen. A *Draw()* method is defined for each view, providing automatic clipping.

2.3.2 Features

In an application built using XVT-Power++ there's one *CApplication* object which controls the program once started: it's the first object to be instantiated and the last to be destroyed. The *CBoss* class manages the events and message-passing among classes. Data can be propagated in two different ways:

1. through the *DoCommand* chain mechanism: the document acts as a server to all of its client windows. Windows can ask for a change or pass messages to other windows by simply generating a DoCommand.
2. through the *ADP (Automatic Data Propagation)* system: it plays around the concept of *notifiers*: *providers* and *dependents*.

Views can be built up of other views and a view can contain several other views. *Enclosures* are provided to ensure clipping on included views. Among the predefined shapes are: Rectangle (Square), Oval (Circle), Arc, Polygon (Regular Polygon), Line. They are all derived from *CSubView* and can act as enclosures for other objects, receive and communicate events, be moved/sized, or also generate commands. The package offers helper views providing *stickiness* properties, data structures to manage attributes, helper objects for enabling a view to be moved and sized. It provides classes for inserting, removing, moving objects on cells and sizing grids; classes for handling strings/texts (from *Rogue Wave*); classes for run-time type checking of objects and debugging utilities; classes to automatically handle traversals of menubar hierarchy; methods to read and write several different image file formats: BMP, PICT, xpm, xbm.
Program resources on UNIX platforms are coded using the URL (Universal Resource Language). It is possible to store resources in a separate file (read at launch time by the application) or store them in a separate code-file.

2.3.3 XVT-Architect

Built on top of XVT-DSC++, it offers most of the functionalities of XVT-Power++ to graphically design and build an application. XVT-Architect supports different editors for managing global data and refining the application: Menu, Command, Accelerators, *etc.* It is possible to import/export and cut/paste projects. XVT-Architect consists of three main visual components: the Blueprint, the Drafting Board and the Strata Interface.

- *Blueprint* module, to establish the primary object hierarchy and inter-object communication for the application;
- *Drafting Board* module, to lay out views and subviews making up the GUI interface of the application;

- *Strata Interface*, the attribute editor.

2.3.4 XVT-DSC++ 3.2, globally

XVT Development Solution for C++ seems to be globally a very good product for developing GUI applications within an object-oriented application framework. It can assure portability across the most important hardware platforms and graphics environments by means of a robust and complete API library provided by the XVT-Portability Toolkit component. XVT-Power++ integrates the Rogue Wave's Tools.h++ data structures and its class hierarchy is based on Rogue Wave Collectable classes. That's really a strong advantage of this package, since Rogue Wave class library provides a robust and rich set of collections, utility classes and data structures. Templates are not used within XVT-Power++, however it can benefit from the use of Rogue Wave template classes. Very interesting and powerful is the Automatic Data Propagation (ADP) system to propagate changes of data from objects to objects, as well as the possibility to perform run-time type checking on objects. XVT-Portability Toolkit realizes a robust and reliable interface to operate on UI graphics objects independently from the underlying graphics system or hardware platform, providing in addition a flexible and powerful hypertext-based help system. By means of XVT-Architect, the user can make use of all features and functionalities provided by the XVT-Power++ class library in a very ease and intuitive way. The possibility of import/export and cut/paste projects files becomes very useful to structure the development for a team-oriented organization. The "Factory"-system method is a skilful solution for application maintenance and for reducing efforts when updating or refining parts of the application being developed. Documentation and tutorial on classes and API library as well as XVT-Architect on-line help are good and well organized. Strong point:

- The user always has under control the general structure of the project and the class hierarchy of objects he is working on; it's very ease to skip among them to visualize or change current settings and/or inherited attributes.

On the other hand:

- RW persistence feature is not supported by XVT-Power++ (it will be in a future release);
- Resources management is quite complex, since on UNIX platforms it makes use of the URL language and the final generated application has to be delivered together with a separated binary file containing all specific resource settings (which will be loaded in memory once at launch-time).

References

1 G. Cosmo et al., *GEANT 4 : an Object-Oriented toolkit for simulation in HEP*, CERN/LHCC/95-70 (1995).

2 B.A. Myers and M.B. Rosson *Survey on User Interface Programming. Human Factors in Computing Systems*, Proceedings SIGCHI'92, Monterey, CA, pp. 195-202 (1992).