A new control system for ISOLTRAP

D. Beck [a,*], K. Blaum [a,b], H. Brand [a], F. Herfurth [a,b] and S. Schwarz [c]

[a] *GSI-Darmstadt, DVEE and AP, Planckstr. 1, D-64291 Darmstadt, Germany*

[b] *CERN, Division PH, 1211 Geneva 23, Switzerland*

[c] *NSCL, Michigan State University, East Lansing, MI 48824, USA*

Abstract

A new LabVIEW based control system for the ISOLTRAP facility at ISOLDE/CERN has been implemented by using the Control System (CS) framework which has been developed by DVEE/GSI during the last two years. CS is an object-oriented, multi-threaded, event-driven framework with Supervisory Control and Data Acquisition (SCADA) functionality. It allows one to implement distributed control systems by adding experiment specific add-ons. This paper gives an overview on the CS framework, describes the requirements for ISOLTRAP and reports on the implementation of the new control system.

Key words: control system, LabVIEW, unstable nuclei, Penning traps
PACS: 07.05.Dz, 07.05.Hd, 07.05.Bx, 82.80.Ms, 82.80.Qx, 21.10.Dr, 23.90.+w

1 Introduction

ISOLTRAP [1,2] is a facility tailored for on-line mass measurements of short-lived unstable nuclei, which are produced at ISOLDE/CERN [3]. The mass of a nuclide is determined by measuring the cyclotron frequency of ions stored in a Penning trap [4]. The motivation for high-precision mass measurements or determination of nuclear binding energies with ISOLTRAP is manifold. First, experimental mass values with relative uncertainties of $\delta m/m \approx 10^{-6}$ or better allow one to validate nuclear models which are needed to predict properties for nuclides not accessible by experiments, like the cross-sections for neutron capture along the r-process path [5]. Second, the systematic study

* corresponding author
Email address: d.beck@gsi.de (D. Beck).

of nuclear binding energies ($\delta m/m \approx 10^{-7}$) as a function of neutron and proton number yields information on nuclear structure effects like pairing, shell-closures and the onset of deformation [6–9]. Third, high-precision mass values ($\delta m/m \approx 10^{-8}$) for specific nuclides of interest contribute to test fundamental physics, like the conserved vector current (CVC) hypothesis, the unitarity of the Cabibbo-Kobayashi-Maskawa (CKM) mixing matrix, the isobaric multiplet mass equation (IMME) and the search for scalar or tensor currents in the weak interaction [10–13].

ISOLTRAP is a versatile facility. It can in principle be applied to all nuclides produced at ISOLDE and addresses a variety of questions in physics. This is reflected by the requirements to the control system. It must be flexible enough to easily adapt to the different experimental techniques employed. Furthermore, it must follow the changes and improvements of the experimental set-up. This does not only concern the experimental procedure but also the ever increasing number and exchange of hardware devices as well as hardware device types. The control system does not only serve to set and monitor static parameters. Instead, it must enable the experimentalists to actively manipulate the stored ions in consecutive steps, similar to an accelerator complex. Note, that the nuclides of interest have half-lives in the ms range requiring a "fast" control system. The beam time at ISOLDE is typically restricted to a few days per year and experiment. Beam time must not be lost and the control system needs to be reliable and stable. When conducting experiments, novices to the control system should quickly be able to learn how to run the apparatus. Since the control system is maintained and developed further by PhD students, a simple development environment is required.

A new control system framework, CS, has been developed at GSI [14]. It is object-oriented, multi-threaded, event-driven and provides some SCADA functionality. The development of the CS framework is driven by small experiments with about a few thousand process variable. A dedicated experiment control system can be implemented by adding a few experiment specific add-ons to the framework. As an example, the new control system for ISOLTRAP is presented in this paper.

2 ISOLTRAP

Figure 1 gives an overview of the experimental setup [1,2,15–17]. The main components are three ion traps in which the ions are manipulated by means of radio-frequency (rf) fields and buffer gas collisions. The RFQ structure serves for retardation, cooling and bunching of the 60 keV quasi DC beam delivered by ISOLDE. In the preparation Penning trap the ions are further cooled and isobaric contaminations are removed. The precision Penning trap is used to

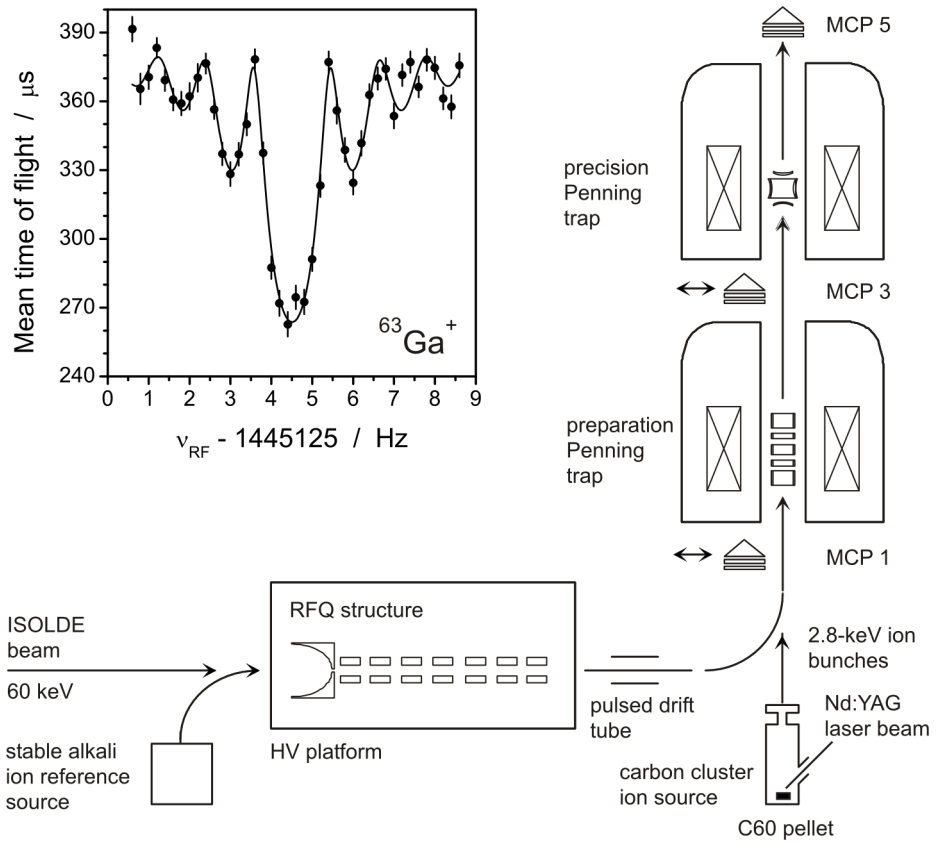

Fig. 1. The ISOLTRAP setup. The main components are three ion traps, the RFQ structure, the preparation Penning trap and the precision Penning trap. The insert in the upper left shows a resonance curve of the short-lived radio-nuclide $^{63}Ge^{+}$ ($t_{1/2} = 31.4\,s$). The true cyclotron frequency is determined by fitting the theoretically expected line shape (solid line) to the data points [18].

determine the mass of the stored ions. The experimental procedure includes an excitation of the ion motion with a rf-field oscillating near the true cyclotron frequency. When scanning the frequency of the rf-field, a resonance signal is obtained as shown in the insert of Fig. 1.

The experimental procedure consists of *cycles* and *scans*. A typical cycle for a nuclide with $t_{1/2} = 1\,s$ is depicted in Fig. 2. It consists of many steps which have to be synchronized with a precision of typically $1\,\mu s$. The start of a cycle is triggered by a proton pulse from the PS-Booster impinging on the ISOLDE target. The length of a cycle is typically around $0.3\,s - 1.5\,s$, depending on the half-life of the nuclide of interest. The result of such a cycle is one data point shown in the insert of Fig. 1. A resonance curve is obtained by performing several scans. A scan consists of many cycles, each at a different frequency value of the rf-field, starting from the lowest frequency point to the highest one. The curve shown in Fig. 1 consists of 42 scans, each scan consists of 41 cycles.

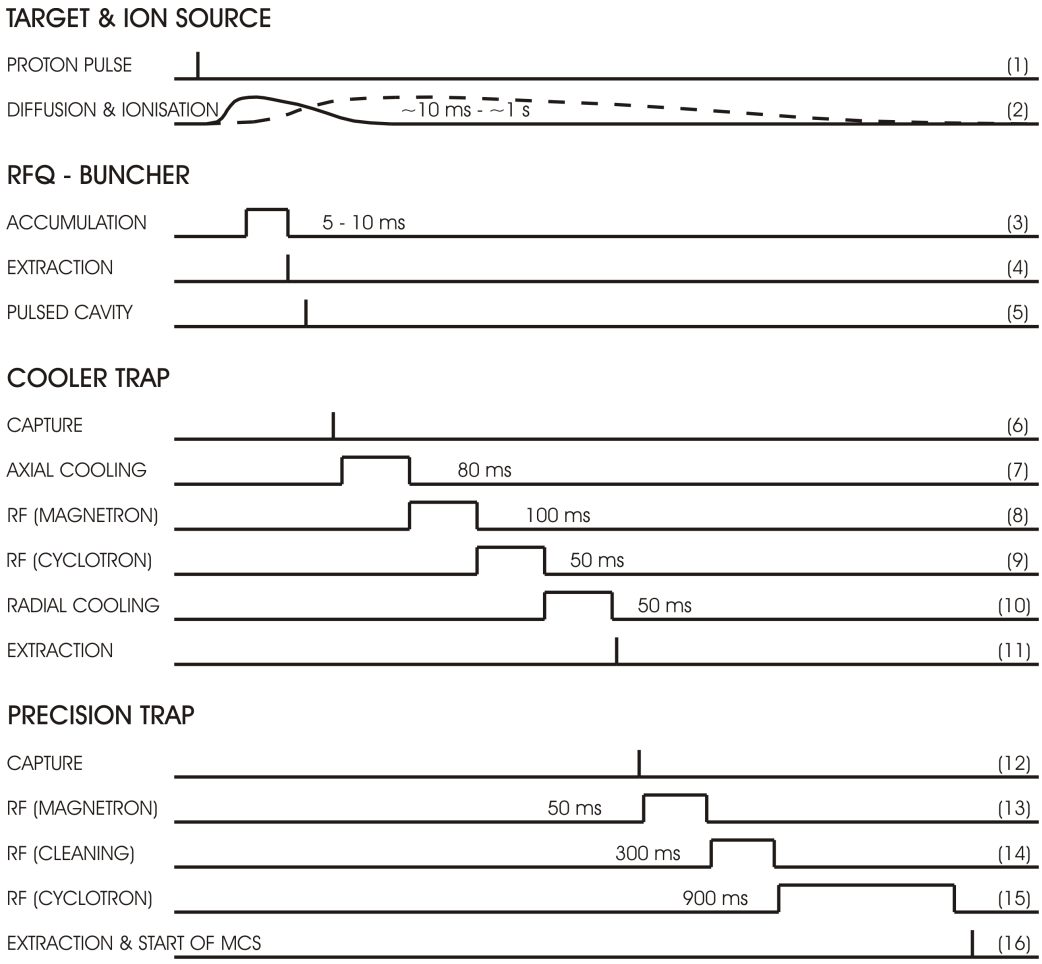

Fig. 2. Overview of a measurement cycle. A cycle consists of many steps that must be synchronized with a precision of typically 1 μs.

A rough sketch of the old control system used until spring 2003 is depicted on the left side of Fig. 3. A VME-bus with a Motorola E6 CPU served as the main platform for the control system that had been developed with GNU C [19]. The hardware devices of ISOLTRAP were controlled via GPIB, analog voltages, digital in- and outputs as well as Profibus. The graphical user interface (GUI) had been developed with Borland C++ and was operated from a PC which is connected to the VME-bus via TCP/IP [20]. This GUI served for configuring the control system and to analyze the acquired data on-line. It was continuously developed since many years and has been ported to Windows 2000 [1] . This old control system was successfully used for more than ten years. Unfortunately, the VME based hardware had become outdated and unreliable requiring a replacement.

The plan for the implementation of the new control system is sketched in Fig. 3. The GUI is reused as well as the existing hardware devices of ISOLTRAP. The

[1] The software runs on Windows XP as well.

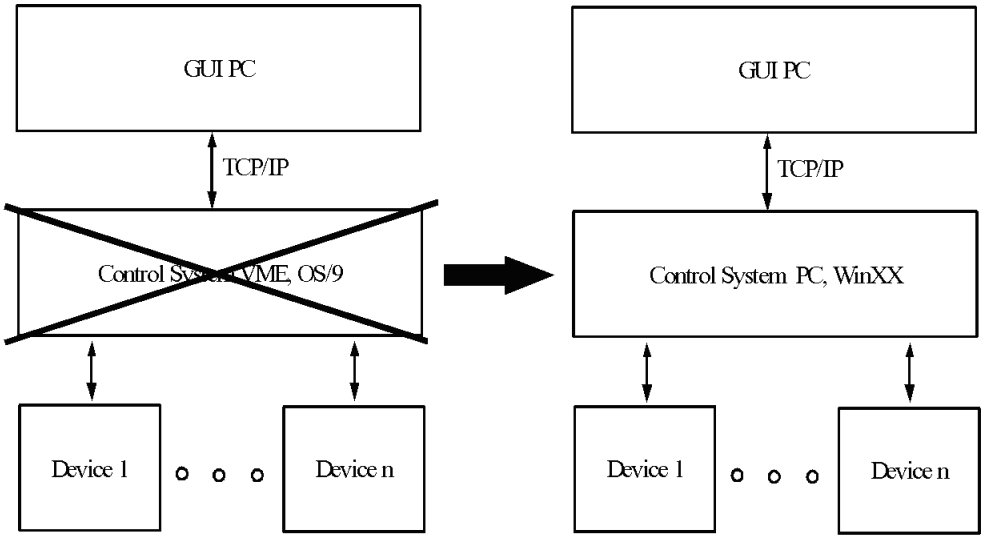

Fig. 3. Left: The old VME based control system of ISOLTRAP. Right: For the new control system, only the VME-bus system is replaced by a PC with new LabVIEW based control system software.

VME-bus is replaced by a PC. Instead of porting the control system software to the new platform, a completely new control system software based on the CS framework [14] has been developed.

3 CS framework

The interest in developing the CS framework is twofold: First, a couple of small experiments at GSI need a new control system and it was decided to base it on LabVIEW from National Instruments (NI) [21]. The framework aims at control systems with up to 10,000 process variables. Second, one would like to investigate the performance and scalability of a LabVIEW based system. This is an important question, since GSI will undergo a substantial upgrade to a new facility with larger experiments in the coming years [22]. Details on the implementation of the CS framework are not in the main line of this paper and are presented in the appendix.

3.1 Requirements

The main requirements summed up below are common to a couple of experiments and not specific to ISOLTRAP.

(1) A simple data acquisition must be included in the control system. Important is an unambiguous assignment of the measured data to each individual cycle. SCADA functionalities like alarming, trending and user management are desirable.
(2) A PhD student must be able to maintain and develop the system further after a few months of learning time, requiring only *one* development tool that is easy to learn.
(3) The devices are distributed over a larger area 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.
(4) Important is the ability to call any function of any device at any moment. This way, 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 which are not used in the preconfigured operational modes. This implies operational states to be configured on the fly.
(5) The control system should be open, so that new hardware can easily be included. Especially for small experiments, the challenge is not so much the number of I/O channels but the number of different hardware device types.
(6) For small experiments with only a few days of beam time per year, the stability and reliability of the control system is an important factor.

3.2 Solution path

LabVIEW from NI has been chosen as the development tool for the CS framework. It can easily be linked to third party hard- and software and it is very easy to build GUIs. An object-oriented approach is used to provide flexibility and maintainability. Each hardware device or software module is represented by its own object. The objects do not communicate via direct method calls but via events. This has three advantages: First, one can easily replace an object by another one, since the receiver of an event is only defined by the name of the object. Second, one can replace one method by another method, since methods are identified by the name of an event. Third, if an object should not be created on the local host but on a remote PC, one just sends the necessary events to the remote node. SCADA functionality like trending and alarming is provided by the Datalogging & Supervisory Control (DSC) module of LabVIEW. However, the functionality of the DSC module is encapsulated in a dedicated class. Hardware interfaces like RS232, RS485, GPIB, CAN or Profibus are based on PCI cards. Connected to those interfaces are external devices for control and data acquisition from various manufacturers (see Tab. 1). By this, one can avoid PC specific based measurement and automation hardware.

Although LabVIEW is not object-oriented, one can use an object-oriented approach within LabVIEW. One possibility is the usage of ObjectVIEW [23], a third party toolkit, that can be used together with LabVIEW. The CS framework has been inspired by ObjectVIEW but is independent from it. Objects are represented by Virtual Instruments (VIs) and classes by so called VI-templates (VIT). During run time, multiple objects (each object is a separate VI) can be created dynamically from a class (represented by a VIT) by the *VI-server* functionality of LabVIEW (see App. A.1). Inheritance is possible, but not as straight forward as in C++. A simplified view of the resulting class design is depicted in Fig. 4. The number of inheritance levels and classes is limited. Thus, the classes have more functionality than in a real object-oriented language. The base class of the CS framework is the *CAEObj* (Concurrent Active Event Object) 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 so-called 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.

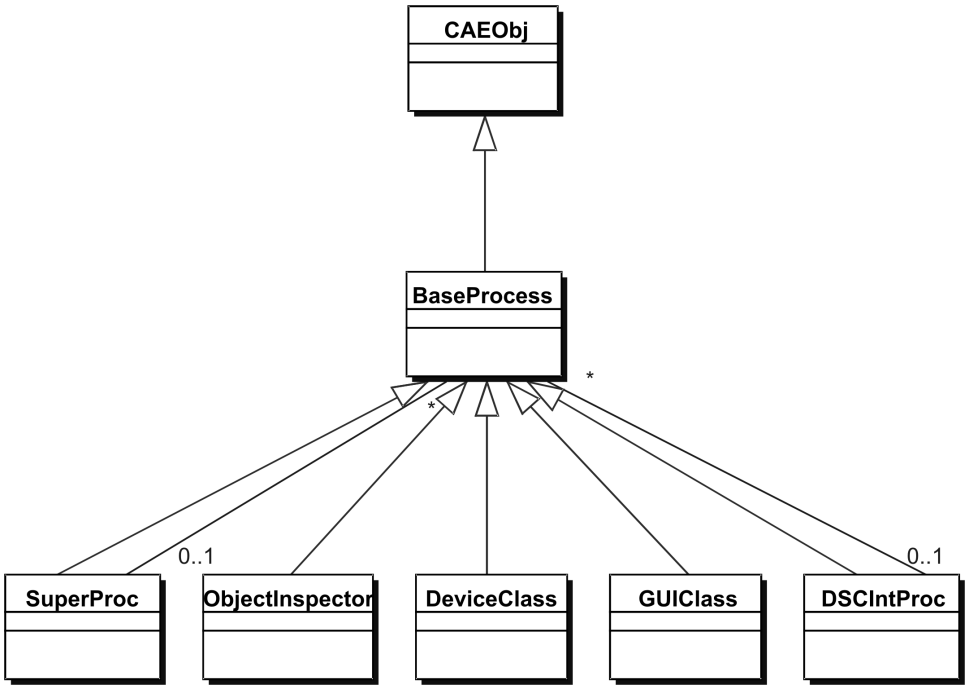

Fig. 4. Simplified class hierarchy of the CS framework depicted in the Unified Modeling Language (UML). Classes are represented as boxes. Arrows indicate inheritance dependencies. Lines indicate associations and their cardinality. Detailed explanation in the text.

The BaseProcess class is associated to two other classes. SCADA functionality is realized by the *DSCIntProc* class that provides the interface to the DSC module from NI. All alarm and trend values are sent to the DSCIntProc class for alarming and trending. The *SuperProc* class serves for creating and destroying objects of the BaseProcess and child classes [2]. Moreover, the SuperProc class allows to reserve objects for use so that they can not be destroyed accidentally. Almost all classes of the framework are direct children of the BaseProcess class. Each type of a hardware device is represented by a class.

Frequently, control systems are designed according to the so called three layer architecture which consists of a device layer, an application layer and a GUI layer. Within CS, even GUI and application layer classes are direct children of the BaseProcess class. Thus, the affiliation of a class to a specific layer is defined by the functionality of the class and not by its interface.

3.4 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 at any time. The callee (receiver) of an event and the method to be called are specified by their names, *ObjectName* and *EventName*. Data are passed as byte arrays. The BaseProcess class provides three types of events. *Simple*: The caller sends an event to the callee. *Synchronous*: The caller sends an event to the callee and waits for an answer of the callee before its thread continues. *Asynchronous*: The caller sends an event to the callee and the callee sends its answer to a third object which has been specified by the caller. Simple events may be buffered, using the LabVIEW message queues, or unbuffered, using the notifiers provided by LabVIEW. Other event types are always buffered. An object on a remote node is addressed in the form *ObjectName@NodeName*. As an example, Fig. 5 shows the principle of a synchronous call. The communication between different PCs is realized via dedicated client and server objects.

The BaseProcess class allows for introspection of an object by providing a method *GetDescriptors*. By this, one can query each object for its descriptors of all events defined by the class of that object. A descriptor contains the name and a text description of an event, the names of the parameters required, their data types as well as a text description for each parameter. Thus, documentation about each method and its parameters can be retrieved from an object. As a result, the use of an object via events is intuitive and

[2] The only exception is the SuperProc class itself. During run-time, one object of the SuperProc class is the first class to be created and the last object to be destroyed.

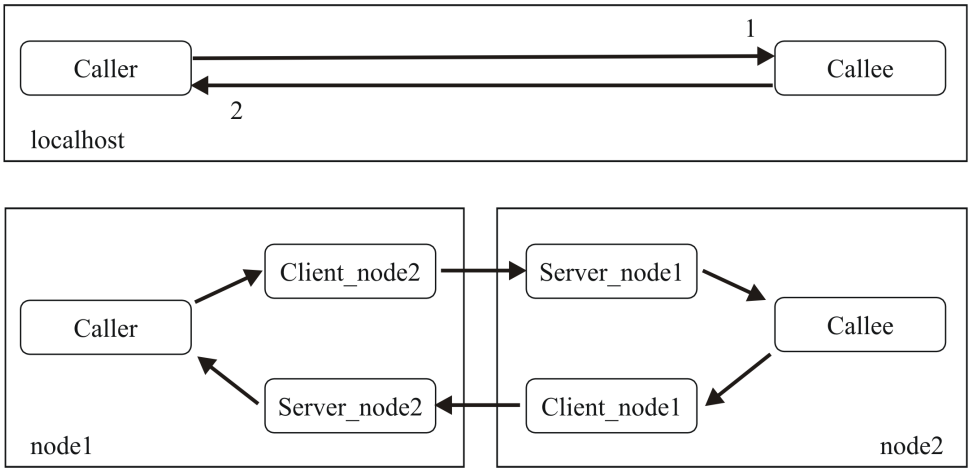

Fig. 5. A synchronous call to a callee on a local host (top) and on a remote host (bottom). The caller sends a message to the message queue of the callee (1). The latter sends the answer back to the caller via a temporary message queue (2).

almost self documenting. Furthermore, this feature allows the use of generic GUIs that can gather all required information on objects during run-time.

The event-driven communication of the CS framework does not require a central event manager. If the number of events that are sent to an object is higher than the event rate that can be handled by that object, the number of pending buffered events will pile up. Then, an object can no longer be addressed. This problem is solved by adding an optional timeout to an event. If the time specified by the timeout has passed, an event becomes invalid and will be discarded by the receiver object.

3.5 Database

In principle it is possible to instantiate as many objects of any class as required by the user during run time. Most objects need configuration data like the name of the hardware interface and the bus address of a device. Another example are the names of the objects that need to be created upon the startup of a control system. This information is stored in a database. If a *CreateObject*-event is sent to an object of the SuperProc class together with the name of the object to be created, the SuperProc class queries the configuration database via the Structured Query Language (SQL). This query yields the class name of the object together with the required configuration data. Finally, the requested object is created by the SuperProc object.

3.6 Examples of GUI classes

Since a typical GUI class is also a child of the BaseProcess class, one can send events to a GUI and use it as a compound device or subsystem. Two examples will be given.

The *ObjectInspector* class allows to retrieve information on the created objects like their age, the number of pending events, the status and the error of the individual threads and their event rate. The user interface of the ObjectInspector runs in a separate thread. However, querying the objects for their properties is done in the event handling thread, so an ObjectInspector must send an event to itself to retrieve the required information. This results in a minute overhead but enables an ObjectInspector on one PC to obtain information of objects on a remote PC by sending an event to the remote ObjectInspector object.

The *GeneralObjectGUI* (GOG) is a generic user interface. It can be used to send commands to and retrieve information from the created objects. First, the user generates a list of all objects he/she is interested in. For this, the names of all pre-configured objects are obtained by querying the database with an SQL-statement. Then, the user can choose the required objects. Second, those objects are created. Third, one obtains a list of all methods for each object by querying the objects. Then, the user can select the methods that should be executed. As a result, the user can configure a subsystem like an electrostatic beam line. Since the objects and their method are called in a well defined order, a GOG object can moreover be used as a simple sequencer. The configuration of a GOG object can be stored in an Extensible Markup Language (XML) configuration file. Note, that such a configuration file can be loaded remotely by sending an event to one GOG object from any other object.

3.7 Application and performance of the CS framework

The CS framework has been developed at GSI. The code is available under the GNU General Public License (GPL) and can be downloaded [24]. Table 1 lists the device types that are presently supported. One of the strengths of the framework is the ability to connect to all kind of hard and software. First, new devices for all interfaces supported by LabVIEW like RS232, RS485, GPIB, CAN, OPC and others can easily be incorporated. Second, the framework can be linked to non-LabVIEW applications by implementing specific interface classes. The other outstanding feature of the framework is its flexibility that allows its application not only to ISOLTRAP, but also to other projects like life-time measurements of highly charged ions [25], SHIPTRAP [26], LEBIT

Table 1
Overview of the devices that are supported by the CS framework. The package name is given in the first column. The name of the model and the manufacturer are listed in the next two columns. Some classes have been written for commonly used devices with Interchangeable Virtual Instruments (IVI) drivers. IVI is a standard for instrument driver software. Models marked by * are used at ISOLTRAP.

Package	Model	Manufacturer
CSAcquisition	IVIOscilloscope	all oscilloscopes with IVI drivers
	SR430*	Stanford Research Systems
	TDS640	Tektronix
CSAfg	IVIWaveGen	all arbitrary function generators with IVI drivers
	AG33250A*	Agilent
	DS345*	Stanford Research Systems
	HP3325B*	Hewlett-Packard
	SML01	Rohde & Schwarz
CSDelayGate	BNC555	Berkeley Nucleonics
	DF94011*	University of Mainz, Germany
	IECMemory*	University of Mainz, Germany
	PPG100	Becker & Hickl
	QC9310*	Quantum Composer
	DG535	Stanford Research Systems
CSMotion	PA-Control	IEF Werner
	SixPack	Trinamics
CSPowerSupply	AnalogDevice*	all power supplies controlled by analog I/O
	EHQFXXX	iseg
	HCN*	FUG
	HVSwitch	GSI-Darmstadt
	HP6629A	Hewlett-Packard
	PBPowerSupply*	analog devices via Profibus
	PBRelay*	digital devices via Profibus
CSVacuum	RVC300	Pfeiffer Vacuum

[27] and PHELIX [28].

Presently, the event rate for a synchronous call (see sect. 3.4) can be as high

as a few hundred Hertz per PC. This includes already the overhead for error handling, alarming, trending and watchdog functionality. Typically, around 100 objects are created per PC. One has to keep in mind that each object has quite some functionality and, typically, corresponds to one hardware device. On a PIII CPU with 500MHz, the overhead of the CS framework results in a CPU load of a few percent.

If distributed over several PCs, the CS framework can handle up to $10,000$ process variables. Scaling the framework to a larger number of process variables does not seem to be feasible. This is due to the properties of LabVIEW [3] [29]: First, the application requires a significant amount of memory. One object typically uses a few MBytes of RAM. Second, it takes about one second to create one object. Moreover, this time increases with the number of objects that have already been created. These two facts limit the number of objects per PC to a few hundred. Third, occasional crashes of the LabVIEW environment are observed. This is not yet understood, but results in a practical limitation for the number of PCs involved in one control system.

4 The ISOLTRAP control system

The control system must be capable of performing the scans and cycles required to obtain a resonance signal as shown in the insert of Fig. 1. The necessary steps are described further down. The timing requirements of about 100 ns within a cycle (see Fig. 2) are realized in hardware. The VME-bus system is replaced by a PC. The control system software is based on the CS framework. Only a few experiment specific features have been added. Although the CS framework supports distributed control systems, the first version of the new control system for ISOLTRAP uses only one rack-mountable PC for simplicity [4]. The devices of the set-up are connected via PCI interface cards plugged into the PC. All devices can be accessed either by GPIB or via OPC. Presently, two GPIB cards from NI and one Profibus master card from Beckhoff are used. The connection to the RFQ buncher, which is operated on high-voltage, is provided by optical GPIB and Profibus links.

Figure 6 shows a simplified view of the ISOLTRAP control system. Two PCs are involved. The *Control PC* replaces the old VME-bus system. The *GUI PC* and the *Control and on-line analysis GUI* are basically unchanged (see Fig. 3). Typically around 70 objects are created for operating the experiment. As examples, eight objects are shown in Fig. 6. *SR430* is a multi-channel-

[3] Presently, the CS framework uses LabVIEW 7.0.

[4] The GUI PC does not count as a second PC, since its software is not based on the CS framework.

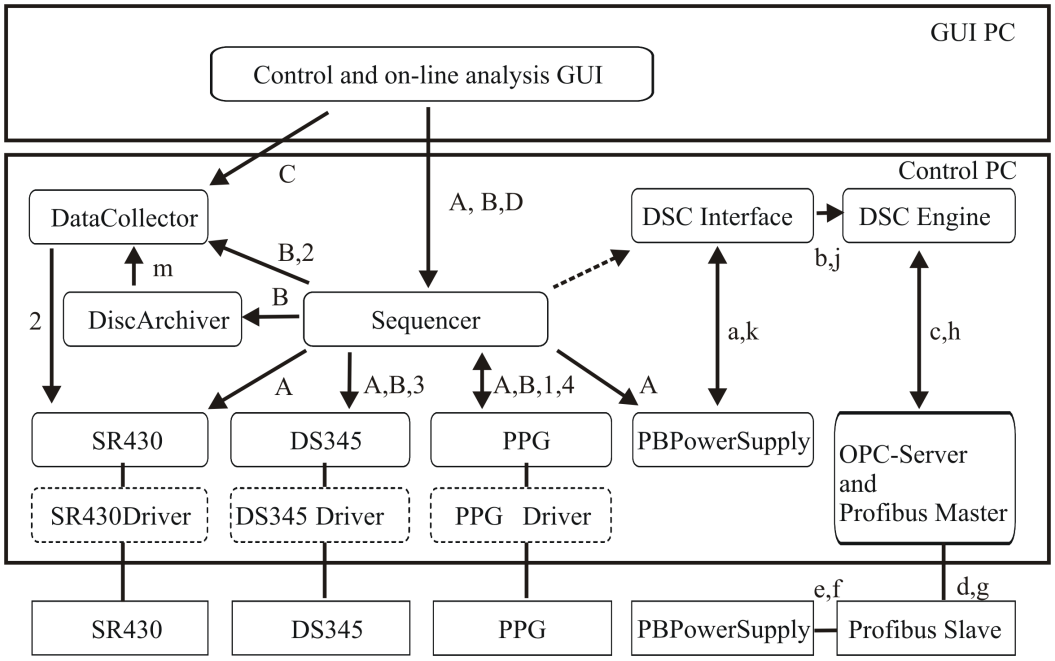

Fig. 6. Simplified layout of the ISOLTRAP control system. Hardware is shown in rectangular boxes. Software is depicted by boxes with rounded corners. Active (inactive) software objects have solid (dashed) bordered boxes. Arrows indicate event-driven communication. The direction of the arrows do not indicate the direction of the data flow, but mark caller and callee. To simplify the figure, trending and alarming is indicated only for the *Sequencer* but not for the other active objects on the *Control PC* (dashed arrow). Direct method calls and hardware connections are marked by lines. Uppercase and lowercase letters as well as numbers denote communication paths (see text).

scaler for data acquisition. *DS345* is a function generator to generate rf-fields for the excitation of the ion motion. *PPG* is a pulsed patter generator and produces bit patterns with a precision of 100 ns. Those bit patterns trigger hardware devices like delay gate generators (not shown). *PBPowerSupply* is a power supply controlled by analog I/O of the Profibus. The *Sequencer* is the heart of the control system and will be explained below. The *DataCollector* serves to collect and buffer data from acquisition devices. The *DiscArchiver* retrieves the buffered data from the DataCollector and writes it to a permanent storage device. The *DSCInterface* is the interface between all objects and the *DSCEngine* from NI. The DSCEngine represents the DSC module from NI. It serves for trending and alarming as well as a client that is connected to an *OPC Server and Profibus Master* via OLE for Process Control (OPC). Like the DSCEngine, the OPC server and Profibus master is a commercial product (Beckhoff).

The following actions are initiated via the GUI.

A. The configuration data for a measurement is sent to the Sequencer. All

objects involved are configured and consequently the hardware devices initialize.

B. Start of a measurement. The Sequencer calls the DiscArchiver and the DataCollector objects. The DiscArchiver closes a possibly open file and opens a new file. The DataCollector clears the data buffers. The DS345 is set to the frequency of the first cycle of a scan. Finally, the PPG is triggered and the measurement started.

C. The GUI retrieves buffered data from the DataCollector. The data are used for on-line analysis by the GUI.

D. The measurement is stopped by the user.

After a measurement has been started by the user, the experiment runs on its own. No interaction from the GUI or the user is required. The Sequencer takes over and controls the experiment.

1. After the PPG has been triggered, it starts to produce bit patterns that control the timing and trigger devices like delay-gate generators or pulsed power supplies within a cycle. An example for the timing of a typical cycle is depicted in Fig. 2. After a cycle is finished, the PPG sends an event to the Sequencer.

2. The Sequencer sends an event to the DataCollector that collects and buffers the data from all used acquisition devices. In this example, the SR430 is the only acquisition device. If the DataCollector encounters problems, the Sequencer stops.

3. An event is sent to the DS345 together with the rf-frequency value for the next cycle. If the DS345 encounters problems, the Sequencer stops.

4. If all previous steps have been executed successfully, the Sequencer sends an event to the PPG. This will trigger the PPG and the next cycle may start (step 1).

The DiscArchiver is not synchronized to the cycle. From time to time, it retrieves the buffered data from the DataCollector (m). If the DiscArchiver fails to retrieve the data, the Sequencer can still continue (steps $1-4$). However, if the amount of buffered data exceeds a specified limit, the DataCollector will report an error in step 2 and the Sequencer discontinues the measurement. The GUI is treated differently by the DataCollector. If the GUI fails to retrieve data for on-line analysis (C), the DataCollector will *not* report an error in step 2. By this, a measurement can continue in case the connection to the GUI PC breaks or the GUI is closed by the user.

The communication with the Profibus devices is not synchronized with the Sequencer. This is an advantage since, as an example, a changed read-back value of a static voltage of a PBPowerSupply can be updated in the control system independently of the Sequencer.

a. The PBPowerSupply sends a set-value of a power supply to the DSCInterface.
b. The value is written to the real time database of the DSCEngine.
c. The DSCEngine, that also acts as an OPC client, transmits the value to the OPCServer.
d. The OPCServer writes the value to the Profibus master that includes the set-value into the cyclic data exchange of the Profibus.
e. Via a DAC, the ProfibusSlave sets the value at the PBPowerSupply device.
f. The PBPowerSupply transmits the get-value to an ADC of the ProfibusSlave.
g. The ProfibusSlave includes the new get-value in the data exchange of the Profibus.
h. The Profibus master reads the get-value from the Profibus and writes it to the OPCServer that transmits the value to the DSCEngine.
j. The DSCInterface periodically checks the DSCEngine for changed values.
k. If the get-value has changed, the DSCInterface sends an event with the new value to the PBPowerSupply object, where it can be used for different purposes.

The link between the GUI PC and the control PC is provided by a protocol based on TCP/IP. An interface class *CommandServer* (not shown in Fig. 6) has been implemented in the LabVIEW part of the control system. All communication between the LabVIEW based control system and the GUI is handled by the CommandServer. The CommandServer emulates the protocol that was used on the old VME-bus system. Only minute changes were required for reusing the old GUI with the new control system.

In total, four classes are specific to the ISOLTRAP experiment, but only the Sequencer and the GUI are shown in Fig. 6. All other components are part of and provided by the CS framework. Of course, some classes for hardware devices types, like the one for the AG33250A from Agilent, have been written while implementing the new control system. However, these classes can be used by other experiments as well and are not specific to the ISOLTRAP experiment. It should be noted, that even ISOLTRAP-specific software is reusable. The GUI, the Sequencer and the CommandServer have been reused for the control system of SHIPTRAP at GSI [26] and LEBIT at MSU [27].

5 Conclusion

ISOLTRAP is a versatile facility for high-precision mass measurements of short-lived nuclides. The mass values are used to test nuclear models, for investigating nuclear structure effects and as input data for experiments testing the Standard Model. Different questions in physics require the acquisition of

different types of data under different experimental conditions. This results in frequent changes of the experimental procedure that are only possible with a flexible control system that is an integrated part of the experimental setup. Due to aging of the hardware, the old VME-bus based control system had to be replaced.

The LabVIEW based CS framework [14] has been chosen as the basis for the new control system. The CS framework has been developed at GSI during the last two years. It is object-oriented, multi-threaded, event-driven, distributed and provides some SCADA functionality. Today, CS provides classes for about twenty different device types as well as two classes for devices with available IVI drivers. The CS framework is used to implement control systems for a couple of experiments. The software is GPL licensed and available for download [24].

Due to the use of the CS framework, the new control system for ISOLTRAP was implemented in about nine man months. So far, the ISOLTRAP control system is the most complex control system that is based on the CS framework and already data taking. It not only replaces the old control system but provides more functionality. This does not only concern new types of devices but also features like alarming and historic trending. This enhances the experimental capabilities of ISOLTRAP that are required to produce outstanding data for exciting physics.

6 Acknowledgements

We gratefully acknowledge GSI for supporting the development of the CS framework. Many thanks to our colleagues Georg Bollen, Wolfgang Geithner, Harald Hahn, Heinz-Jürgen Kluge, Manas Mukherjee, Klaus Poppensieker, Mathias Richter, Sumit Saxena, Uwe Thiemer, Chabouh Yazidjian and others who contributed ideas and software. We would like to thank the people from Vogel Automatisierungstechnik for fruitful discussions.

A Implementation

The CS framework is implemented in LabVIEW and has been inspired by the third party toolkit ObjectVIEW [23]. Of capital importance for the implementation is the use of the *VI-Server*-methods. In the following a few examples of code will be given.

A.1 Creating an object

Figure A.1 shows the LabVIEW code that is required to create an object of a class. A class is implemented as a so-called VI-template (*.vit). Only the path

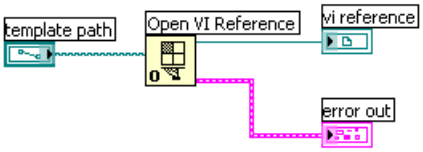

Fig. A.1. LabVIEW code required to create an object. *Template path*: Path to the VI-template (class code). *VI reference*: Reference to the created VI (object).

to that VI-template must be specified. Then, the *Open VI reference.vi* creates the object and returns the reference to that object.

A.2 Inheritance in LabVIEW

Child classes of the BaseProcess class will inherit the event handling thread and the periodic action thread of the BaseProcess class. All events defined together with all methods of the BaseProcess class will be inherited. Typically, the method executed inside the periodic action thread will not be inherited but overloaded by the child class. A child class must provide five methods.

(1) ProcConstructor: constructor code
(2) ProcDesctructor: desctructor code
(3) ProcEvents: defines events and their descriptors (see Fig. A.2)
(4) ProcCases: calls methods that correspond to the defined events (see Fig. A.3)
(5) ProcPeriodic: method called within the periodic action thread

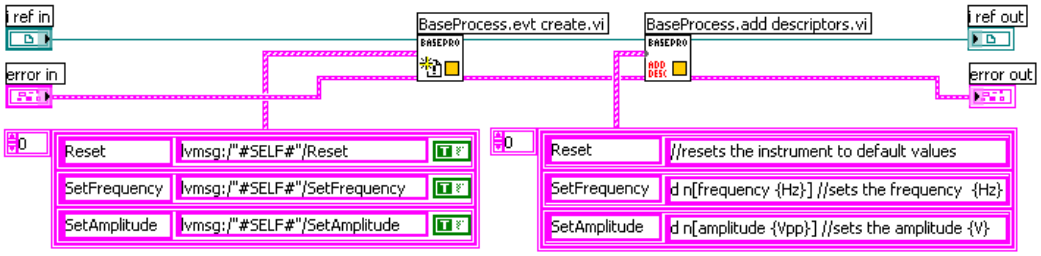

Fig. A.2. Method AG33250A.ProcEvents. Definition of events for the class implementing an arbitrary function generator AG33250A from Agilent.

If a new class has been created, only these five methods must be provided by the developer. As an example, the class for the function generator AG33250A has the public methods *Reset*, *SetFrequency* and *SetAmplitude*. Those public methods are typically placed within the method *ProcCases*. They are executed

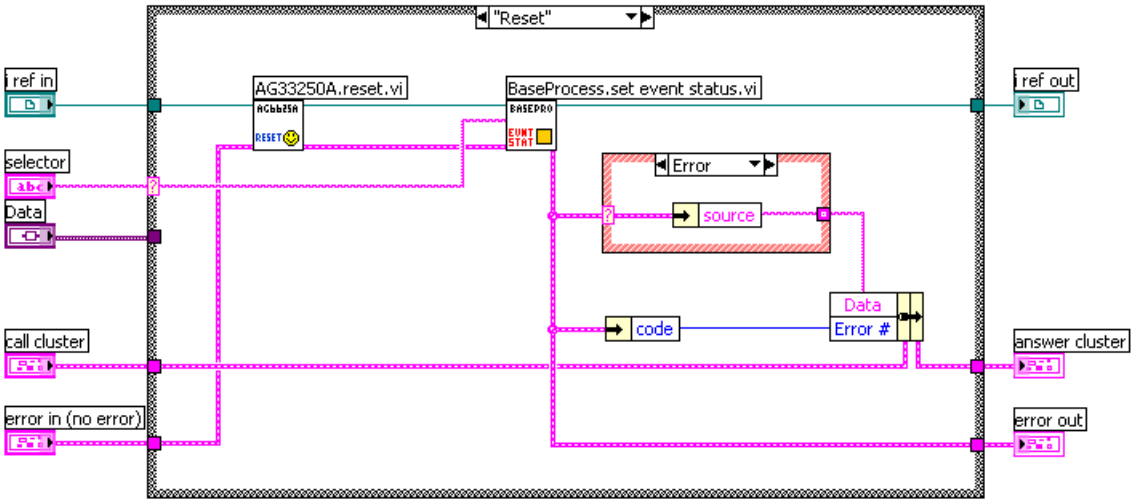

Fig. A.3. Method AG33250A.ProcCases. This is a case structure that calls the public methods that correspond to the events defined in the method ProcEvents (see Fig. A.2). Shown is the code required to call the public method *AG33250A.Reset*.

when the object receives the corresponding event. The five methods above are called via virtual function calls from methods of the BaseProcess class. Figure A.4 shows an example of a virtual function call in LabVIEW. Any method

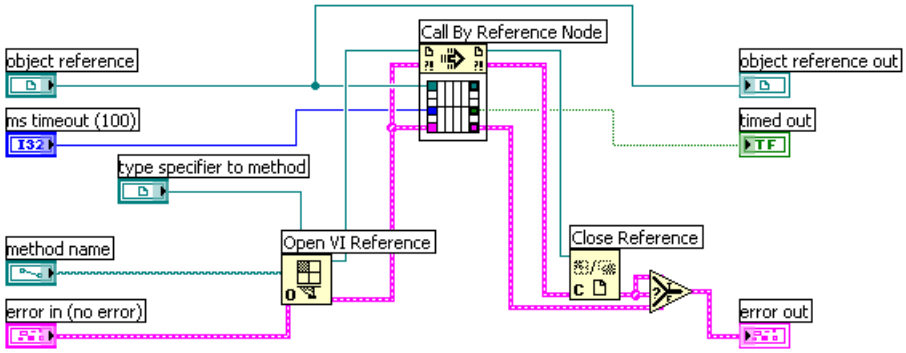

Fig. A.4. A virtual function call in LabVIEW. Any method or VI of known type can be called by its name.

or VI of known type can be called by its name. By this, the event handling thread, that is inherited from the BaseProcess class, can call the method *AG33250A.ProcCases* by its name. Child classes of the BaseProcess class only need to provide a simple wrapper around *main* method of the BaseProcess class. Hence, child classes execute exactly the same code as the BaseProcess class except for the five methods that are mentioned above. Figure A.5 shows the *main* method of the AG33250A class. The main method of the AG33250A class is extremely simple, since the functionality of the BaseProcess class is encapsulated in the main method of the BaseProcess class. To summarize, a developer of a new class *ChildClass* must at least provide five methods. Public methods of ChildClass are called from the methods *ChildClass.ProcCases*. This simplifies the development of a new class that is a child class of the

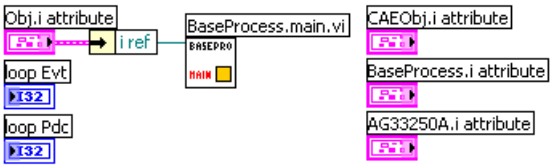

Fig. A.5. *Main* method of the AG33250A class. It is just a wrapper around the *main* method of the BaseProcess class.

BaseProcess class.

References

[1] G. Bollen et al., Nucl. Instr. Meth. A 368 (1996) 675-697.

[2] F. Herfurth et al., Nucl. Instr. Meth. A 469 (2001) 254-275.

[3] E. Kugler et al., Nucl. Instr. Meth. B 70 (1992) 41.

[4] L. S. Brown und G. Gabrielse, Rev. Mod. Phys. 58 (1986) 233-311.

[5] P. Möller et al., Atomic Data Nucl. Data Tables 59 (1995) 185-381.

[6] F. Ames et al., Nucl. Phys. A 651 (1999) 3-30.

[7] D. Beck et al., Eur. Phys. J. A 8 (2000) 307-329.

[8] S. Schwarz et al., Nucl. Phys. A 693 (2001) 533-545.

[9] H. Raimbault-Hartmann et al., Nucl. Phys. A 706 (2002) 3-14.

[10] A. Kellerbauer et al., Nucl. Phys. A (2003) submitted and Phys. Rev. Lett. in preparation.

[11] F. Herfurth et al., Phys. Rev. Lett. 87 (2001) 142501.

[12] K. Blaum et al., Phys. Rev. Lett. 91 (2003) 260801.

[13] F Herfurth et al., Eur. Phys. J. A 15 (2002) 17-20.

[14] D. Beck and H. Brand, GSI Scientific Report 2002 (2003) 210.

[15] K. Blaum et al., Nucl. Instr. Meth. B 204 (2003) 478-481.

[16] A. Kellerbauer et al., Int. J. Mass Spectr. 229 (2003) 107-115.

[17] F. Herfurth et al., J. Phys. B 36 (2003) 931-939.

[18] M. König et al., Int. J. Mass Spectr. Ion. Proc. 142 (1995) 95-116.

[19] J. Emmes, University of Mainz, Germany (1993) thesis.

[20] W. Rohde, University of Mainz, Germany (1993) thesis.

[21] R. Jamal and H. Pichlik, "LabVIEW Applications and Solutions" (1999) Prentice Hall.

[22] Conceptual Design Report, "An International Accelerator Facility for Beams of Ions and Antiprotons" (2001) GSI

[23] R. Buhrke, "LabVIEW Technical Resource" Vol. 9 (2002) No. 3.

[24] For online documentation and software download, check http://www.gsi.de/ and http://labview.gsi.de/.

[25] S. Toleikis et al., GSI Scientific Report 2002 (2003) 86-87.

[26] J. Dilling et al., Hyp. Int. 127 (2000) 491-496.

[27] S. Schwarz et al., Nucl. Instr. Meth. B204 (2003) 507-511.

[28] E. W. Gaul et al., GSI Scientific Report 2002 (2003) 101-103.

[29] D. Beck et al., Proceedings " IX International Conference on Accelerator and Large Experimental Physics Control Systems", Gyeongju, Korea, October 2003, in print.