CS - A CONTROL SYSTEM FRAMEWORK FOR EXPERIMENTS (NOT ONLY) AT GSI

D. Beck*, H. Brand, GSI, DVEE, Planckstr. 1, D-64291 Darmstadt, Germany
F. Herfurth, CERN, CH-1211 Genève 23, Switzerland

Abstract

GSI performs basic and applied research in physics and related natural science disciplines using a heavy ion accelerator facility. The control system group of the department for Computing and Experiment Electronics (DVEE) helps the experiments in developing and implementing experiment control systems.

In many cases, control systems for small experiments with up to a few thousands process variables have been developed for specific experiments. The reusability of such a system is limited. Another possibility is the development of a general control system framework. Such a framework can be applied to a large variety of experiments by implementing a few experiment specific add-ons. During the past two years, the Control System (CS) framework [1], that is based on LabVIEW [2], has been developed at GSI. In autumn 2002, that framework has gone into production at the SHIPTRAP [3] experiment.

INTRODUCTION

So far, the main focus of the control system group of GSI/DVEE was on small systems of a few hundred up to a few thousand process variables. Only a limited number of persons are directly working at these small experiments. Typically, a few PhD students do the major part of the work and one student is responsible for the control system. Since students leave after a few years, it is not easy to keep the know-how within a research group. In many cases, the graphical programming language LabVIEW has become the default development environment since it has a fast learning curve and provides easy access to typical hardware equipment.

Around the year 2001 a new control system was required for a couple of projects. SHIPTRAP [3] is placed behind the velocity filter SHIP at GSI and makes heavy ions produced by SHIP available to experiments at low energy (< 1 eV). ISOLTRAP [4] at ISOLDE/CERN is a facility tailored for mass measurements of unstable nuclides far from stability. PHELIX [5] at GSI is a high energy laser with the objective to combine intense laser beams with high-current heavy-ion beams availabe at GSI. LEBIT [6] is a facility at NSCL/MSU that employes trap techniques to make rare species produced at a fragment separator available to low energy experiments.

In such a situation the reusability of code plays a major role in the design of a new control system. This gave rise to the idea of developing the control system framework CS based on LabVIEW. This development has two aims. First, the framework should provide the basic functionality that is common to the four experiments mentioned here. A dedicated control system can easily be set up by adding experiment specific add-ons to the framework. Second, having in mind the future extension of GSI, one must investigate how well such a LabVIEW based system can be scaled to more than 100,000 process variables.

REQUIREMENTS

The experiments require a highly flexible control system, where operational states can be configured on the fly. Another challenge is not so much the number of I/O channels but the number of different hardware device types. Even without programming experience, a PhD student must be able to maintain and develop the system further after a few months of learning time. A simple data acquisition must be included in the control system. SCADA functionalities like alarming, trending and user management are desirable.

The four experiments mentioned above have to do more than just setting some parameters and to acquire data. Sequences containing of up to 50 specific actions have to be repeated periodically with a rate exceeding 10Hz. Moreover, the different steps in such a sequence need to be synchronized with a precision of about 100ns, sometimes even in the sub-nanosecond range.

Important is the ability to call any function of any device at any moment. Then, observables can be measured as a function of any parameter of the experiment. This feature allows investigating systematic effects and debugging of an apparatus even for those parameters that are not used in the preconfigured operational modes.

The controlled devices are distributed and connected to different PCs. In some cases safety requires that the users must not be close to the experiment. This implies a distributed system with remote access.

SOLUTION

Today, the CS framework is based on LabVIEW only. The third party toolkit ObjectVIEW [7] eases object oriented programming with LabVIEW, but this toolkit is not required. Most hardware devices stem from third party manufacturers and can be connected via interfaces like RS232, RS485, GPIB and CAN-Bus. OPC connectivity gives access to, as an example, hardware connected via Profibus. The required precision on the synchronization is realized in configurable hardware like delay gate and pattern generators.

* Corresponding author: d.beck@gsi.de, +49 6159 71 2520

Object oriented approach and multi-threading

Although LabVIEW is not object oriented, one can use an object oriented approach within LabVIEW. Objects are represented by Virtual Instruments (VIs) and classes by so called VI-templates. During run time, multiple objects (VIs) can be created dynamically from a class (VI-template) by making use of the "VI server" functionality of LabVIEW. Inheritance is possible, but not as straight forward as in C++. The resulting class design has a limited number of inheritance levels and the classes have more functionality than in a real object oriented language. The base class of the CS framework is the CAEObj class. It provides active objects with the ability to communicate via events. If required for reasons of consistency, access to resources like attribute data can be locked by semaphores. The BaseProcess class is a child of the CAEObj class. It provides two threads, one for event handling and one for periodic action. Watchdog functionality for both threads is included. Optionally, a flat state machine is available in a third thread. This class also defines a protocol that is used for the event driven communication. Almost all classes in the framework are direct children of the BaseProcess class.

Event driven communication

In most cases, objects communicate via events. Direct method calls are rarely used. Every object may send an event to any other object. The receiver of an event and the method to be called is specified by their names, "ObjectName" and "EventName". Data are passed as byte arrays. The BaseProcess class provides three types of events.

- Simple: The sender sends an event to the receiver
- Synchronous: The sender sends an event to the receiver and waits for an answer of the receiver before its thread continues.
- Asynchronous: The sender sends an event to the receiver. The receiver sends its answer to a third object that has been specified by the sender.

Simple events may be buffered or unbuffered. The other event types are always buffered.

The BaseProcess class provides a method "GetDescriptors". By using this method one can query each object for its event descriptors. These contain the names of all events as well as the parameter and the parameter types required by the corresponding methods. Documentation about each method and its parameters is included in the event descriptors. By this, the use of an object via events is intuitive and almost self documenting.

SCADA functionality

SCADA functionality like trending, alarming and user management is provided by the Datalogging & Supervisory Control (DSC) module of LabVIEW. However, the functionality of the DSC module is encapsulated in the DSCIntProc class. This allows for a possible exchange of the DSC module by other software, if required in the future. The user management is not yet implemented.

Distribution based on TCP/IP

LabVIEW does not provide the functionality of sending events from one PC to another in the desired form. To make this possible, the classes QueueListener, QueueServer and QueueClient use their own protocol above TCP/IP. An object on a remote node is addressed in the form "ObjectName@NodeName".

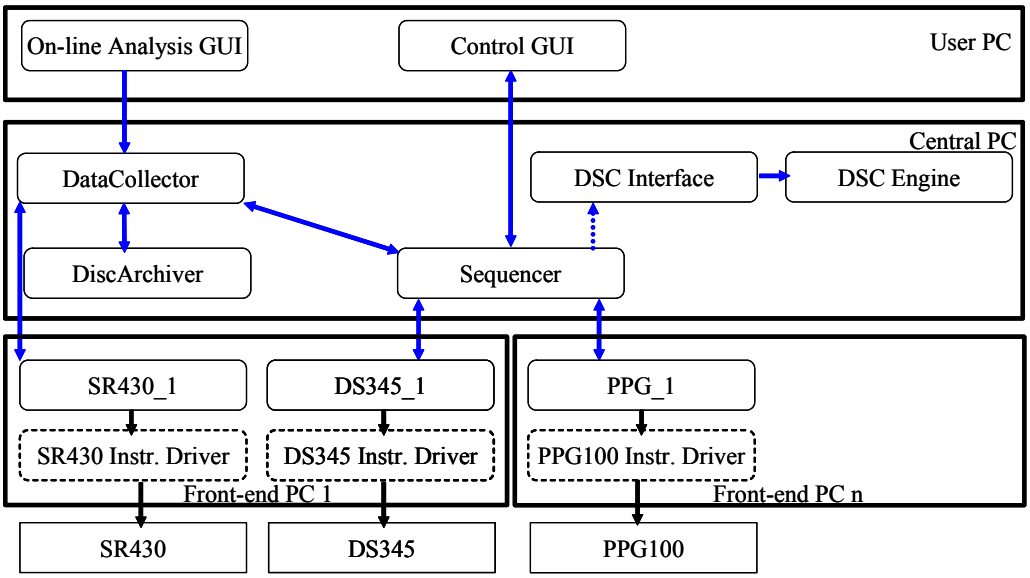

Figure 1: Simplified view of the SHIPTRAP control system. Solid bordered boxes with rectangular corners represent hardware like PCs or devices. Solid bordered boxes with round corners denote active objects of classes that are children of the BaseProcess class. Dashed bordered boxes represent libraries or drivers. Arrows indicate events or direct method calls.

THE SHIPTRAP CONTROL SYSTEM

As an example for a system based on the CS framework, Figure 1 shows the simplified design of the SHIPTRAP control system. The "Sequencer" and the GUIs have been derived from the "BaseProcess" class and are experiment specific add-ons. All other classes are part of the CS framework. The "ControlGUI" sends the parameters to the "Sequencer", which creates and parameterizes the objects required. Then, it starts the pattern generator "PPG100_1", which produces bit patterns. These are used to trigger devices and to synchronize actions with a resolution of 100ns. Then, the "Sequencer" calls the "DataCollector", which reads and buffers data from the data acquisition devices like "SR430_1". Clients are connected to the "DataCollector". One is the "DiscArchiver" that retrieves and writes the data to disk. Another client is the "On-line Analysis GUI" which analyzes and displays part of the data. All objects send their status and error information to the "DSCInterface" object for trending and alarming. This is indicated as a dashed arrow in the case of the "Sequencer". Today, the SHIPTRAP control system uses about 150 active objects that are distributed over four PCs.

SCALING TO LARGE SYSTEMS

Today, around 100 active objects are used on each PC and the number of PCs per control system is below ten. This is sufficient for control systems with up to a few thousand process variables. For scaling such a control system to 100,000 and more process variables, one either has to increase the number of active objects per PC or the number of PCs or both. So far, the following bottle necks for a LabVIEW based system have been identified.

A multi-threaded system must use so called "reentrant" VIs. As an example, a VI waiting for events is used in many threads at the same time. Unfortunately, LabVIEW needs to load the code of a reentrant VI as often into memory as the reentrant VI is used. As a result, lots of memory is consumed. Typically, PCs with 100 objects should have at least 512Mb of RAM.

Some applications requiring a high event rate observe occasional crashes of the control system after a couple of days. It can not be decided whether the crashes are due to bugs in the CS framework or due to bugs in LabVIEW itself. Due to the crash of the LabVIEW development system itself, debugging is difficult if not impossible.

Each device or channel is represented by one object. For starting the system, about 100 objects have to be created per PC. Creating one object typically takes a few seconds, so starting up one node would take a few minutes in the ideal case. Unfortunately, the time required to create a certain number of objects does not scale linear with the number of objects as illustrated in Figure 2. This is not a feature of our code, but results from the LabVIEW system.

CONCLUSION

After a development time of about three man-years, the CS framework is already in good shape. About 60 classes have been written that allow to access arbitrary function generators, delay gate generators, oscilloscopes, a multi channel scaler, step motor controllers, high voltage equipment and hardware like ADCs, DACs, DIOs on Profibus. The source code is GPL licensed and available for download [1].

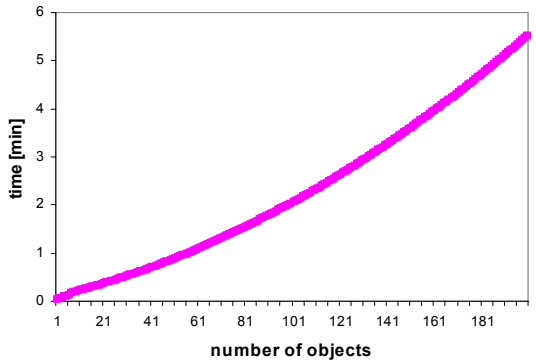

Figure 2: Measured time required for creating objects.

The framework is used at four experiments and being commissioned at a few other experiments. The number of process variables is up to a few thousand. So far, the feedback from the experiments is positive.

The first aim, supplying a couple of experiments with a new control system, is fulfilled. The second aim is to check the scalability of the approach presented here to large systems exceeding 100,000 process variables. At present, this can not be accomplished. Neither the number of objects per PC nor the number of PCs can be increased significantly. The limitations are due to the memory management and the performance of LabVIEW 7.0. National Instruments has been contacted.

So far, the development of the framework was driven by DVEE at GSI. The future of the development presented here depends on its acceptance by and the input from the experiments as well as on improvements expected in future versions of LabVIEW.

REFERENCES

[1] http://labview.gsi.de/CS/cs.htm.
[2] R. Jamal and H. Pichlik, LabVIEW Applications and Solutions, Prentice Hall, 1999.
[3] J. Dilling et al., Hyperfine Interactions 127 (2000) 491-496.
[4] G. Bollen et al., Nucl. Instr. Meth. A 368 (1996) 675.
[5] E.W. Gaul et al., GSI Scientific Report 2002 (2003) 101-103.
[6] S. Schwarz et al., Nucl. Instr. Meth. B 204 (2003) 507-511.
[7] R. Buhrke, LabVIEW Technical Resource, Vol.9, 3.