SICS: SINQ Instrument Control Software

Mark Könnecke
Heinz Heer
Labor für Neutronenstreuung
Paul Scherrer Institut
CH-5232 Villigen PSI
Switzerland
Mark.Koennecke@psi.ch
Heinz.Heer@psi.ch

Contents

1. Introduction

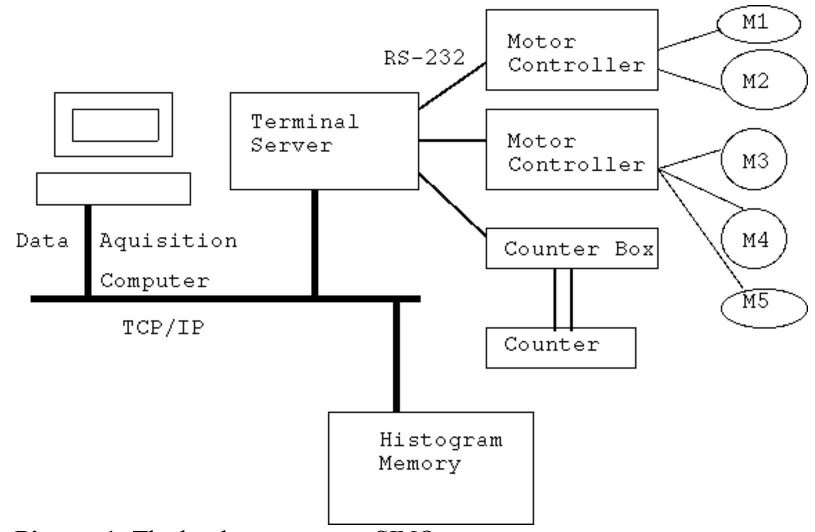
At the new spallation source SINQ at PSI a whole set of new neutron scattering instruments are being installed. All these new instruments need a instrument control control system. After a review of similar systems out in the market it was found that none fully met the requirements defined for SINQ or could easily be extended to do so. Therefore it was decided to design a new system. This new system SICS, the SINQ Instrument Control System, had to meet the following specifications:

- Control the instrument reliably.
- Good remote access to the instrument via the internet.
- Portability across operating system platforms.
- Enhanced portability across instrument hardware. This means that it should be easy to add other types of motors, counters or other hardware to the system.
- Support authorization on the command and variable level. This means that certain instrument settings can be protected against harmful changes by less knowledgable users.
- Good maintainability and extendability.
- Be capable to acomodate graphical user interfaces (GUI).
- Single code base for all instruments.
- Powerful macro language.

A suitable new system was implemented using an object oriented design which matches the above criteria.

2. The SINQ Hardware Setup

SICS had to take in account the SINQ hardware setup which had been decided upon earlier on. Most hardware such as motors and counters is controlled via RS-232 interfaces. These devices connect to a Macintosh PC which has a terminal server program running on it. This terminal server program collects request to the hardware from a TCP/IP port and forwards them to the serial device. The instrument control program runs on a workstation running DigitalUnix. Communication with the hardware happens via TCP/IP through the terminal server. Some hardware devices, such as the histogram memory, can handle TCP/IP themselves. With such devices the instrument control program communicates directly through TCP/IP, without a terminal server. All hardware devices take care of their real time needs themselves. Thus the only task of the instrument control program is to orchestrate the hardware devices. SICS is designed with this setup up in mind, but is not restricted to it. A schematic view of the SINQ hardware setup is given in picture 1.



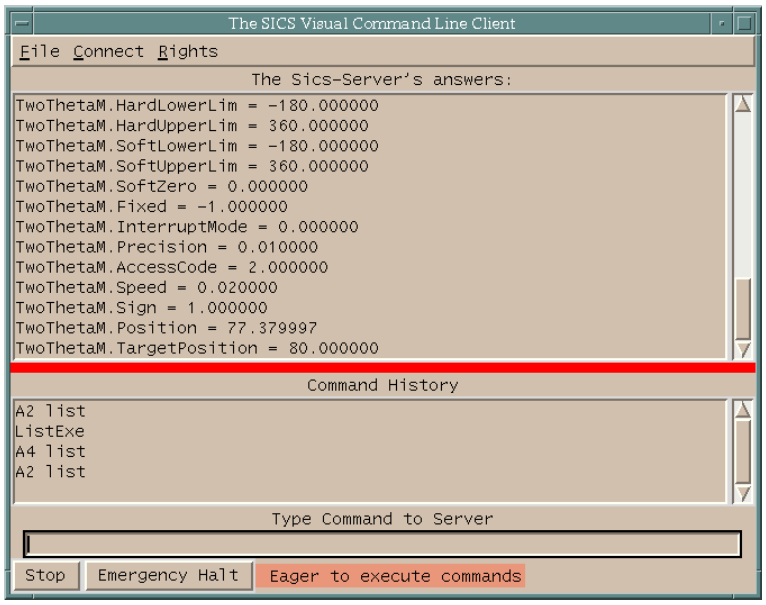
Picture 1: The hardware setup at SINQ.

3. SICS Overall Design

In order to achieve the design goals stated above it was decided to divide the system into a client server system. This means that there are at least two programs necessary to run an instrument: a client program and a server program. The server program, the SICS server, does all the work and implements the actual instrument control. The SICS server usually runs on the DAQ computer. The client program may run on any computer on the world and implements the user interface to the instrument. Any numbers of clients can communicate with one SICS server. The SICS server and the clients communicate via a simple ASCII command protocol through TCP/IP sockets. With this design good remote control through the network is easily achieved. SICS clients can be implemented in any language or system capable of handling TCP/IP. Thus the user interface and the functional aspect are well separated. This allows for easy exchange of user interfaces by writing new clients.

4. SICS Clients

SICS Clients implement the SICS user interface. Current client programs have been written in Tcl/TK[1], but work is under way to recode clients in Java for maximum platform portability. This is a real concern at SINQ where VMS, Intel-PC, Macintosh and Unix users have to be satisfied. As many instrument scientists still prefer the command line for interacting with instruments, the most used client is a visual command line client. Status displays are another sort of specialized client programs. Graphical user interfaces are under consideration for some instruments. As an example for a client a screen shot of the command line client is given in picture 2.


The SICS Visual Command Line Client
File Connect Rights
The Sics-Server's answers:
Command History
Type Command to Server
Stop
Emergency Halt
Eager to execute commands

```
TwoThetaM.HardLowerLim = -180.000000
TwoThetaM.HardUpperLim = 360.000000
TwoThetaM.SoftLowerLim = -180.000000
TwoThetaM.SoftUpperLim = 360.000000
TwoThetaM.SoftZero = 0.000000
TwoThetaM.Fixed = -1.000000
TwoThetaM.InterruptMode = 0.000000
TwoThetaM.Precision = 0.010000
TwoThetaM.AccessCode = 2.000000
TwoThetaM.Speed = 0.020000
TwoThetaM.Sign = 1.000000
TwoThetaM.Position = 77.379997
TwoThetaM.TargetPosition = 80.000000
```

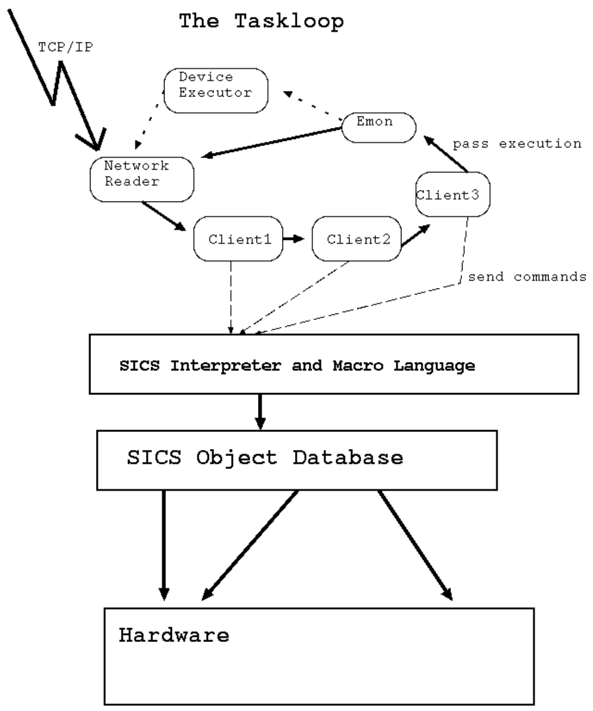
```
A2 list
ListExe
A4 list
A2 list
```

Picture 2: This picture shows a sample command line client for SICS. The upper part shows the SICS server's replies, the lower area shows a history of commands and the line at the very bottom is the command input. The lowermost row contains a few buttons for quickly stopping an instrument if something goes wrong and a status line which says what the server is doing.

5. The SICS Server

The SICS server is the core component of the SICS system. The SICS server is responsible for doing all the work in instrument control. Additionally the server has to answer the requests of possibly multiple clients. The SICS server can be subdivided into three subsystems: The kernel, a database of SICS objects and an interpreter. The SICS server kernel takes care of client multitasking and the preservation of the proper I/O and error context for each client command executing.

SICS objects are software modules which represent all aspects of an instrument: hardware devices, commands, measurement strategies and data storage. The SICS server's database of objects is initialized at server startup time from an initialization script. The third SICS server component is an interpreter which allows to issue commands to the objects in the objects database. A schematic drawing of the SICS server's structure is given in picture 3.



Picture 3: The schema shows the task loop, the interpreter and the SICS object database.

5.1 The SICS Server Kernel

Any server-program has the problem how to organize multiple clients accessing the same server and how to stop one client reading data, which another client is just writing. The approach used for the SICS server is a combination of polling and cooperative multitasking. This scheme is simple and can be implemented in an operating system independent manner. One way to look at the SICS server is as a series of tasks in a circular queue executing one after another. There are several system tasks and one task for each living client connection. The servers main loop does nothing but executing the tasks in this circular buffer in an endless loop. Thus only one task executes at any given time and data access is efficiently serialized.

One of the main system tasks (and the one which will be always there) is the network reader. The network reader has a list of open network connections and checks each of them for pending requests. What happens when data is pending on an open network port depends on the type of port: If it is the servers main connection port, the network reader will try to accept and verify a new client connection and create the associated data structures. If the port belongs to an open client connection the network reader will read the command pending and put it onto a command stack existing for each client connection. When it is time for a client task to execute, it will fetch a command from its very own command stack and execute it. When the net reader finds an user interrupt pending, the interrupt is executed. This is how the SICS server deals with client requests.

The scheme described above relies on the fact that most SICS command need only very little time to execute. A command needing time extensive calculations may effectively block the server. Implementations of such commands have to take care that control passes back to the task switching loop at regular intervalls in order to prevent the server from blocking.

Another problem in a server handling multiple client requests is how to maintain the proper execution context for each client. This includes the clients I/O-context (socket), the authorisation of the client and possible error conditions pending for a client connection. SICS does this via a connection object, a special data structure holding all the above information plus a set of functions operating on this data structure. This connection object is passed along with many calls throughout the whole system.

Multiple clients issuing commands to the SICS server may mean that multiple clients might try to move motors or access other hardware in conflicting ways. As there is only one set of instrument hardware this needs to be prevented. This is achieved by a convention. No SICS object drives hardware directly but registers it's request with a special object, the device executor. This device executor starts the requested operation and reserves the hardware to the client for the length of the operation. During the execution of such an hardware request all other clients requests to drive the hardware will return an error. The device executor is also responsible for monitoring the progress of an hardware operation. It does so by adding a special task into the system which checks the status of the operation each time this tasks executes. When the hardware operation is finished this device executor task will end. A special system facility allows a client task to wait for the device executor task to end while the rest of the task queue is still executing. In this way time intensive hardware operations can be performed by drive, count or scan commands without blocking the whole system for other clients.

Most experiments do not happen at ambient room conditions but require some special environment for the sample. Mostly this is temperature but it can also be magnetic of electric fields etc. Most of such devices can regulate themselves but the data acquisition program needs to monitor such devices. Within SICS this is done via a special system object, the environment monitor. A environment device, for example a temperature controller, registers it's presence with this object. Then an special system task will control this device when it is executing, check for possible out of range errors and initiates the proper error handling if such a problem is encountered.

5.2 The SICS Interpreter

When a task belonging to a client connection executes a command it will pass the command along with the connection object to the SICS interpreter. The SICS interpreter will then analyze the command and forward it to the appropriate SICS object in the object database for further action. The SICS interpreter is very much modeled after the Tcl interpreter as devised by John Ousterhout[1]. For each SICS object visible from the interpreter there is a wrapper function. Using the first word of the command as a key, the interpreter will locate the objects wrapper function. If such a function is found it is passed the command parameters, the interpreter object and the connection object for further processing. An interface exists to add and remove commands to this interpreter very easily. Thus the actual command list can be configured easily to match the instrument in question, sometimes even at run time. Given the closeness of the design of the SICS interpreter to the Tcl interpreter, the reader may not be surprised to learn that the SICS server incorporates Tcl as its internal macro language. The internal macro language may use Tcl commands as well as SICS commands.

5.3 SICS Objects

As already said, SICS objects implement the true functionality of SICS instrument control. All hardware, all commands and procedures, all data handling strategies are implemented as SICS objects. Hardware objects, for instance motors deserve some special attention. Such objects are divided into two objects in the SICS system: A logical hardware object and a driver object. The logical object is responsible for implementing all the nuts and bolts of the hardware device, whereas the driver defines a set of primitive operations on the device. The benefit of this scheme is twofold: switching to new hardware, for instance a new type of motor, just requires to incorporate a new driver into the system. Internally, independent from the actual hardware, all hardware object of the same type (for example motors) look the same and can be treated the same by higher level objects. No need to rewrite a scan command because a motor changed.

In order to live happily within the SICS system SICS object have to adhere to a system of protocols. There are protocols for:

- Input/Output to the client.
- Error handling.
- Interaction with the interpreter.
- For identification of the object to the system at run time.
- For interacting with hardware (see device executor above).
- For checking the authorisation of the client who wants to execute the command.

SICS uses NeXus[2], the upcoming standard for data exchange for neutron and X_ray scattering as its raw data format.

SICS objects have the ability to notify clients and other objects of internal state changes. For example when a motor is driven, the motor object can be configured to tell SICS clients or other SICS objects about his new position.

The SICS server has been implemented in ANSI-C for maximum portability. ANSI-C proved powerful enough to support SICS's object oriented programming concepts.

6. SICS Working Example

In order to illustrate the inner working of the SICS server a command to drive a motor will be traced through the system.

- The network reader finds data pending at one of the client ports.
- The network reader reads the command, splits it into single lines and put those on the top of the client connections command stack. The network reader passes control to the task switcher.
- In due time the client connection task executes, inspects its command stack, pops the command pending and forwards it together with a pointer to itself to the SICS interpreter.
- The SICS interpreter inspects the first word of the command. Using this key the interpreter finds the drive

command wrapper function and passes control to that function.

- The drive command wrapper function will check further arguments, checks the clients authorisation if appropriate for the action requested. Depending on the checks, the wrapper function will create an error message or do its work.
- Assuming everything is OK, the motor is located in the system.
- The drive command wrapper function asks the device executor to run the motor.
- The device executor verifies that nobody else is driving, then starts the motor and grabs hardware control. The device executor also starts a task monitoring the activity of the motor.
- The drive command wrapper function now enters a wait state. This means the task switcher will execute other tasks, except the connection task requesting the wait state. The client connection and task executing the drive command will not be able to process further commands.
- The device executor task will keep on monitoring the progress of the motor driving whenever the task switcher allows it to execute.
- In due time the device executor task will find that the motor finished driving. The task will then die. The clients grab of the hardware driving permission will be released.
- At this stage the drive command wrapper function will awake and continue execution. This means inspecting errors and reporting to the client how things worked out.
- This done, control passes back through the interpreter and the connection task to the task switcher. The client connection is free to execute other commands.
- The next task executes.

7. Current Status

Currently, SICS is in operation on the powder diffractometer DMC, the two circle diffractometer TOPSI and the small angle scattering diffractometer SANS. First clients have been implemented in Tcl/TK[1] and ANSI-C for a command line interface and status displays. First experiences with the system were positive. SICS currently supports the following hardware:

- SINQ EL734 motor controller.
- SINQ EL737 counter box.
- SINQ Histogram Memory.
- Dornier Velocity Selector.
- Oxford Instruments ITC4 temperature controller.

SICS is in ongoing development. Support for the reactor time-of-flight instrument FOCUS, the reflectometer AMOR and the four circle diffractometer TRICS will be added in near future. Support for chopper control and more sample environment devices is on the way. More clients will be written in the Java programming language for optimal portability. Parties interested in using SICS or wishing to collaborate in its development are kindly asked to contact the authors.

8. References

1. John K Ousterhout: Tcl and the Tk toolkit, Addison-Wesley, 1994.