Object-Oriented Technology in LabVIEW Programming

Kirill Rybaltchenko
JINR, DUBNA

Abstract

LabVIEW[1] is a powerful graphical package accessing any type of equipment of accelerator′s controls and offering well-designed Graphical User Interface (GUI). This paper consider the questions about LabVIEW programming in Object-Oriented manner and integration LabVIEW to the high-performance distributed Object-Oriented software system using modern technologies (CORBA).

1 Introduction

Now LabVIEW becomes more and more popular development environment to create a software controlling the different kinds of equipment in the control systems of accelerators. The control group of the SPS and LEP accelerators at CERN, Geneva, uses LabVIEW to control different accelerator subsystems, such as Beamloss monitoring system, Hydrostatic levelling system, Bunchlength measurement system, etc. LabVIEW uses a software package (SL-EQUIP) to access any equipment connected to any fieldbus (BITBUS, GPIB, RS-232, JBUS) remotely [2].

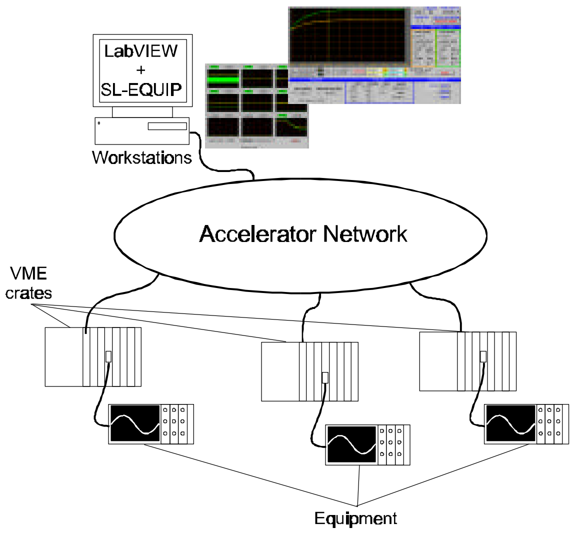


Fig. 1

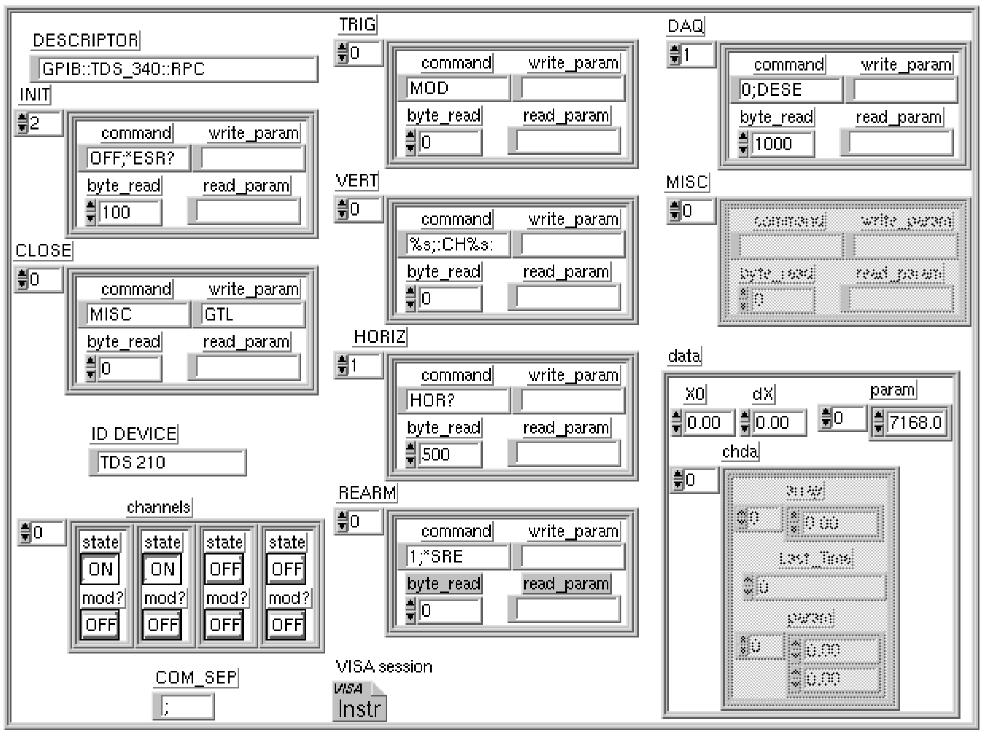
For example, in the Multi-Orbit Positioning (MOPOS) Timing Diagnostics system the LabVIEW program running on HP workstation controls 9 GPIB-oscilloscopes installed around SPS ring (fig.1). And sometimes the controlled system has a dynamically changeable configuration. For example, in the Time-Division Multiplexing Monitoring system the LabVIEW collects data from more than 70 points installed around accelerators and every point has different changeable configuration. In this example LabVIEW uses an ORACLE [3] database as a configuration database. In those examples there are strong requirements for the program flexibility: program should be able to connect to or disconnect from the system its different parts dynamically, keep current information associated with active parts of system. In that case the implementation of the OOP technology seemed to be very useful. Of course, we cannot say about OOP in a full meaning because of the LabVIEW graphical programming language ″G″ is not an Object-Oriented language. But some aspects of OOP are applicable and could give some profit.

Let′s consider a system of several devices, for instance oscilloscopes, like in the MOPOS Timing Diagnostics system. We can suppose even more complicated system: some of that scopes are Tektronix TDS-320, another are Tektronix TDS-210, some of them have RS-232 port instead of GPIB interface. And the configuration of the system may be changed: operator can include or exclude some devices from the data acquisition process in the run-time. In that case an implementation of OOP could help to avoid extra-complexity.

What is the typical way to write the OO program for device in C++? We create the class of devices describing entire family of devices, for instance scopes TDS-2xx and TDS-3xx. This class contains function-members to perform on that family of devices and all data-members containing the information about the current state of particular device. It may be the following functions: VerticalSetup, HorizontalSetup, GetData, etc. This class contains the Constructor and Destructor as well. Constructor, for instance, allocates memory for data associated with this device instance, opens connection to the device, performs device initialisation sequence, reads start-up values of device parameters. Destructor puts the device back to the local state, terminates the connection and release memory, allocated for this device instance. If we′d like to create a class for TDS-210 device, for instance, we create a subclass of this basic class and redefine the members unique for this particular device type.

How this methodology can be implemented in a LabVIEW programming? Normally all devices in family have the same set of functions. They differ from each other just in a format of command and response, number of controlling parameters. We create a global data table describing all common commands and parameters (fig.2).

This table is sharing between all device instances.



DESCRIPTOR: GPIB::TDS_340::RPC

INIT: 2

command	write_param
OFF;*ESR?	
byte_read	**read_param**
100	

CLOSE: 0

command	write_param
MISC	GTL
byte_read	**read_param**
0	

ID DEVICE: TDS 210

channels: 0

state	state	state	state
ON	ON	OFF	OFF
mod?	mod?	mod?	mod?
OFF	OFF	OFF	OFF

COM_SEP: ;

TRIG: 0

command	write_param
MOD	
byte_read	**read_param**
0	

VERT: 0

command	write_param
%s;:CH%s:	
byte_read	**read_param**
0	

HORIZ: 1

command	write_param
HOR?	
byte_read	**read_param**
500	

REARM: 0

command	write_param
1;*SRE	
byte_read	**read_param**
0	

VISA session: Instr

DAQ: 1

command	write_param
0;DESE	
byte_read	**read_param**
1000	

MISC: 0

command	write_param
byte_read	**read_param**
0	

data

X0	dX		param
0.00	0.00	0	7168.0

chda: 0

array: 0, 0.00

Last_Time: 0

param: 0, 0.00, 0.00

Fig. 2

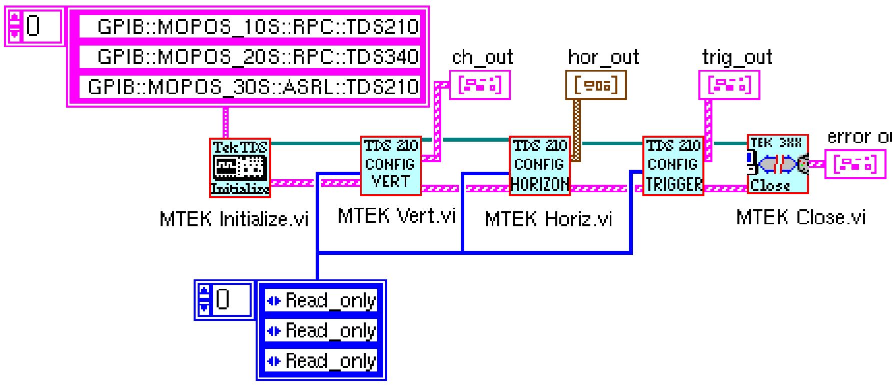


Fig. 3

When constructor creates an instance of device it creates a local data table containing all elements unique for this instance. All tables have the same fields. For instance, ′Horizontal′ field contains the format of command/response string and current Horizontal settings of the device. And the member-function HorizontalSetup sends takes from this field appropriate command, sends it to the device, extract all parameters from the device response using format response string and put all parameters to their places. So there is only one function HorizontalSetup for all devices. The block-diagram in fig.3 demonstrates how it can be implemented.

This small program opens connection to three devices of different type with different interface types. MTEK_Initialize.vi (Multiple Device Constructor) creates the instances of all devices and puts the array of references to these instances to all other functions. There is no need to take care about properties of every device functions pick up all what it need from data associated with that device and pointed by this reference to device instance.

Another important part of implementation OOP in LabVIEW is the communication between LabVIEW program and Front-End computers or between different Back-End applications (one or more of them are the LabVIEW applications) running on the different computers. If both applications are object-oriented, the old-fashioned communication software becomes bottleneck of such system. Now the CORBA [4] standard becomes more and more popular as a communication environment in the object-oriented software. An ILU [5] is one of the CORBA realisations which was tested for the Front End – Back End communication. The evaluation of ILU figured out that LabVIEW could be integrated easily in the distributed computing system using CORBA standard and the modern CORBA realisations like ILU could be implemented for the Front-End – Back-End communication.

Acknowledgements

I would like to thank my colleagues in SL division (CERN) and in LCTA JINR (Dubna) who helped me to test and to develop this software.

References

[1] LabVIEW – trademark of National Instrument company.

[2] Integrating Fieldbuses at the application level: C interface and LabVIEW integration – VITA Europe Congress 7-9 Oct. 1996, P.Charrue, K.Rybaltchenko.

[3] ORACLE – trademark of ORACLE corporation.

[4] CORBA – Common Object Request Broker Architecture standard developed by Object Management Group.

[5] ILU – Inter-Language Unification system – free software package developed by Rank Xerox company.