The First Approach to Object Oriented Programming for LabVIEW Real-Time Targets

Dietrich Beck, Holger Brand, Christos Karagiannis and Christian Rauth

***Abstract*—For the first time, an object oriented approach is used together with LabVIEW RT. This real-time variant of the well known graphical programming language is based on the PharLap OS. This allows for two different scenarios. First, existing object oriented LabVIEW code developed for MS-Windows and Linux platforms can be reused on a real-time platform, next to a time critical loop with high priority. The reusability of existing code drastically reduces the development time for complex data acquisition and control systems, based on PXI or cPCI platforms. Second, small PAC systems like the FieldPoint series from National Instruments have limited CPU power and memory. However, dedicated classes allow the usage of an object oriented approach and serve to integrate such sub-systems to larger SCADA systems.**

***Index Terms*—Object oriented methods, Real time systems, Safety, Vacuum control, LabVIEW**

I. INTRODUCTION

DURING the last years, the object oriented, event-driven, multi-threaded and distributed *CS* (Control System) framework has been developed at GSI [1]. Today, its typical applications are small experiments with up to 10,000 process variables. Typically, such control systems are maintained and developed further by PhD students. Therefore, the *CS* framework has been developed using the graphical programming language LabVIEW [2].

However, some control and data acquisition tasks require high reliability and determinism. Typical examples are systems for protection of machines and personnel, control systems that need synchronizing with the timing structure of accelerators, or data acquisition systems that are closely linked to the front-end hardware [3], [4]. When high performance is required, real-time systems are important, as an example, to implement time critical closed loops for fuzzy or PID regulators. In some cases, determinism in the microsecond range can be achieved. Other examples are systems that must react within a guaranteed response time to interrupts triggered by external signals. Depending on the architecture and the implementation, reaction times between tens of microseconds and milliseconds are achieved. Today, a large range of real-time targets exists. On the one hand, embedded systems are used for dedicated tasks. So called PLCs (Programmable Logic Controllers) are applied for autonomous safety systems requiring high reliability. On the other hand, powerful architectures with a large amount of CPU power and memory, like ordinary PCs or modern VME-processors can be used with real-time OSs (Operating Systems) like LynxOS or VxWorks.

Programming ordinary PLCs like the S7 from Siemens or intelligent bus terminal controllers from Beckhoff requires specialist know-how. However, many experiments do not have permanent staff but are run, maintained and developed further by PhD students or Post-docs. Here, conventional PLC programming is too time-consuming for two reasons. First, PLC programs only need to be changed rarely and no knowledge is built up amongst the students. Second, new students are often required to learn PLC programming from scratch. As a consequence, quite a few experiments explicitly require the usage of LabVIEW. Here, the alternative to PLC programming is the usage of a PAC (Programmable Automation Controller). PACs offer the flexibility of a PC and the reliability of a PLC. They can be programmed with LabVIEW and allow for setting up critical sub-systems, where a high reliability is required.

This work reports on the application of the *CS* framework to real-time targets based on LabVIEW RT (LabVIEW Real-Time), a real-time variant of LabVIEW which is based on the real-time OS PharLap. This approach drastically reduces the development time and eases maintenance of code written by personnel with short term obligations to control systems.

The outline of the paper is the following. It gives a brief review of the *CS* framework, describes the implication of using *CS* on LabVIEW RT targets, presents an application example and finally reports on performance benchmarks.

II. THE *CS*-FRAMEWORK

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 (Supervisory Control And Data Acquisition) features like alarming, trending and interfaces to field-buses like CAN and Profibus as well as OPC-servers are required. Easy maintenance paired with a fast learning curve is a major issue, requiring only *one* development tool.

Manuscript received June 5, 2005; revised March 3, 2006.

Dietrich Beck (corresponding author; phone: +49 6159 71 2520; fax: +49 6159 71 2986; e-mail: d.beck@gsi.de), Holger Brand (e-mail: h.brand@gsi.de), Christos Karagiannis (e-mail: c.karagiannis@gsi.de) and Christian Rauth (e-mail: c.rauth@gsi.de) are with GSI-Darmstadt, Planckstr. 1, 64291 Darmstadt, Germany.

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 an example, a class implements the functionality of a specific device type. Each real instrument of this device type is then represented by an object of that class. Other classes may provide more abstract functionality like a sequencer or a GUI (Graphical User Interface). SCADA functionality is introduced by the DSC (Datalogging and Supervisory Control) module of LabVIEW.

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 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.

CS is maintained by the ECoS group [5] 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 now nominated *CS* as a promising candidate for new control systems. For more details on the *CS* framework see [1] and [6].

III. Applying *CS* to LabVIEW RT

A. LabVIEW RT

LabVIEW RT 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. Two possibilities exist for porting the *CS* framework to LabVIEW RT depending on the hardware platform in question.

First, embedded PXI (PCI eXtensions for Instrumentation) controllers provide the performance of modern desktop PCs with respect to CPU speed and available memory[1]. In this case the *CS* framework can basically be used without changes. This allows re-using existing code which is especially interesting for classes dedicated to instruments, communication as well as supervision. To make use of the real-time capabilities, threads for typical objects are scheduled with low priority. Only dedicated objects responsible for e.g. synchronous I/O to parts of an accelerator are scheduled to run at high priority.

Second, the FieldPoint PACs from National Instruments provide embedded controllers for reliable, stand-alone operation that can be used to implement rugged systems in a front-end environment. Typically, they are equipped with x86 processors of about 200MHz and 24 MBytes of user accessible RAM. Different analog and discrete I/O modules can be attached to the controller and connect to the outside world. However, the limited CPU capability does not allow reusing the *CS* framework with the same classes as for the other platforms. This is not a restriction since the driver for the I/O modules is completely different to the drivers used on the more powerful architectures; new classes for the FieldPoint hardware have to be implemented in any case. In the following, the *CS* framework is only discussed for the FieldPoint platform.

B. Class Hierarchy

Fig. 1 shows the simplified class hierarchy. Common to all platforms are two classes. The *CSObj* class is *the* base class providing basic functionality like management of attribute data or closing and opening of front panels. Each sub-class defines additional attribute data. The mutual exclusion of attribute data is realized by one semaphore, which is provided by the *CSObj* class. The *CAEObj* class allows creating active objects and gives access to a basic event mechanism. All other classes are specific for real-time targets. The *RTBase* class gives access to a configuration data base and to the distributed event mechanism implemented within the *CS* framework. The *RTDigiOut* class serves as an example for instrument specific classes and represents digital output modules. Typically, an application specific *Sequencer* class contains the time critical code. It implements the required functionality by using the

[1] Since 2005, LabVIEW RT is also supported on desktop PCs.

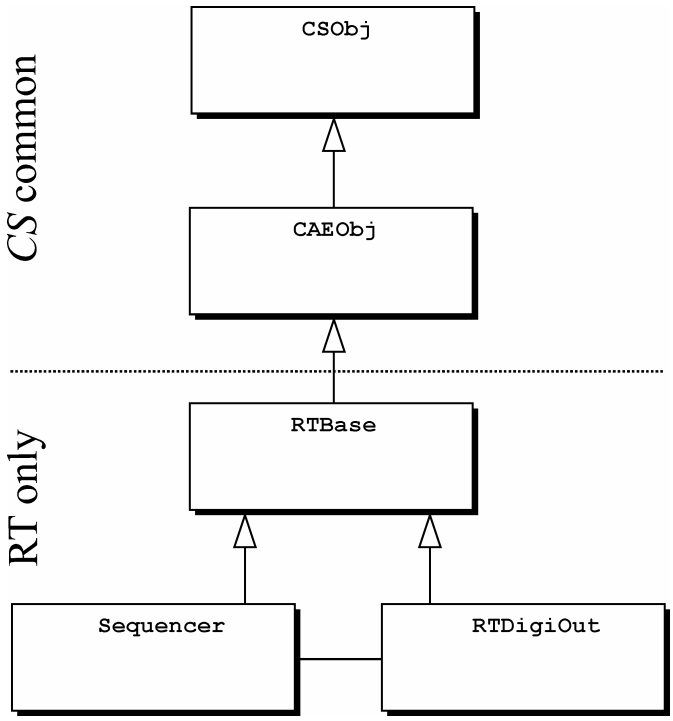

Fig. 1. Simplified class hierarchy for the FieldPoint real-time target. Boxes denote classes and arrows point from a derived class to its base class. All classes are part of the *CS* framework except the *Sequencer*, which is application specific. Only the *Sequencer* class is scheduled to execute at high, time critical priority. All classes above (below) the dotted horizontal line are platform independent (limited to the RT platform).

methods of other classes.

C. Implementation

The *CSObj* and *CAEObj* classes are used on LabVIEW RT without any changes. They are common to the MS-Windows, Linux and LabVIEW RT platform. The *RTBase* class is the first class specific for the LabVIEW RT platform and corresponds to a class *BaseProcess* which is available on the Linux and MS-Windows platforms [1]. In order to reduce memory consumption, *RTBase* has a limited functionality compared to the *BaseProcess* class. As an example, the *BaseProcess* class has two threads, one for event handling and one for periodic actions. For the *RTBase* class, the thread for periodic action has been omitted. Other changes take into account requirements that are specific for front-end systems requiring high availability. An example for such a requirement is the capability of re-booting and starting up after a power-cut independent of the Ethernet. On non real-time systems, the *BaseProcess* obtains the configuration data for each object from a relational data base via the Structured Query Language (SQL) from a central server. However, the *RTBase* class can not do so, since accessing a central server would require the server and the Ethernet to be operational. Hence, the *RTBase* class uses a local configuration data base which is implemented with ini-files[2].

The *Sequencer* has at least two threads. The first thread implements a time critical loop in straightforward LabVIEW. For deterministic performance, this thread does not use event driven communication but direct method calls to other classes. The second thread has low priority. It serves as an interface by publishing data and receiving commands.

The *CS* framework provides more classes of general interest. Most importantly are device classes for hardware I/O like the *RTDigiOut* class, which implements a logical abstraction to digital output lines. Not shown in Fig. 1 are other classes that provide a TCP/IP interface which is required to distribute events across the network. This is necessary to integrate the real-time system into distributed applications based on the *CS* framework.

IV. Vacuum Safety System for SHIPTRAP

The new approach to object oriented real-time programming presented in this work was triggered by the need for a new vacuum safety system for SHIPTRAP [7]. This solution is a successor of a PLC based system. The reason for the replacement lies in the fact that acquiring the ability of PLC programming takes too long for students. SHIPTRAP explicitly required the new solution not to be based on a PLC. The main requirements for the new vacuum safety system are the same as for the PLC system [8]. Turbo molecular pumps and roughing pumps serve to evacuate the apparatus and beam-lines. Some sections of the apparatus need to be at ultra high vacuum better than 1×10^{-8} mbar, while three dedicated sections are filled with helium buffer gas between 1×10^{-5} mbar and 0.1 bar. Shutter valves can be used to separate the different sections between each other and from the pumps. The vacuum safety system has three main requirements. First, it protects the vacuum sections in case of, as an example, a failure of a pump by closing adjacent shutter valves. Second, it analyzes the state of the vacuum system and automatically re-starts pumping the apparatus after a power cut. Third, it receives commands from a GUI on a remote node and performs actions required by the user like opening or closing valves. Fig. 2 shows the layout of the new vacuum safety system and is described in the following.

A. Hardware

The hardware components are all commercially available and depicted by rectangular boxes with solid borders. The controlled hardware for the vacuum system itself consists of pumps, gauges and valves. The control system hardware consists of a FieldPoint controller FP2010 and various I/O modules from National Instruments. The user interface and SCADA functionality is implemented on PCs. Only one PC is required, but the software allows distributing the tasks over various PCs as shown in Fig. 2.

B. Software

Fig. 2 depicts software components by boxes with round corners. The *Web Server* as well as the *LV DSC* (LabVIEW DSC module) is commercial. All other software has been developed at GSI. Only two pieces of software, the *Sequencer*

[2] For deployment, LabVIEW RT allows for downloading such ini-files as well as binaries via a built-in FTP-server

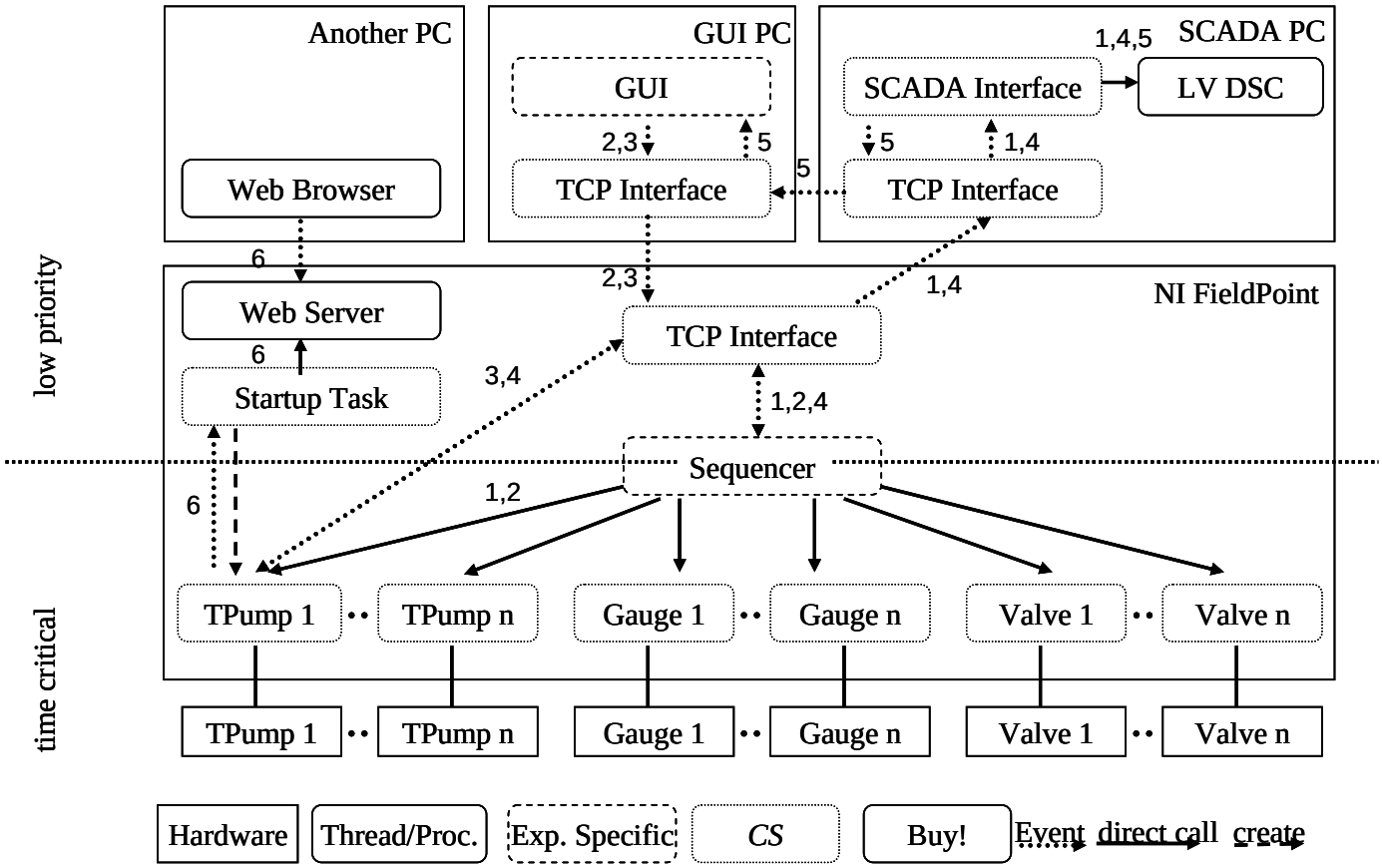

Fig. 2. Layout of the SHIPTRAP vacuum safety system. Hardware (software) is depicted by boxes with rectangular (round) corners. Dashed (dotted/solid) borders of boxes denote experiment specific (*CS* common/commercial) components. Direct method calls are marked by solid arrows. Event driven communication is marked by dotted arrows. Creating an object is marked by a dashed arrow. The direction of the arrows does not indicate the direction of data flow but the relationship between caller and receiver. Numbers denote communication paths for courses of action. For more information see text.

and the *GUI* are specific for this project. All other software is commonly used by other experiments as well and is provided by the *CS* framework. The *Sequencer* is an object of a class providing two threads; see sect. III.B. The border between time critical and low priority tasks is indicated by a dotted horizontal line.

C. Starting the real-time application

Since the binaries for the application are stored locally on the FieldPoint controller together with the configuration data base, the time critical part of the application is independent of the network. The first thread to be started is the *Startup Task*. First, it reads its ini-file that contains the names of all objects that have to be created. For each individual object, the class name and the name of an object specific ini-file is given as well. Second, all objects are created (*TPump1..ValveN*, *Sequencer*, *TCP Interface)*. Each object reads its ini-file while executing the constructor of its base class *RTBase*; see sect. III.B. Third, the *Sequencer* starts executing its time critical thread, after all objects have been created.

D. Communication paths

The *CS* framework provides two ways for communication between objects, event driven communication and direct method calls. Fig. 2 contains numbers indicating the communication paths for certain types of actions. Please note that not all communication paths are shown.

1) Sequencer

In its time critical thread, the sequencer uses direct method calls to the objects corresponding to the vacuum system hardware. It closes or opens valves and queries the status information. The status information is passed as an event via two *TCP Interfaces* to the *SCADA Interface*. The latter object encapsulates the SCADA functionality. Here, the *LabVIEW DSC* module serves as the SCADA backend. However, one could also use an interface to DIM [9] and PVSSII as a SCADA backend [10].

2) User requests

Via the *GUI* a user may request actions of the vacuum safety system. Such a request is sent as an event via two *TCP Interfaces* to the *Sequencer*, which checks the validity of the request. If the request is valid, the *Sequencer* performs the requested action like switching on *TPump1*.

3) Asynchronous user requests

Via the *GUI* the user may directly request actions from the device objects like *TPump1*. These requests are sent as events and processed by a low priority thread of the device object. Depending on the configuration in the corresponding ini-file, such actions can be allowed or denied. As an example, directly reading the status of *TPump1* may be allowed. But a request to open *Valve1*, bypassing the *Sequencer*, is denied by

the *Valve1* object.

4) Sequencer status and error information

Depending on the read-back values of the device objects, the *Sequencer* object will calculate the status of the vacuum system or sub-systems. The status information is sent as events to the SCADA system via the *TCP Interfaces* and the *SCADA Interface*. In some cases, device objects like *TPump1* may publish their status information directly.

5) Display of status and error information

In order to reduce the load on the FieldPoint system, the *GUI* does not query the status information directly from the device objects or the *Sequencer*. Instead, it obtains the data from the SCADA backend. For improved performance and better scalability, the *SCADA Interface* is not polled by the *GUI*, but the following mechanism is used.

The *SCADA Interface* serves as a data server for on-line trending of values which are acquired by the SCADA backend. Clients may subscribe to the *SCADA Interface* to receive values. Then, the *SCADA Interface* publishes the values of data points by sending events to the clients. This happens once after the subscription and each time the value of the data point changes. In Fig. 2, the *GUI* represents a client that subscribes to the *SCADA Interface*. If *GUI* and *SCADA Interface* are situated on different nodes, the events containing the data are sent via *TCP Interfaces*.

6) Web server

The FieldPoint controllers are equipped with an on-board *Web Server*. In this application, the web server is used as an additional communication path that is mainly used for debugging purposes. It mainly displays status information which is made available via the *Startup Task*. As an example, the names of the created objects are displayed together with the names of the ini-files used. All objects of the real-time system may publish text information via the web server by sending an event, containing a status text or an error message, to the *Startup Up* task.

E. Remote access and security

Remote access to the services of the FieldPoint system itself can be allowed or denied by configuration tools provided by National Instruments. Examples for such services are the *Web Server* or an FTP-Server. The latter is not shown in Fig. 2 and allows downloading ini-files or binaries to the FieldPoint controller.

The *TCP Interface* objects provided by the *CS* framework do deny all remote access from other *CS* nodes. Only nodes

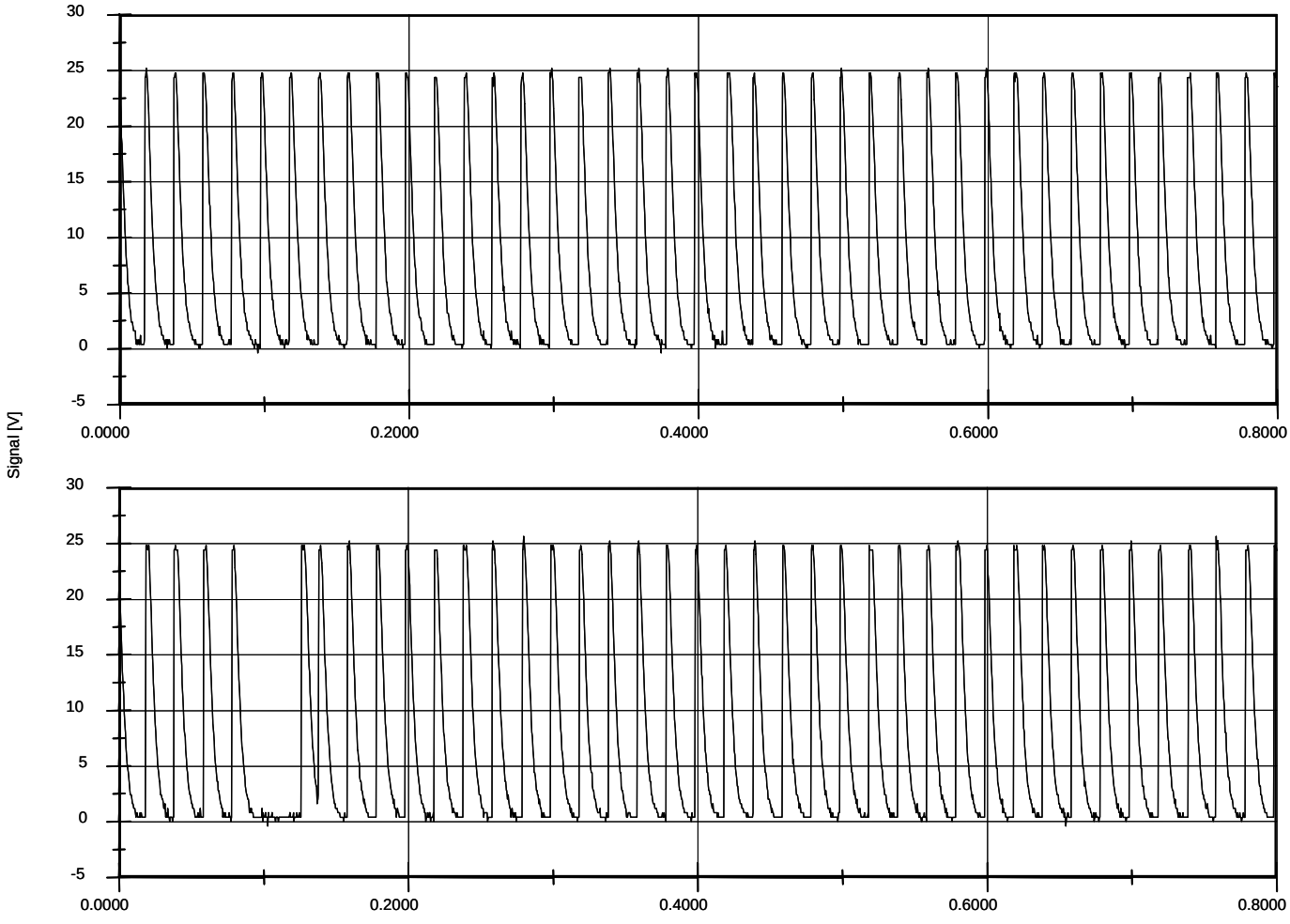

Fig. 3. Real-time behavior of the vacuum safety system for SHIPTRAP. Shown are digital pulses that are created each time the time critical loop iterates as a function of time. The time critical loop iterates at 50Hz. Top: Undisturbed loop iteration. Bottom: Occasional perturbation of regular loop iteration at 0.1s, see text.

that are configured to allow remote access via the ini-files will allow access.

However, these protection mechanisms are not intended for preventing malicious attacks from the outside. Therefore, the FieldPoint station should be situated in a secure network behind a firewall like all other control system nodes.

V. Performance

To account for the limited CPU power of FieldPoint controllers, the design and implementation of each individual class is elementary and straightforward. Since the *CS* framework already provides most of the infrastructure shown in Fig. 2, the main application specific work is the *GUI* and the *Sequencer*. For measuring the performance of the time critical part of the application, a digital pulse is produced upon each repetition of the time critical loop. The digital pulses were recorded on an oscilloscope and the frequency and jitter of the digital pulses is used as a measure for the real-time performance of the system. Fig. 3 shows the result of this measurement. The normal behavior is shown in the upper part of the figure. The time critical loop runs with a repetition rate of 50Hz. This corresponds to a guaranteed reaction time of 20ms. The jitter of the signal is in the order of 1ms. However, the recorded data yield an occasional disturbance as shown in the lower part of Fig. 3. At t=0.1s, one iteration is significantly delayed. This behavior can be explained as a consequence of the time required for the hardware I/O. Especially the 16bit ADCs use filters for noise reduction which results in a measurement time of about 20ms. As a possible solution, one could acquire the analog signals with lower priority and repetition rate. The complete application uses about 20MBytes of RAM, including the operating system. 12MBytes remain unused leaving room for further extensions.

VI. Conclusion

The *CS* framework, an object oriented approach to LabVIEW programming on MS-Windows and Linux platforms, has been presented in a previous work [1]. Within the work presented in this paper, the *CS* framework has been extended successfully to the real-time platform LabVIEW RT. This platform is typically employed for time critical and/or safety relevant sub-systems based on LabVIEW. The most important benefits from the work presented here are the following. First, existing code such as *CS* base classes developed for other operating systems can be reused. This guarantees an easy integration of a real-time sub system into a distributed SCADA system based on the *CS* framework. Second, many classes are specific for the real-time system but not specific for a single application. As a consequence, they can be reused by other LabVIEW RT applications. Third, the usage of LabVIEW is easy for non-specialists who work only a limited amount of time for an experiment or a facility. The required learning time is in the order of a couple of weeks, including the setting up of a new sub-system. To summarize, the new approach presented in this work drastically reduces the development time for new sub-systems. The vigorous usage of common classes, which are not specific for a particular application, will ease the maintenance as well as further extensions of applications significantly. As a first application of this work, a vacuum safety system for SHIPTRAP has been implemented using a PAC from National Instruments as the hardware platform.

References

[1] D. Beck, K. Blaum, H. Brand, F. Herfurth and S. Schwarz , "A new control system for ISOLTRAP", *Nucl. Instrum. Methods* A 527 (2004) pp. 567-579.

[2] R. Jamal and H.Pichlik, “LabVIEW Applications and Solutions” (1999) Prentice Hall. See also: http://www.ni.com/.

[3] H. G. Essel et al., "The new data acquisition system at GSI", *IEEE Trans. of Nucl. Science*, Vol.43, No.1 (1996) pp. 132-135.

[4] H. G. Essel and N. Kurz, "The general purpose data acquisition system MBS", *IEEE Trans. of Nucl. Science*, Vol.47, No.2 (2000) 337-339.

[5] For details on the ECoS (Experiment Control Systems) group see http://www.gsi.de/controls.

[6] The on-line documentation of *CS* is available at http://www-w2k.gsi.de/controls/CS/cs.htm.

[7] G. Sikler et al., "First on-line test of SHIPTRAP", *Nucl. Instrum. Methods*, B 204 (2003) pp. 482-486.

[8] C. Karigiannis, *diploma thesis*, technical college Darmstadt (2003) unpublished.

[9] 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, Canada, 8.-11. June 1993.

[10] For detailed information on the usage of PVSSII in large physics experiments see http://itcobe.web.cern.ch/itcobe/Projects/Framework/.