THE *CS* FRAMEWORK – A LABVIEW BASED APPROACH TO SCADA SYSTEMS

D. Beck[1], H. Brand[1], S. Götte[1], F. Herfurth[1], C. Rauth[1], R. Savreux[2], S. Schwarz[3], and C. Yazidjian[1]
[1]*GSI, Planckstaße 1, 64291 Darmstadt, Germany*
[2]*CERN, Physics Department, 1211 Genève 23, Switzerland*
[3]*NSCL, Michigan State University, East Lansing, MI 48824-1321, USA*

ABSTRACT

Since a few years, the *CS* (Control System) framework is in use at a couple of experiments at various laboratories. The main idea behind *CS* is to provide a common basis which can be used to develop dedicated control systems by adding experiment specific add-ons. The key features of *CS* are an object oriented approach, event driven communication, the lack of intrinsic bottle necks like a central event manager, its ability to be distributed over many nodes, and the usage of LabVIEW which guarantees a fast learning curve as well as an excellent connectivity to hardware. The aim of this paper is to present the status of *CS* as well as to highlight recent achievements.

INTRODUCTION

In the past years, the *CS* framework has been developed at GSI [1]. The typical applications of *CS* have a couple of thousands process variables and require a large flexibility. However, the main emphasis of *CS* is not on the control of a huge number of process variables but to support a large variety of different hardware device types. SCADA (Supervisory Control And Data Acquisition) features like alarming and trending as well as 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 or Post-docs.

As a consequence, it was decided to develop the framework based on LabVIEW from National Instruments. Although LabVIEW is typically employed to table top experiments and test systems, it provides multi-threading, event-driven communication, a large amount of hardware interfaces, and many drivers for commercial devices. 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. Today, the *CS* framework is in use at about ten experiments at MSU, CERN and GSI. Although the original development has been done on the MS-Windows platform, it has been ported to Linux as well as to LabVIEW RT, a real-time implementation based on the PharLap real-time OS (Operating System). SCADA functionality is introduced by the DSC (Datalogging and Supervisory Control) module of LabVIEW. An interface to DIM [2], a communication system for mixed environments, has been implemented as well [3].

BASIC PROPERTIES OF *CS*

Implementation with LabVIEW

LabVIEW [4] is a graphical programming language. Its main advantages are its fast learning curve as well as its excellent connectivity to a lot of different types of industrial hardware. Multi-threading is an inherent feature of LabVIEW. Moreover, LabVIEW provides tools for synchronizing threads and for sending messages from one thread to another thread. SCADA features like alarming and trending are provided by the DSC (Datalogging and Supervisory Control) module of LabVIEW. In addition, the DSC module can serve as an OPC (OLE for Process Control) client and server. At GSI, the usage of LabVIEW is an explicit requirement by quite a few experiments.

Object Oriented Approach

In order to benefit from the advantages of object oriented techniques, those should be used together with LabVIEW. Unfortunately, this language has no built-in support for those techniques. Two third-party toolkits exist on the market that can be used, GOOP [5] and ObjectVIEW [6]. However, these toolkits are no open source software. Then, hunting for possible bugs may become a nightmare since one can not identify, if a bug is due to the toolkit or due to LabVIEW. However, the first version of *CS* was based on ObjectVIEW, but it turned out that this toolkit did not meet the performance requirements. As a consequence, a new object oriented approach was implemented within the *CS* framework [1]. This approach allows creating and destroying objects on the fly; tools for inheritance and a class browser have been implemented as well. As one of the primary requirements, *CS* is implemented on pure LabVIEW, and does not depend on external libraries linked to a specific OS. Implementation details are described in the appendix of [1].

Base classes form the core of *CS* and provide features like class specific attribute data, methods of accessing the attribute data and locking them to maintain their integrity. Inherited from the base classes are other classes that represent a specific hardware device type like, as an example, a function generator. Each physical function generator is then represented by an instance of that class.

Event Driven Communication

The main idea of the *CS* framework is to couple the object oriented approach with an event driven communication mechanism. In most cases, objects do not communicate by calling directly (*hard-wired*) 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.

Events are directly transmitted from one object to any other object. One the one hand, this implies some overhead for each object, since the validity of the event has to be checked by the object receiving the event. On the other hand, no intrinsic bottle neck exists, since a central event manager is not required. Within *CS*, events are based on the LabVIEW "message queues", which are buffered, and on the "notifications", which are not buffered. A timeout value can be set for each event. If the specified time has passed, the event will no longer be processed by the object receiving the event. This helps to avoid piling up of events and blocking a control system when a node is overloaded.

Configuration Data Base

If, as an example, an object represents a function generator that is connected via GPIB, the object needs information about the GPIB interface card representing the master of the GPIB bus as well as the GPIB address of the function generator on the bus. Such information is made available to the object via a configuration data base which is queried while an object is instantiated. Typically, a configuration data base is implemented using a commercial data base like MS-Access or Oracle and is queried via SQL (Structured Query Language).

Within the first *CS* versions, the configuration data base was configured as an ODBC (Open DataBase Connectivity) data source on each node of a control system. This approach has two disadvantages. First, an additional configuration of the ODBC data source was required for each node. Second, the LabVIEW SQL toolkit was required on each node.

The recommended alternative is depicted in Figure 1. A dedicated *SQL Server* accesses the configuration data base directly via SQL and ODBC. When an object is instantiated on any node of the control system, the configuration data are accessed via a *SQL Client*. The client then connects to the *SQL Server* and obtains the configuration data via TCP/IP. By this, the ODBC data source has to be configured only on one central node and the LabVIEW SQL toolkit is required only once per control system.

SCADA Backend

Per default, SCADA functionality like alarming and trending is provided by the DSC module of LabVIEW. This module is based on a Citadel real-time data base. Each object writes data to items, so called tags, of the DSC module. The tags and their properties like alarm limits have to be configured prior to starting the application. Tags can not be created on-the-fly. The configuration can be done via the so called *Tag Configuration Editor* that is part of the DSC module or via MS-Excel sheets, which can be imported. As a recommendation, the number of tags per DSC module should not exceed 10,000. Of course, one can use more than one DSC module per control system.

Per convention, typical *CS* classes do not write directly to tags of the DSC module. Instead, the SCADA functionality is encapsulated by a dedicated *DSCIntProc* class. Objects write and read tag values via an object of that class. For each object of the *CS* framework the name of the *DSCIntProc* object and its node are defined in the configuration data base. This encapsulation has two consequences. On the one hand, an intrinsic bottle neck is introduced when passing data between the *CS* system and the SCADA backend, if only one *DSCIntProc* object is used. On the other hand, this encapsulation allows replacing the DSC module of LabVIEW by another SCADA backend (see section *DIM Integration*).

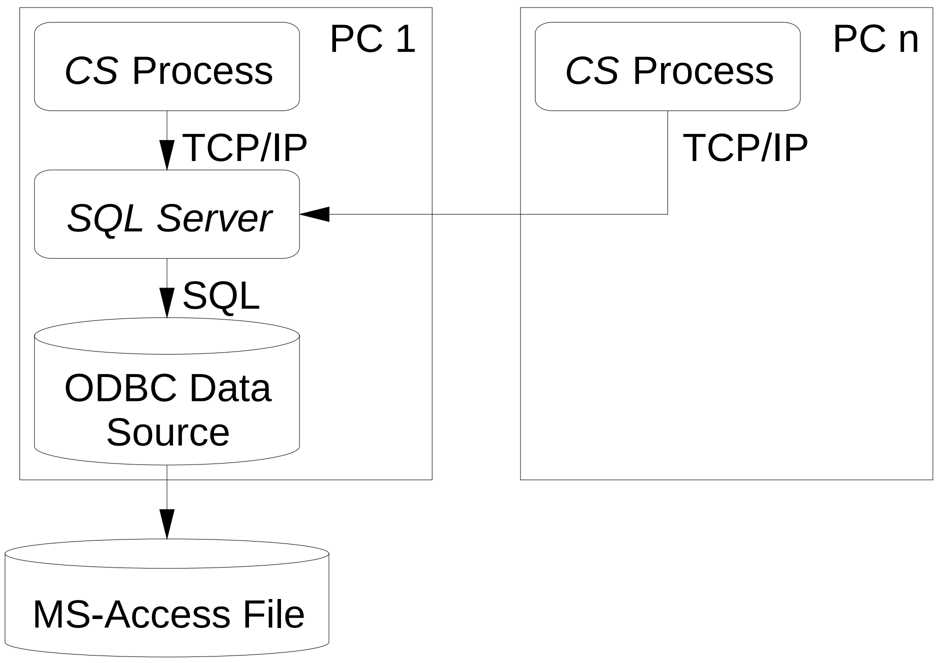

Figure 1: Recommended communication path for connection to the configuration data base. Here, all *CS* processes connect to a *SQL Server* via TCP/IP, which queries the data base via SQL.

OPERATING SYSTEMS

Since *CS* itself is based on pure LabVIEW, it can in principle run on all OSs that are supported by LabVIEW like MS-Windows, Mac OS X, Linux, SOLARIS as well as the real-time OS PharLap. In the following the OSs on which *CS* is actually used are discussed. Moreover, control systems based on *CS* can consist of nodes with different OSs and each node can use the OS which fits the task of the node best.

MS-Windows

Although LabVIEW originates from Mac OS, MS-Windows is the OS which is most commonly used with LabVIEW. As a consequence, this platform provides the best support by National Instruments and new features are implemented on this platform first. Moreover, most hardware manufacturers developing hardware for PC-based industrial automation have concentrated on MS-Windows. Hence, the support for hardware drivers for this OS is unbeaten. Another example is industrial communication via OPC, which is based on the DCOM standard by Microsoft. Huge amounts of hardware come with OPC servers and thus require MS-Windows. As a consequence, MS-Windows is the OS on which *CS* has been developed and which is used most frequently and tested best.

Linux

Linux plays a major role in the research community. Its main benefits are the usability by more than one user per node and in many cases a better performance and stability compared to MS-Windows. For various reasons, some experiments require the usage of Linux for control systems as well. The core classes of *CS* can be used on Linux without changes. However, the support for commercial hardware drivers is missing in many cases. As an example, the new DAQmx driver from National Instruments is not available for Linux until now. This imposes a significant limitation on the usability of many hardware device types. However, the possibility of a network installation as well as its good stability makes *CS* installations on Linux more and more common.

Real-Time

Some control and data acquisition tasks require high reliability and determinism. Typical examples are systems for protection of machines and personnel or control systems that need to synchronize with the timing structure of accelerators. This is the domain of real-time systems. LabVIEW RT, the real-time variant of LabVIEW, is based on the real-time OS PharLap or RTX, a Windows Realtime Extension. It adheres to preemptive and round-robin scheduling, optimized for deterministic performance. Threads with higher priority always preempt execution of lower priority threads. When threads of equal priority need to execute, round-robin scheduling gives each thread an equal amount of time with the processor. Depending on the hardware, deterministic performance with minimal jitter in the order of microseconds can be achieved for one dedicated time critical thread with highest priority. Other threads with lower priority process non-time critical tasks like publishing data or receiving commands via TCP/IP. The *CS* framework is successfully used with LabVIEW RT [7].

DIM INTEGRATION

DIM (Distributed Information Management) [2] is a communication interface for distributed systems and has been developed at CERN. DIM is based on a client-server architecture. A server publishes so-called services, which can be of an elementary data type, an array of arbitrary size or a user defined data type. A DIM server can receive and process commands. Of course, DIM is fully event driven. DIM is implemented for the operating systems Windows, Linux, VMS, Unix as well as the real-time platforms OS9, LynxOs and VxWorks. Supported languages are C, C++, Java und Fortran. Its high performance as well as its capability to interconnect a wide range of different platforms makes it an ideal tool to connect LabVIEW to non-LabVIEW applications. A LabVIEW interface has been developed in a different work [3].

The integration of DIM into *CS* is done via a dedicated *DIMIntProc* class. This class implements a DIM server. The usage of this class by other classes is identical to the *DSCIntProc* class described in section *SCADA Backend*. As an additional functionality, the *DIMIntProc* class may receive commands via DIM and send the command together with the received data to any object as an event. By replacing a *DSCIntProc* object by a *DIMIntProc* object, two scenarios are opened up. First, instead of using a real SCADA backend, one can just publish the values of tags to DIM and make them available to all DIM clients on the same network. In that case, DIM also serves as a kind of real-time database for the *CS* based control system. Second, the DIM can now serve as a communication interface to SCADA back ends different from the DSC module. As an example, one can use the SCADA system PVSSII by ETM. A DIM interface for PVSSII has been implemented within the JCOP framework [8].

DEPLOYMENT AND OPERATION

During real operation, *CS* systems should not be used in the LabVIEW development environment. Instead, executables should be built using the LabVIEW *Application Builder*. First, executables are significantly faster und consume less memory. Second, executables can only be replaced but not changed. If a new version of the software has a problem, changing back to old binaries can be done within a few seconds. To make sure all nodes use the same software version and to ease the deployment of new software, binaries are preferably installed on a network drive.

For operation, two additional requirements must be taken into account. First, all required processes on each PC must be started automatically after a PC is booted. Examples are the binary for the *CS* process or OPC servers. The required processes may differ from PC to PC. Second, there is always a possibility that a process crashes unexpectedly. In such a case, the process must be restarted immediately without user interaction.

To take these requirements into account, an additional process, *DomainConsole*, may be started on each node. When started, *DomainConsole* reads an ini-file which has for each node sections containing a list of processes. *DomainConsole* starts all processes that are listed in that section. By this mechanism, *DomainConsole* is able to start all processes specific for each node. Each process listed in the ini-file has an additional parameter, which determines how this process is started by *DomainConsole*: A process can be disabled, it can be started once, or it can be started whenever it is not executing. In the latter case, *DomainConsole* periodically checks, if that process is contained in the list of processes maintained by the operating system. If a process is not listed, it has either not been started or it has crashed. Then, the process is restarted by *DomainConsole*.

PERFORMANCE

Today, the largest control system implemented with *CS* is the one for the PHELIX (Petewatt High Energy Laser for Ion eXperiments) facility at GSI, which will have about 10,000 process variables in its final configuration [9]. About 100-200 active objects or hardware devices can be used per PC. The rate achieved for synchronous events[*] 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.

MAINTENANCE AND DISTRIBUTION

The core of the *CS* framework and all classes of general interest are maintained by the KS (KontrollSysteme) group of the EE (Experiment-Elektronik) department at GSI. Application specific classes like a dedicated GUI of an experiment are maintained by the experimentalists. The software is available under the terms of the GPL (GNU General Public License). Up to date information and downloads are available via a dedicated web-site [10].

APPLICATIONS

The development of *CS* was mainly triggered by facilities using ion traps to investigate unstable nuclei. *CS* is successfully used since about two years at the following places: SHIPTRAP is a facility at GSI dedicated for investigating super-heavy transuranium elements which are created by fusion-evaporation reactions [11]. ISOLTRAP at ISOLDE/CERN determines masses of unstable nuclei for nuclear physics and fundamental tests [1]. REXTRAP at ISOLDE/CERN serves for cooling and bunching of ISOLDE beams [12]. LEBIT at MSU performs mass measurements of rare isotopes [13]. An atomic physics experiment at GSI conducts measurements of atomic life times in the pico-second range [14]. PHELIX is a Petawatt laser being set up at GSI [9].

Two new *CS* based control systems are being developed for two experiment at GSI, FOPI [15] and RISING. [16]. A couple of upcoming facilities at GSI, like MATS [17] and HITRAP [18], have already nominated *CS* as a promising candidate for new control systems.

CONCLUSION AND OUTLOOK

The *CS* framework is used in production runs at various experiments. In general, the key requirements of the experimentalists are fulfilled. Results that were obtained using *CS* as control and data acquisition systems have been published. New *CS* installations are being set up at GSI and *CS* is also envisaged as a candidate for control systems at FAIR (Facility for Antiproton and Ion Research) in Darmstadt.

CS can cooperate with other control system software by using DIM or OPC as protocols. The usage is no longer restricted to the MS-Windows platform. Today, *CS* can also be used with Linux or, by using LabVIEW RT, for applications requiring hard real-time. Two tools, *SQLServer* and

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

DomainConsole, have been developed which assist in setting up a *CS* based control system that is distributed on many nodes.

So far, the emphasis of *CS* was to give access to a large number of different hardware types and to provide high flexibility for the experimentalists. The number of process variables itself was a minor issue. So far, up to 10,000 process variables are used in distributed installations with at most ten PCs. In the future, the scaling of *CS* to a larger number of process variables will be investigated and the application layer is becoming more important. As an example, future developments will deal with security/locking mechanisms as well as an object broker and object nets.

REFERENCES

[1] D. Beck et al., "A new control system for ISOLTRAP", Nucl. Instrum. Methods A 527 (2004) 567-579.
[2] C. Gaspar and M. Dönszelmann, "DIM - A Distributed Information Management System for the Delphi experiment at CERN", Proc. IEEE Eight Conference REAL TIME '93 on Computer Applications in Nuclear, Particle and Plasma Physics, Vancouver, Kanada.
[3] D. Beck et al., Proc. "Virtuelle Instrumente in der Praxis 2005", "Die LabVIEW-DIM Schnittstelle: Das Tor zur standardisierten Kommunikation zwischen LabVIEW und einer Vielfalt von Programmiersprachen und Betriebssystemen", VIP 2005, Fürstenfeldbruck, Germany, Editors R. Jamal and H. Jaschinski, ISBN 3-7785-2947-1, 20-26.
[4] R. Jamal and H.Pichlik, "LabVIEW Applications and Solutions" (1999) Prentice Hall. See also: http://www.ni.com/.
[5] David Hoadley, "What is GOOP?", LabVIEW Technical Resource, Vol.11, 4 (2003).
[6] R. Buhrke, "G++ with ObjectVIEW - A new concept of advanced object-oriented LabVIEW programming", LabVIEW Technical Resource, Vol. 9, 3 (2002).
[7] D. Beck et al., "The First Approach to Object Oriented Programming for LabVIEW Real-Time Targets", IEEE Trans. of Nucl. Science, submitted.
[8] For detailed information on the usage of PVSSII in large physics experiments see http://itcobe.web.cern.ch/itcobe/Projects/Framework/.
[9] S. Borneis et al., "Status of PHELIX", GSI Scientific Report 2004 (2005), 222-223.
[10] http://www-w2k.gsi.de/controls/CS/cs.htm.
[11] G. Sikler et al., "First on-line test of SHIPTRAP", Nucl. Instrum. Methods, B 204 (2003) 482-486.
[12] D. Habs et al., "The REX-ISOLDE project", Hyperfine Interactions 129 (2000) 43–66.
[13] S. Schwarz et al., "The low-energy-beam and ion-trap facility at NSCL/MSU", Nucl. Instr. Meth.B204 (2003) 507-511.
[14] S. Toleikis et al., " Lifetime of the 2 $^3P_0$ state of He-like ^{197}Au", Phys. Rev. A 69, (2004) 022507-022511.
[15] W. Reisdorf et al., "Central Collisions of Au on Au at 150, 250 and 400 A MeV", Nucl.Phys. A612 (1997) 493-556.
[16] F. Becker et al., "Status of the RISING project at GSI", Eur. Phys. J. A (2005) 719–722.
[17] K. Blaum et al., "Technical Proposal for the Design, Construction, Commissioning and Operation of MATS", Mainz, 2005.
[18] W. Quint et al., "HITRAP: A Facility for Experiments with Trapped Highly Charged Ions", Hyperfine Interactions 132 (2001) 453-457.