Dynamic Graphical User Interfaces using XML and JPython

G. Guglielmo

Fermi National Accelerator Laboratory, USA

Abstract

We report on a new evaluation effort for data acquisition for Pixel and other module test stands aimed at producing a framework for dynamically building Graphical User Interfaces. The idea is to allow customization of the control interface to be controlled via an input file. To accomplish this we are planning to use the eXtensible Markup Language (XML) and Jpython. XML is designed to be a universal format for structured documents and data in a web based environment which appears well matched for the control file format. JPython, which is a certified 100 % Pure Java implementation of the object-oriented Python scripting language used on the D0 experiment, allows the use of Java classes, including the Swing classes, and is excellent for rapid prototyping of Graphical User Interfaces. Together XML and JPython promise to provide an easy parsing of the necessary information and the flexibility of rapid prototyping desired in a dynamic Graphical User Interface framework. This work is sponsored by DOE contract No. DE-AC02-76CH03000.

Keywords: DAQ,ONLINE,GUI,XML,JAVA,JPYTHON

1 Introduction

Modern application interfaces are often graphical in nature and referred to as Graphical User Interfaces (GUI's). One of the drawbacks to many of these interfaces is that they provide only a predefined set of options and functionality and thus have a limited user adjustable configuration. Since individual users can have vastly different preferences on how an interface is configured, application designers usually have to take a least common denominator approach to configuration options to keep their projects manageable. Thus, it is uncommon to find an interface that allows the user to dynamically redefine the executable commands controlled by the interface. While this approach is adequate for most users and in most situations, it unfortunately can be a hindrance for experts working in a more demanding environment. We believe that data acquisition and monitoring environments represent situations where greatly enhanced flexibility would be useful, and that modern programming tools such as eXtensible Markup Language (XML)[1], JPython[2] and Java can be used to provide this flexibility.

Our DART/DA system was started nine years ago and parts of that system have been used by 10 experiments to date. DART uses tcl/tk for the user interface. In each experiment between 2 (E835) and 10 (KteV) experimenters have used this tool base to provide GUI's for other parts of their experiment online and data acquisition systems – for detector initialization, monitoring etc. For the CDF experiment, SVX (silicon vertex detector) Diagnostic GUI's are in Java. To date about 8 people have worked on extending and coding the GUI as the system has been used by

[1]A universal format for structured documents and data. See `http://www.w3.org/XML/`.

[2]A Java aware scripting language. See `http://www.jpython.org/`.

the gamut of engineers, technicians, experimenters and integrators needed to fully commission and test the SVX modules. These experiences have taught us the benefits of having flexible, extensible, modular and easy to define and implement, GUI's for data acquisition environments.

The dynamic GUI building effort is part of the Online Data to Experiments (ODE)[3] project at Fermilab. The main goal of the dynamic GUI effort is to develop a framework which will allow greatly enhanced flexibility and customization of application GUI's for testing, data acquisition and monitoring environments. In addition to allowing user configuration of the presentation of the GUI, we are also developing a way for the user to dynamically redefine the underlying code to be executed when a widget is selected (a button click for example). In this paper the current status of this effort as well as future plans will be discussed.

2 The Dynamic GUI Builder

The initial phase of the dynamic GUI building effort was to investigate programming tools and develop a simple prototype for evaluating the feasibility of the dynamic GUI concept. The focus of the prototype was on demonstrating that various requirements for a real system could be achieved as opposed to being an alpha version of the program. Therefore we needed tools well suited for a rapid prototyping environment.

2.1 Programming Tools

We chose JPython, a rewrite in pure Java of the Python Object Oriented scripting language, as the main programming language. JPython is an easy to use scripting language that is Java aware and thus allows one to use the Swing classes for rich GUI development. Scripting languages are well suited for rapid prototyping and thus well suited for the given project. Coupling rapid prototyping with the ability to use Java classes directly meant that re-implementing sections of the code in Java if needed would be fairly easy to do. Java and JPython are also highly platform independent.

Early on we realized that XML, which is like HTML (Hypertext Markup Language) with the ability to define your own tags, was well suited as a language for the configuration information. The advent of XML has given us an appropriate base to build upon. For instance, the decision to use XML meant that we would not have to write our own parser since XML parsers are available from several sources and for several different programming languages. We have chosen to use the xml4j parser, from the IBM AlphaWorks site [4], which is written in Java. The availability of parsers and the numerous examples of projects using XML that can be found on the web demonstrate that XML is rapidly becoming a recognized and accepted tool for software development.

2.2 Basic Principles

The dynamic GUI builder should use a file driven configuration approach which allows a user to easily save their preferred configuration for future sessions. The builder should provide the ability to dynamically rebuild the application GUI incorporating any changes made to the configuration files. There are two types of information relating to an application GUI. The first type is data which covers parameters, actions to execute and documentation. The second type of information is how the data should be displayed. These two types of information should be stored separately since they can be viewed as two different types of configuration. The former represents functionality while the latter represents esthetics. Additionally, an application GUI that has been built should be able to redefine the action code (code executed when a widget is selected) dynamically.

[3] See `http://www-ese.fnal.gov/ods/ode/`.
[4] See `http://www.alphaWorks.ibm.com/`.

The configuration file should be flexible enough to allow documentation to be included along with the parameter data in the configuration file. This allows all of the information associated with the interface parameters to be stored along with the data and minimizes the risk of the documentation getting out of synchronization with the data. Finally, the program should be as platform independent as possible.

3 Current Status

The initial phase of the project has been completed culminating in a prototype version of a dynamic GUI building program. The program is written in JPython and uses XML for defining the configuration data. At the moment the display information is stored as a JPython Dictionary, but this will be converted to a format consistent with the XML philosophy in the future. We are currently using Java 1.1.6, JPython 1.1 alpha 3, and xml4j 2.0.15 on Linux platforms for development.

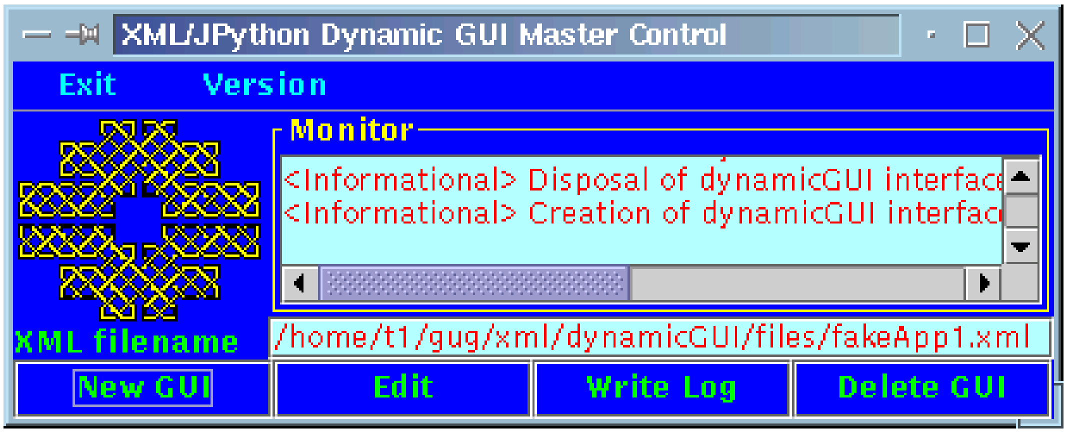

Figure 1: The main control panel which can rebuild the application GUI on the fly.

The dynamic GUI builder has a main control panel which allows a user to create or recreate an application GUI (see Figure 1). All of the colors can be set in the display configuration file. The "New GUI" button when selected will generate an application GUI based on the configuration data stored in the file listed on the "XML filename" line of the main control panel (/home/t1/gug/xml/dynamicGUI/files/fakeApp1.xml in Figure 1 for example). The actions associated with the selection of each widget displayed are named in the configuration data file. If the action is a simple single line of JPython code, then that code can be placed directly in the configuration file. For executing more complex instructions, the name of the method is listed in the configuration file and the method code is placed in a file of user methods and classes. One advantage of using a method defined in the user methods file is that those methods can be re-sourced on the fly to incorporate any changes without having to rebuild the application GUI. The user can also redefine user classes and create new instances in the same manner.

The current prototype only defines buttons and text entry field widgets for each parameter defined in the configuration file. The buttons can be grouped within a page or placed on separate tabbed panes in the application GUI. We have restricted ourselves to this minimal set of widgets in the initial prototype since it is sufficient to demonstrate the feasibility of the approach without the burden of extra complexity. There is also an automatic help window that displays all of the documentation listed in the configuration file. Another feature of the program is that all informational, warning and error messages are written out in XML format to allow for advanced searching techniques to be applied in the future. Figure 2 shows the application GUI generated for a simple binomial distribution test.

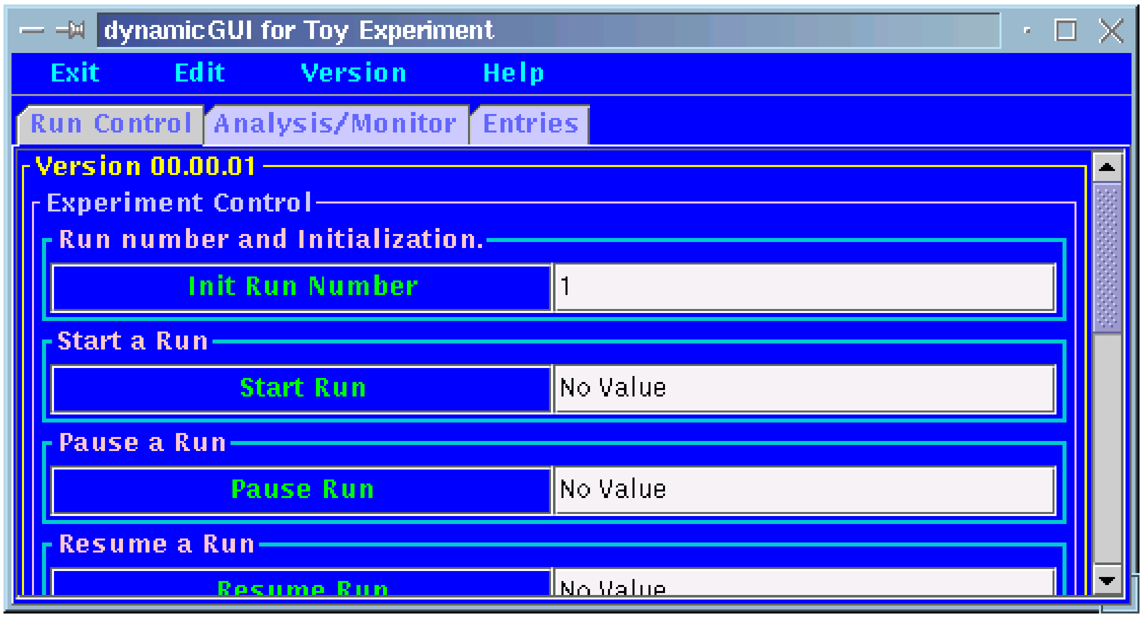

Figure 2: Example of a dynamically created application GUI for running a binomial distribution test using a random number generator.

4 Future Plans

The initial phase has been quite successful and is an encouraging sign as we begin the next phase of the effort which is to design and develop a real dynamic GUI builder. Keeping the data and display information separate will be a challenge, but not an insurmountable one. The design of the configuration infrastructure will have to be expanded. A system for scheduling repeated activities will also need to be implemented. Support for additional widget types is also on the list of things to do. In parallel, work on infrastructure for integrating with other components used in the ODE project will be performed. Some of the other components include LabView, merlin[5] (a message logger), and an electronic logbook. We have also taken note of other High Energy Physics related GUI's at Fermilab such as histoscope, root and jas for example which may prove valuable to the effort.

Performance will be an issue that poses a big challenge for the project. Java is considerably slower than native code like C or C++ because it is an interpreted language. JPython which is also an interpreted language written in Java make matters worse. Initially we will try to keep most of the code written in JPython, and to a lesser extent Java, to make platform independence as easy as possible. If the performance becomes a problem, some sections of the code will have to be rewritten in a native language and incorporated using the Java to Native Interface.

[5] See `http://www-b0.fnal.gov:8000/merlin/`.