Status and Development of the *CS* Control System Framework

Dietrich Beck and Holger Brand
GSI-Darmstadt, DVEE, Planckstr. 1, D-64291 Darmstadt

Introduction

In the past years, the *CS* (Control System) framework has been developed at GSI [1]. The typical applications of *CS* have a couple of thousands process variables and require a large flexibility. The main emphasis of *CS* is not to control a huge number of process variables but to support a large variety of different hardware device types. SCADA features like alarming, trending and interfaces to field-busses like CAN and Profibus as well as OPC-servers are required. Easy maintenance paired with a fast learning curve is a major issue, since the main work is typically done by PhD students and Post-docs.
LabVIEW from National Instruments has been used for the implementation. Although LabVIEW is typically employed to table top experiments and test systems, it provides multi-threading, event-driven communication and a large number of hardware interfaces, including drivers for commercial devices. In order to improve the scaling to larger control systems, an object oriented approach was developed within the *CS* framework as well as the possibility to distribute control systems to a large number of nodes by implementing an event driven communication mechanism across the network. SCADA functionality is introduced by the DSC (Datalogging and Supervisory Control) module of LabVIEW.
CS is maintained by the ECoS group [2] at GSI. For building a dedicated control system, experiment specific add-ons like a sequencer may be implemented by the experimentalists. On the one hand, new requirements by experiments often lead to the development of new features within the *CS* framework and are implemented by the ECoS group. On the other hand, code written by a specific experiment, like a class for a device type, may become part of the *CS* framework. From this fruitful interaction between experiments and ECoS, *CS* based control systems have evolved at GSI (SHIPTRAP, CaveA, PHELIX), CERN (ISOLTRAP, REXTRAP) and MSU (LEBIT). A couple of facilities for the upcoming extension of GSI, like MATS and HITRAP, have already nominated *CS* as a promising candidate for new control systems.

Idea

The main idea of the *CS* framework is to couple an object oriented approach with an event driven communication mechanism. As an example, a class implements the functionality of a specific device type. Each real instrument of this device is then represented by an object of that class. Other classes may provide more abstract functionality like a sequencer or a GUI.
In most cases, objects do not communicate by calling directly the methods of other objects. Instead, they send events to the other objects. Events are typically buffered and contain information like the object name of the receiver, the name of the method to be called, a timeout value and data. The usage of events has two main advantages. First, one does not have to decide on the class type or the method to be called during the implementation. Since each object can in principle send an event to all other objects, the flexibility obtained is outstanding. One can even reconfigure a control system on the fly. Second, events can be transmitted across the network. As a result, it does not matter on which node the objects are created or where the hardware device is connected physically. In addition, the distribution of a control system over many nodes allows scaling a control system to a large number of process variables.

Status and recent developments

The base classes of *CS* provide the event mechanism, management of attribute data, simple state machines as well as a dedicated class to the SCADA system that may also connect to OPC. About 30 classes for different hardware device types allow a reasonable choice between a variety of instruments and manufacturers. A DIM [3] interface for LabVIEW has been implemented as well [4]. DIM provides transparent platform independent communication layer across the network. Originally, *CS* has been implemented on the MS-Windows platform. Meanwhile, it has been ported to Linux as well as to PharLap, a real-time OS used by the real-time variant LabVIEW RT. To complement the *CS* development, the tool *DomainConsole* has been implemented [4]. *DomainConsole* allows to monitor and to automatically re-start processes required by a control system. In addition, it allows monitoring CPU and memory usage on remote systems.

Performance

Today, the largest control system implemented with *CS* is the one for the PHELIX petawatt laser at GSI, which will have about 10,000 process variables in its final configuration. Per PC about 100-200 active objects or hardware devices can be used. The rate achieved for synchronous events[1] is about 1-2 kHz. As the hardware platform, a PC with PIII processor, 700MHz, and 1Gbyte of RAM is the recommended minimum. However, a stripped version of *CS* also runs on FieldPoint PACs from National Instruments with 16Mbyte of RAM. Under typical conditions, *CS* runs stable for at least a couple of hundred hours. In most cases, a shutdown of a distributed *CS* system is not due to instabilities of *CS* rather than typical maintenance like the deployment of a new version.

Summary and Outlook

The *CS* framework is used in production runs at various experiments. Results that were obtained using *CS* as control and data acquisition systems have been published. In general, the experimentalists are satisfied with the *CS* framework. *CS* is also envisaged as a candidate for control systems at FLAIR and HITRAP.
Technically, *CS* has concentrated on the device layer. For larger control systems, the application layer is becoming more important. As an example, future developments will deal with security/locking mechanisms as well as object nets.

Acknowledgements

We would like to thank the experimentalists, Klaus Blaum, Frank Herfurth, Sumit Saxena, Stefan Götte, Christian Rauth, Manas Mukherjee, Romain Savreux, Stefan Schwarz and Chabouh Yazidjian, who contributed to *CS*.

[1] D. Beck et al., Nucl. Instr. Meth. A 527 (2004) 567-579.
[2] http://www.gsi.de/controls.
[3] C. Gaspar and M. Dönszelmann, Proc. *IEEE Eight Conference REAL TIME '93 on Computer Applications in Nuclear, Particle and Plasma Physics*, Vancouver, Canada, 8.-11. June 1993.
[4] D. Beck et al., Proc. "Virtuelle Instrumente in der Praxis 2005", VIP 2005, Fürstenfeldbruck, Germany, Editors R. Jamal and H. Jaschinski, ISBN 3-7785-2947-1, 20-26.

[1] For a "synchronous event", the sender waits for an answer from the receiver prior to sending the next event.