A GUI BUILDER ENVIRONMENT BASED ON LABVIEW FOR THE VIRGO PROJECT

F. Carbognani, B. Lopez, D. Sentenac, EGO, Pisa, Italy.

Abstract

In order to support the ongoing VIRGO Detector commissioning activity and facilitate the transition to full operational mode, the need for new, quickly built, flexible and graphically rich Graphical User Interfaces (GUIs) arose. The challenge was to set up a GUI building environment able to deal with those requirements and to smoothly integrate with the existing distributed control software framework (C/C++ based) and associated software management procedures. We have been able to fulfil those requirements by using LabVIEW as our GUI builder environment and by enhancing its functionalities within a three-layer approach: an interface to our distributed control system, a collection of common functionalities and another of autonomous building blocks.

INTRODUCTION

The VIRGO project consist of a suspended Michelson Interferometer with two 3 Km arms aimed at the detection of gravitational wave signals from cosmic sources [1]. During its commissioning phase, on GUIs side, high flexibility and an efficient way to gather user requirements were needed. In this sense, applying the same development life-cycle used for control applications revealed not to be appropriate and a new strategy that relied on the LabVIEW Prototyping Methods [2], has been introduced. This greatly helped in speeding up the user requirements clarification. On the other hand, in order to use LabVIEW GUIs together with the C/C++-based control applications there was the need of setting up a building environment compliant with the existing software engineering conventions and practices. Regarding the GUIs characteristics it was required them to contain as less as possible “intelligence” like, e.g., protections or control logic that was clearly identified as responsibility of the control processes. Other general operational requirements like, e.g., a maximum of two sub-levels to reach whatever functionality, had also to be considered.

ARCHITECTURE

Three basic components have been found to be necessary for the accomplishment of our requirements: a LabVIEW interface to our control framework, a common function library and a common building block library. Figure 1 shows these components, which we respectively called LVInterface, LVCommon and LVBlocks, and their dependencies. Altogether, they provide the necessary structure and implementation needed for building user interfaces on top of them.

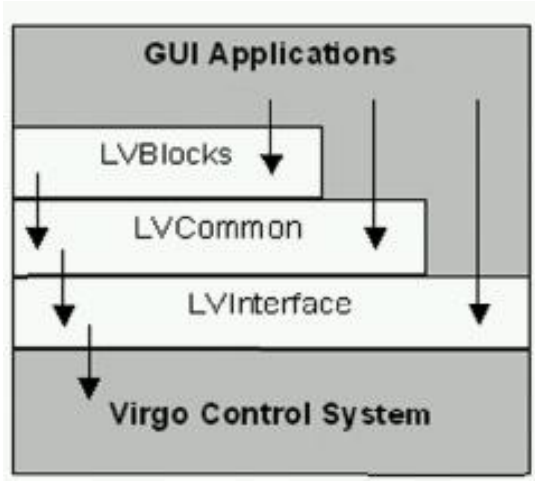

Figure 1: the three layer architecture

The Interface

The interface corresponds naturally to the lowest level and contains the implementation that allows communicating from LabVIEW applications to our control framework. In particular what was needed was a binding to the Virgo specific interprocess communication layer (Cm) [3] similarly to what has been done for other frameworks like EPICS [4] [5] or Tango [6]. For the latter reference cases the interfacing is based on dynamically loaded libraries (DLLs), however due to the specificities of the Cm communication layer it has been considered more appropriate to develop the interface by using a Code Interface Node (CIN) [7]. This CIN permits the sending of control commands to the distributed control system via the Cm communication service, and handles several command parameter data types such as integers, floats and strings. In this context the main effort has been spent in replicating the *cm send* command (able to build and send a message to a Cm application) within the specific CIN entry point routines:

- CINLoad/CINUnload: executed only once at VI load and unload time
- CINInit/CINDispose: executed once at load/unload time for each instance of the CIN block
- CINRun: Executed at each block activation time

The resulting binary code has then been incorporated in two VIs implementing command sending and receiving (CmSend, CmRecv) and one implementing the specific link to the Data Acquisition (DAQ) stream (CmGetData). Figure 2 is showing the block diagram corresponding to the CmSend VI.

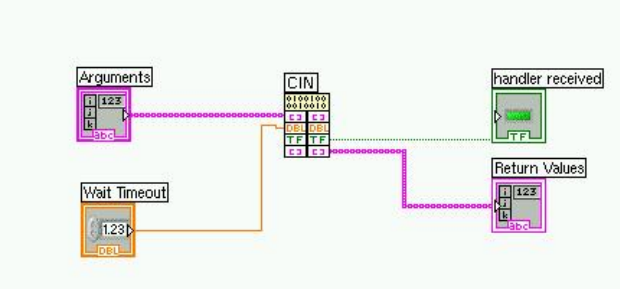

Figure 2: CmSend VI block diagram

Common Function Library

This library uses the described bindings and implements higher level common functionality like e.g. standard control process commands, utilities, etc. It is exclusively made up by LabVIEW VIs and does not contain graphical components. In order to make a clear distinction between common and custom VIs we used a color and prefix convention for the VI icons and filenames, respectively. In this context, we are currently evaluating if we can benefit from the project library approach introduced with LabVIEW 8. Our idea, even if not yet implemented, is to have the common VIs available on a custom function library for their immediate usage at the LabVIEW development environment.

Building Block Library

More than enhanced graphical widgets, when referring to building blocks we talk about completely autonomous, configurable sub-GUIs that can be inserted into higher-level GUIs as needed. Figure 3 shows an example GUI made up of several instances of the same building block.

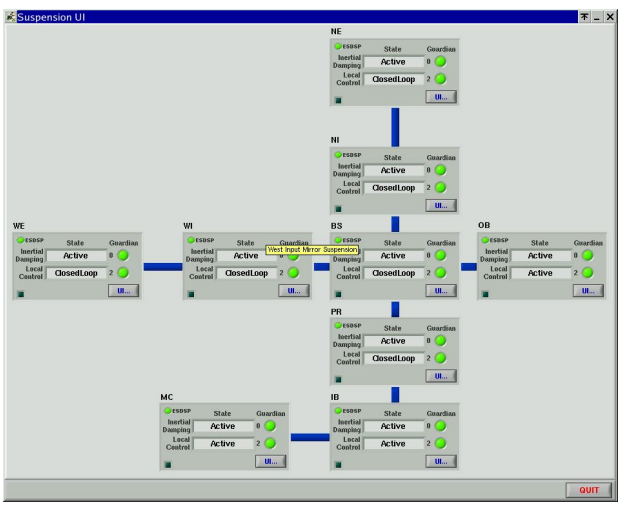

Figure 3: Suspension Towers UI, a GUI based on a single building block instantiated nine times

The methodology used is to place sub-frame containers in the GUI wherever a desired block should be located, and then to load and configure the respective sub-VI dynamically. The latter functionality is contained in a load-block-VI that receives the specific configuration as an input and hides the rest of the implementation to the user. In the SuspensionUI example one basic block has been created for a generic suspension tower and has then been reused several times in the higher-level GUI. As LabVIEW does not allow a straight-forward incorporation of complete blocks (i.e. programming and graphics together) those have to be loaded, configured and run dynamically inside a sub-panel container. Figure 4 shows the usage of a *load* sub-VI that receives whatever configuration is needed as input and returns the respective VI reference.

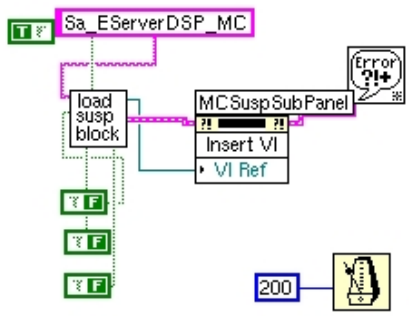

Figure 4: A load-block-VI use example

The described methodology unfortunately seems to find only limited support in the LabVIEW development environment. Consequently, a not negligible effort was spent in order to set up a useful and efficient implementation. However, the invested time has been largely rewarded by the fact that GUI development became fully modular, the code reuse could be maximized and the block diagrams became clear and maintainable. Even more, without this approach the development of the complex and wide spanning GUIs VIRGO is requiring would have quickly hit limitations regarding maintainability and flexibility.

SOURCE CONTROL AND PACKAGE MANAGEMENT

Three software packages have been created in order to contain the source code of the basic components: LVInterface, LVCommon and LVBlocks. As our source control system is based on packages more than files, and being the version number part of the path, we had to find a mechanism that allowed us to correctly align these basic packages whenever a new version was produced. Our aim was to integrate the build of those packages into our Makefile-based approach; however, the limited LabVIEW command line build options did not allow us to reach this goal. In addition, reviews of the LV8 Application Builder and the prior OpenG Builder have not been fruitful. Finally, we decided to perform manual builds and provide associated simple procedures.

TEMPLATES

As soon as the first high-level GUIs had been built by using the new methodology we generalized their structure

and then prepared templates that already contained implementations or structures for status updating, commanding, synchronization, constant definitions and sub-UI execution. This way also a common GUI look-and-feel could be ensured from the beginning. As the functional development had been reduced, we had more time available for the graphical enhancement of the GUIs, as can be seen at the example shown in Figure 5.

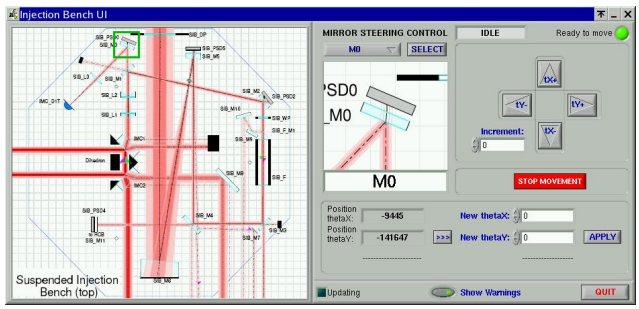

Figure 5: Mirror Steering UI, less time needed for the functional part allowed to concentrate on the graphical enhancement

CONCLUSION

The presented GUI building environment based on LabVIEW required an initial effort of customisation by establishing the right methodology and implementing the basic components, but it is now allowing to build new GUIs in a very short time with a high level of flexibility and maintainability.

Even with the limitations found at the level of LabVIEW command line build options and the not straightforward usage of autonomous building blocks, we are able to state that this LabVIEW based approach has proven to be efficient and permit us to fulfil the increasing amount of GUI-related user demands.

REFERENCES

[1] F. Carbognani et al., “Status of VIRGO”, these proceedings.
[2] LabVIEW Development Guidelines.
[3] C.Arnault, P. Massartal “A multitask communication package”, - LAL, Orsay, France.
[4] K.U. Kasimir. M. Pieck, L.R. Dalesio, “Integrating LabVIEW into a Distributed Computing Environment,” ICALEPCS 2001, San Jose, CA, USA.
[5] D. Thompson and W. Blokland, “A Shared Memory Interface between LabVIEW and EPICS,” ICALEPCS 2003, Gyeongju, Korea.
[6] J-M Chaize, A. Götz, W-D. Klotz, J. Meyer, M. Perez, E. Taurel and P. Verdier, The ESRF Tango Control System Status, ICALEPCS 2001, San Jose, CA, USA.
[7] LabView Documentation: Using External Code in LabVIEW.