Available online at www.sciencedirect.com
SCIENCE DIRECT®
Nuclear Instruments and Methods in Physics Research A 559 (2006) 17–21
NUCLEAR INSTRUMENTS & METHODS IN PHYSICS RESEARCH Section A
www.elsevier.com/locate/nima

The graphics editor in ROOT

Ilka Antcheva[a,*], René Brun[a], Carsten Hof[b], Fons Rademakers[a]

[a]CERN, CH-1211 Geneve, Switzerland
[b]RWTH Aachen, III. Physikalisches Institut A, 52056 Aachen, Germany

Available online 7 December 2005

Abstract

A well-designed Graphical User Interface (GUI) has critical importance in any computer application. The user interface is where the end users and the complex system intersect. An effective interface design can make a powerful and complex system, such as ROOT, easy and intuitive to learn and operate. This paper describes the main goals we defined and the design solution we found developing the graphics editor in ROOT.

PACS: 01.05.hv; 07.05.Wr; 07.05.Rm; 07.05.kf

Keywords: Object-oriented user interface; ROOT framework; Data analysis; Graphics editor; Data visualisation; GUI

1. Introduction

ROOT as a high-energy physics (HEP) analysis framework provides a large selection of HEP specific objects and utilities. The graphical capabilities of ROOT range from 2D primitives to various plots, histograms, and 3D graphical objects. All objects are drawn in a canvas or more generally, in a pad, which is a graphical container that holds them. Each pad has a linked list of pointers to the objects it holds. The ROOT software provides all methods for changing the graphical objects attributes programmatically. A graphics editor for setting them interactively was missing.

1.1. Main goals

The object-oriented ROOT framework [1] offers considerable benefits for developing an object-oriented user interface. But, the large set of all HEP specific objects makes it impossible to design a unique and complex interface for everything that can be drawn in a ROOT canvas. The only way to manage this complexity is to split it into discrete units and organize them to handle the user's actions. The main purpose of the graphics editor evolved to respond dynamically to user actions while keeping the users focused on the objects. It should provide a comfortable and efficient environment where ROOT users can get their work done.

1.2. Focus on users

There are two important steps in the GUI design process: identifying users and supporting different user classes. Various schemes have been proposed to classify the different and changing characteristics of users from one time to another. We consider four different classes of ROOT users. The words we use to describe them are less important than the behavioral characteristics they imply.

Novices (*for a short time*) have theoretical understanding and no practical experience with ROOT. They are impatient with learning concepts, but patient when performing tasks.

Advanced beginners (*many people remain at this level*) focus on a few tasks and learn more on a need-to-do basis. They perform several given tasks very well.

Competent performers (*fewer than previous class*) know and perform complex tasks that require coordinated actions. They are interested in solving problems and tracking down encountered errors.

*Corresponding author. Tel.: +41 22 767 6522; fax: +41 22 767 0300.
E-mail address: ilka.antcheva@cern.ch (I. Antcheva).

0168-9002/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.nima.2005.11.113

Experts (*identified by others*) are able to find solutions in a complex functionality. They are interested in the theory behind the design. Very often they search for a way of interacting with other expert systems. The great feature of the graphics editor design was to provide a shorter learning curve for the novices and the depth required by the experienced users.

2. Design solution

Everything drawn in a ROOT canvas is an object. There are classes for all objects, and they fall into hierarchies. In addition, the ROOT framework has a set of classes for developing GUIs. These classes are fully cross-platform and provide all standard components for an application environment with Windows "look and feel". The object-oriented, event-driven programming model supports the modern signals/slots communication mechanism (as pioneered by Trolltech's Qt [2]). It handles user interface actions and allows total independence of interacting objects and classes. This mechanism uses the ROOT dictionary information and the CINT interpreter [3] to connect signals to slots methods.

Therefore, all necessary elements for an object-oriented editor design are in place. In addition, that design gives the possibility for solving the editor complexity by splitting it into discrete units of so-called object editors. Any object editor provides an object specific user interface. The main purpose of the graphics editor is the organization of the object editors appearance and the task sequence between them.

Analyzing all the possibilities provided by the ROOT framework, we decided to follow a simple naming convention: to use as the name of any object editor the object class name concatenated with 'Editor' (e.g. for *TGraph* objects the object editor is *TGraphEditor*. Thanks to the signals/slots communication mechanism and to the method *DistanceToPrimitive*() that computes a "distance" to an object from the mouse position, it was possible to implement a signal method of the canvas that says which is the selected object and to which pad it belongs. Having this information the graphics editor loads the corresponding object editor and the user interface is ready for use. This way after a user click on 'axis'—the axis editor is active, on a 'pad'—the pad editor, on a 'histogram'—the histogram editor, etc.

The algorithm we use is simple and is based on the object-oriented relationship and communication. When the user activates the editor, according to the selected object *obj* in the canvas it looks for a class name *objEditor*. For that reason the correct naming is very important. If a class with this name is found, the editor verifies that this class derives from the base editor class *TGedFrame*. If all checks are satisfied the editor makes an instance of the object editor using the method *TROOT::ProcessLine*. Then, it scans all object's base classes searching the corresponding object editors. When it finds one, it makes an instance of the base class object editor too.

Every instantiated object editor is registered in the editor list of *TClass*. Once the object editor is in place, it sets the user interface elements according to the object's status. After that it is ready to interact with the object following the user actions.

The base editor's class *TGedFrame* is designed with all necessary characteristics of object editors: to hide and show itself; to apply user changes with immediate feedback; to update user interface elements following the object changes, etc.

The complexity of the software system remains hidden into the code. The editor interface is easier to design and adapted to the users' profiles. It keeps users focused on objects, not on how to carry out actions.

The graphics editor gives an intuitive way to edit objects in a canvas with immediate feedback. Related actions work the same way and reinforce the understanding of the functions it provides. Complexity of some object editors is reduced by hiding GUI elements and revealing them only on users' requests.

The graphics editor in ROOT can be embedded or global. The embedded editor is connected only with one canvas and it shows up on the left side of the canvas window (Fig. 1). It provides the user interface for the selected object. If there is no selected object at that moment, the editor provides the GUI for editing the canvas itself. Users can toggle the graphics editor selecting the Editor in the canvas View menu.

An object in the canvas is selected by clicking on it with the left mouse button. Its name is displayed on the top of the editor frame with a set of GUI elements ready for object interaction. If the editor frame needs more space than the canvas window, a vertical scroll bar appears for easy navigation.

The global editor (Fig. 2) has its own application window. It can be connected to any canvas created in a ROOT session. It shows up via the object's context menu available after the right mouse click on an object in the canvas.

3. Editor design elements

The look and feel, as well as the conceptual organization are the three basic elements of any interface design. The "look and feel" elements are closely related. The quality of the output, whether or not the user produces meaningful results depends on the conceptual elements and the relationship between them.

3.1. The "Look"

The graphics editor appears immediately uncluttered and well-organized. It has a dynamic layout that plays around with the editor frame layout at run time. It recalculates the