Slow Controls for Experiments in Nuclear Physics

R. Fox[1]

Abstract--**The NSCL is a national nuclear physics research facility. Modern experiments in nuclear physics feature an increasing number and type of remotely controlled electronics including detector bias power supplies, discriminators, amplifiers and gas handling systems. These devices in turn connect to the computers controlling them via a wide variety of interfaces including Ethernet, CAN, VME, and even the venerable CAMAC.**

This paper will describe an open, extensible control system for these sorts of devices that uses a pair of Metakit databases in an attempt to impose some sort of order on this chaos. The system described allows system integrators to rapidly add support for new devices and even new interface subsystems. Experimenters, using a simple GUI, can describe the devices in their experiment. This description of the experiment drives the automated generation of a control panel for the devices they use.

I. Background and Goals

The National Superconducting Cyclotron Laboratory (NSCL) is an NSF funded national laboratory dedicated to basic research in heavy ion nuclear physics. The NSCL is the world's premiere facility for studies of nuclei far from stability, and a leading candidate for the location of the Rare Isotope Accelerator project. Research at the NSCL spans a wide variety of topics including nuclear structure, spectroscopy, nuclear statistical mechanics nuclear astrophysics, and even nanotechnology. Additionally the Single Event Effect (SEE) beam line funded by the National Aeronautics and Space Administration (NASA) allows researches to expose prototypes of satellite instrumentation systems to radiation fields that can be expected in-flight. The NSCL serves a user community spread out across the U.S. and has also hosted experimenters from the European, Asian and other international collaborations.

The preparation time-scale of typical NSCL experiments is short relative to that of the high energy physics experiments for which most software toolkits have been built. Where an experiment such as ATLAS at the LHC has had over a decade to prepare the software and hardware packages prior to its first beam-time, the turnaround from proposal to experiment at the NSCL can be under a year. While collaborations preparing large scale high energy physics experiments can take the time to integrate sets of loose software toolkits into finely tuned experimental support software, NSCL users must have access to software tools that require very little integration. Preferably software components that approach the state and quality of a finished application.

In addition to the digitization hardware described in a paper presented at tcl2004[1], experimenters must have accurate and simple control over a variety of programmable electronics. The settings for these programmable electronics are usually tuned prior to production running, and remain stable over the course of several runs within an experiment. These settings comprise an important set of parameters that partially describe the conditions of a data taking run. Control systems for these sorts of electronics devices are traditionally called *slow control* systems to distinguish their relatively lax timing requirements from those of the main experimental data flow.

This paper describes a development initiative to produce an open framework for slow control that approaches the readiness and quality of a finished program. The goals of this initiative were to prototype a system that:

1. Is open and extensible
2. Allows experimenters to quickly define their electronics without detailed knowledge of how modules are controlled
3. Supports the generation of a control panel for the entire slow controls system of an experiment from the experimenters description of the electronics
4. Supports the ability to record and restore the settings of sections of or the entire control system.

Subsequent sections of this paper describe the data model on which the software was built, and how it supports the necessary extensibility of the system. We describe the roles a user may take when interacting with the system and the expectations for each role. The software structure is described along with the applications that have been built to support each anticipated user role. I will wrap up the paper with a

[1]Ron Fox is with the National Superconducting Cyclotron Laboratory at Michigan State University in East Lansing, MI

summary of the status of the software and an evaluation of the degree to which the effort has been successful.

II. Data Model

The slow control system is driven by a pair of Metakit[2] databases. The first of these describes the set of supported hardware, while the second describes the experiment itself.

We call the first of these databases the "system" database, and the second one the "experiment" database. The system database lives in a centralized location. One instance of the system database exists so that changes to this database are picked up by all applications next time they are run. The system database changes infrequently over the lifetime of several experiments.

The experiment database describes the set of modules used by the experiment, and contains saved settings created throughout the lifetime of the experiment. An instance of the experiment database (usually located in the experiment's account home directory tree) exists for each experiment. The configuration part of the experiment database may change several times during the planning and initial test runs of an experiment. The settings part of the experimental database will be somewhat fluid during the initial runs, as the electronics are tuned to maximize the detector systems, but tends to be relatively stable once an experiment enters production mode.

Subsection A will describe the system database. Subsection B will describe the experiment database. Finally, Subsection C will show how a bus and a module type in that bus are represented as well as how that module type is represented as configured into an experiment.

A. *The system database*

Devices supported by the slow control system can connect to host computers in a wide variety of ways ranging from classic instrumentation busses to Ethernet or CAN[3] networks. Each supported module must be described in a way that makes it possible to determine solely from the information in that database:

- The set of supported instrumentation busses
- How addressing on each instrumentation bus works.
- For each supported module the instrumentation bus it lives in.
- For each supported module how the module is to be controlled both by the user and by the software.

The data model for the system database is shown in Figure 1. The **Module Types** table contains one entry for each supported electronic module. Each module is uniquely identified by an integer key, has a descriptive name, and an integer code that identifies the instrumentation bus in which the module lives. The module descriptive name is used to generate the name of a Tcl package that knows how to control the module. The bus type code is used to join module to a record in the **Instrumentation Busses** table.

The **Instrumentation Busses** table defines the set of instrumentation busses that are known to the system. Each instrumentation bus is given a unique integer code and a meaningful name. The code is used not only to join a supported module to the bus it supports, but also to join the bus record to records in the **Bus Address Fields** table that describe the addressing of each bus.

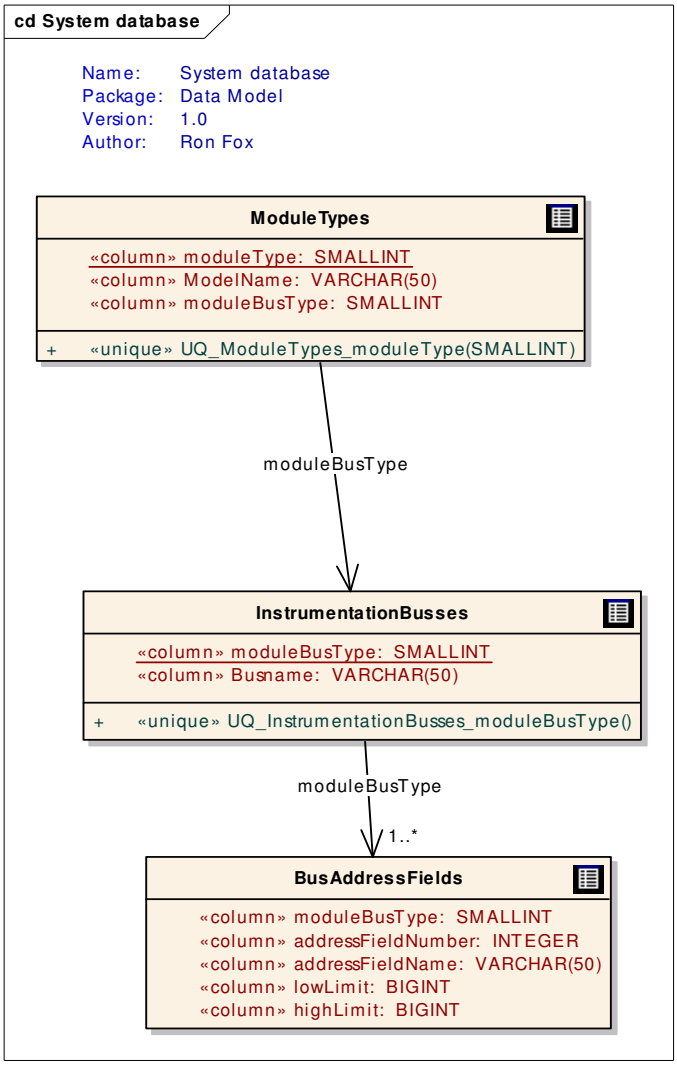

Figure 1 Data model of the system database

The **Bus Address Fields** table is used to describe how addressing on a bus works. Each bus may have any number of address fields. In a CAMAC system, for example one will typically address a module by three fields, Branch, Crate, and slot. On the VME bus, a single base address field, and an

associated address space (address modifier) code addresses a module, while a high voltage power supply with an Ethernet interface may be identified by the four octets of its IP address. Each record of the **Bus Address Fields** table describes a single bus address field and contains:

1. The bus type code (moduleBusType) identifying which bus the record belongs to.
2. The address field number, which provides field ordering.
3. The name of the address field, which identifies the field to the human users of the system.
4. Low and high limits on the values of the address field (for example a CAMAC crate in a parallel branch highway system may have a low limit of 1 and a high limit of 7).

B. The Experiment Database

The experiment database is a set of tables that describe the current experiment setup. The experiment database also stores saved settings created by the experiment control panel. The data model for the experiment database is shown in Figure 2.

The starting point for the experiment configuration is the **Configuration** table. Each entry in this table represents a module is being used in the experiment. The moduleType field is an integer code that joins this module to a module type in the **Module Types** table in the system database. The moduleId field is a unique identifier for that module within the experiment, and is used to join modules across the other tables in the experiment database. The moduleName is a human readable name chosen by the experimenter to identify this module within the experiment.

The **moduleAddresses** table contains actual addressing information for each module in the **Configuration** table. The records in **moduleAddresses** contain the address values assigned the module for each ordered field in the bus in which the module lives. These addresses are assigned to the module by the experimenter when configuring the system. They are used by the module's physical driver to establish a connection to the hardware.

In the course of running an experiment, the user may decide the slow control system is in a state that is worth saving for later re-use. The bottom set of tables in Figure 2 are responsible for maintaining these settings.

The main problem I had to solve for the settings database, was how to describe the settings associated with modules in an extensible system. The slow control system can control high voltage controllers, discriminators, amplifiers, and modules that I have not yet seen and know nothing about. A fixed set of setting attributes would never be workable, or extensible. Instead, I chose to use *property list*s to store setting information. A property list is a set of name/value pairs. In Tcl, a property list is easily represented by a list of 2 element sub lists. For example:

```
{{hv1 100} {hv2 205} {ilimit 200} ...}
```

Is a property list that might describe a 2 channel ISEG VME[4] detector bias supply, that is set with channel 1 at 100 volts, channel 2 at 205 volts, and with a global current limit trip point set at 200mA and "..." means that there are further properties not shown in this example.

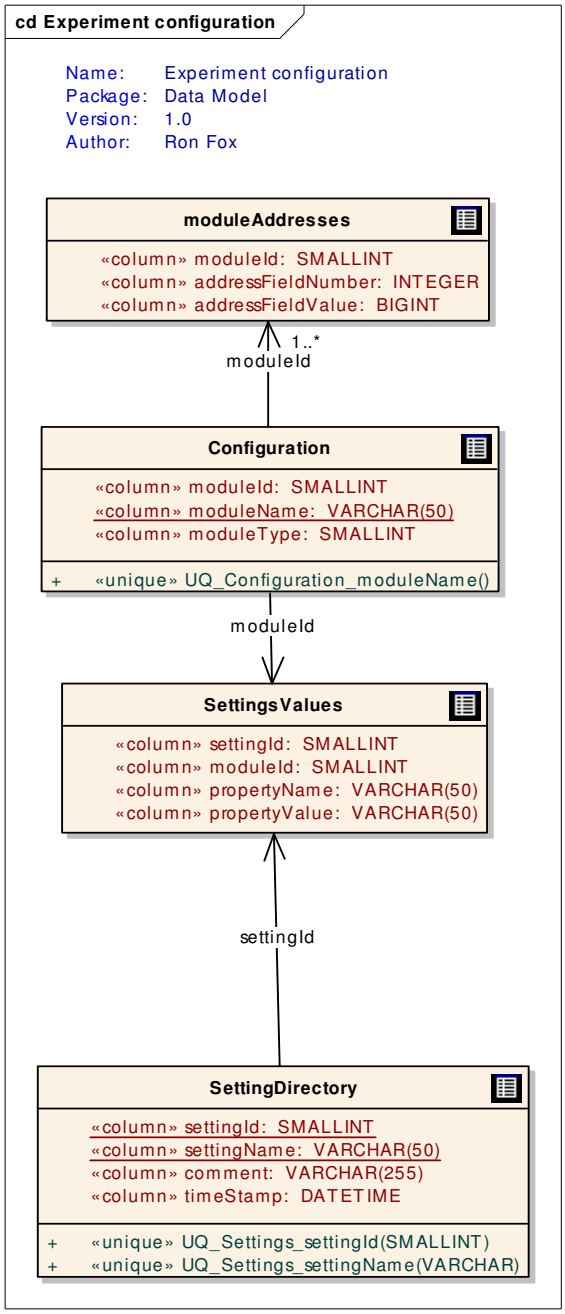

Figure 2 The experiment database

The user will want to be able to identify, and select the set of property lists that make up a useful set of saved settings . This function is provided by the **SettingDirectory** table. This table contains, for each set of saved settings:

1. A user supplied name for the setting.
2. A user supplied optional comment.
3. A system provided timestamp indicating when the `1setting was created.
4. A system provided unique setting id used to join the **SettingDirectory** entry to associated property list elements in the **SettingsValues** table.

The **SettingsValues** table contains the property list elements for all of the settings. Each record in this table contains:

1. A setting id that corresponds to a unique setting id in the **SettingDirectory** table identifying which setting this element belongs to.
2. A moduleId that corresponds to a unique moduleId in the **Configuration** table, that identifies which module's property list this element is a member of.
3. The name of the property
4. The value for the property.

C. Database Examples

This section provides the following examples of database content:

1. The description of the CAMAC parallel branch highway subsystem as it might appear in the system database.
2. The description of a CAEN C805[5] constant fraction discriminator as it could appear in the system database.
3. The configuration of a CAEN C805 constant fraction discriminator as it could appear in the experiment database.

1) Describing the CAMAC parallel branch highway system

The CAMAC parallel branch highway system is an instrumentation bus and interconnection system that were designed for large CAMAC based setups. The specification defines a CAMAC system as consisting of one or more *branches.* Each branch has up to 7 instrumentation chassis called *crates* numbered 1-7. Each crate has 23 usable slots and modules are geographically addressed within the crate. One CAMAC[6] interface used at the NSCL is the CES CBD 8210[7]. The 8210 is a VME based branch highway controller that
supports up to 8 branches per VME crate. The NSCL Data Acquisition system supports up to 8 VME crates per host. The branch highway driver base addresses are determined by the branch number set in each controllers configuration switches.

To summarize, addressing an NSCL CAMAC module requires the following address fields:

Field name	Low limit	High limit
Vmecrate	0	7
Branch	0	7
Crate	1	7
Slot	1	23

Table 1 Address fields for a CAMAC branch highway system

Suppose the branch highway system was allocated bus id 0. The **Instrumentation Busses** table entry for the CAMAC branch highway system might contain:

Field	Contents
moduleBusType	0
Busname	CAMAC Branch highway

Table 2 Instrumentation Busses record for CAMAC Branch highway

There would be four records in the **BusAddressFields** table that describe this bus:

bustype	fieldno	name	lowlimit	hilimit
0	0	VmeCrate	0	7
0	1	Branch	0	7
0	2	CamacCrate	1	7
0	3	Slot	1	23

Table 3 Fields of a parallel branch highway

2) The CAEN C805 in the system database

The CAEN C805 is a 16 channel constant fraction discriminator housed in a single width CAMAC module. Continuing the previous example. If this module had been allocated module Id 12, it would have the following record in the **ModuleTypes** table of the system database:

Field Name	Contents
moduleType	12
ModelName	CAENC805
moduleBusType	0 (CAMAC Branch Highway)

Table 4 CAEN C805 in the system database

3) A C805 constant fraction in an experiment.

Suppose an experimenter has decided to use a CAEN C805 described as in the previous two examples. She decides to call the module *detector hits* and installs it in slot 4 of CAMAC crate 1 in branch 0 of VME crate 0. If the experiment database has allocated module id 1 to this module, the **Configuration** table record for this module will contain:

Field	Contents
moduleId	1
moduleName	detector hits
moduleType	12 (CAENC805 module type)

Table 5 Configuration table entry for a CAEN C805

The **moduleAddresses** table of the experiment database will contain four records for this module which will contain:

moduleId	addressFieldnumber	addressFieldValue
1 (detector hits)	0 (Vmecrate)	0
1	1 (Branch)	0
1	2 (CamacCrate)	1
1	3 (Slot)	4

Table 6 module Addresses of a CAEN C805

III. User roles

The slow controls system software has been defined with three user roles in mind. A user fulfilling each role has specific functions that he or she is expected to perform. These roles, in order of decreasing sophistication are:

- System integrator
- Experiment designer
- Shift operator

A. *The system integrator*

The system integrator is the only role that requires programming knowledge. The system integrator is responsible for ensuring that the required modules have drivers and that they and the busses they live in are correctly described in the system database. To support a new module type, the system integrator must:

- Create a driver package for the module and install it where it will be found by Tcl automatic package loader. Typically this driver will be implemented as a pair of packages, a GUI package which is responsible for managing an instance of the user interface for the module in a frame, and a hardware interface package that is responsible for abstracting communication with hardware.
- If the bus in which the module lives is not yet in the system database, the system integrator must describe the bus, its address fields and the limits on each field.
- The system integrator must add an entry to the module types table of the system database.

The system integrator has a pair of applications that she can use to edit the system database. These applications are bus and module editors.

The bus editor allows the user to create a new bus or to edit an existing bus. The screen shot in Figure 3 , shows how the description of the CAMAC Branch highway might look when being edited. Clicking the **New** button creates a new address field which the user can fill in and **Accept**. Clicking on a line of the field table loads its definition into the form allowing it to be modified. Finally, the selected table line can be moved up or down in the field order using the arrow buttons to the right of the table. Once the fields are defined as desired, the **Ok** button saves the bus definition in the database. The **Cancel** button exits the editor without making any database modifications.

The module definition editor is shown in Figure 4. The table at the top of the GUI lists the set of currently supported module types. This table grows scrollbars if needed. Creating a new module is a matter of clicking the **New** button, typing the module type name in the Type entry, and selecting a bus from the Bus drop down list. If a module has been described as living in the wrong bus type, clicking its line in the table

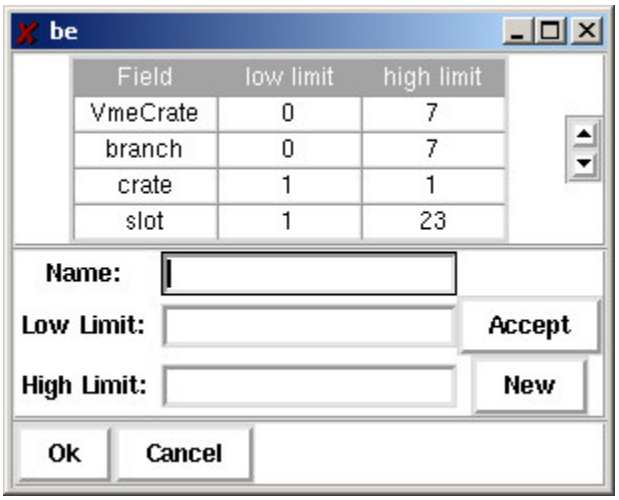

Figure 3 Editing a bus address definition

loads its entry into the middle form where a new bus can be selected from the drop down list. The **OK** button saves all database changes, while the **Cancel** button exits and leaves the database unmodified.

B. The Experiment Designer

The experiment designer is responsible for the electronics design of the experiment. From the point of view of the slow controls system, this means that she must fill in the **Configuration** and **moduleAddresses** table for each module that will be under the control of the system.

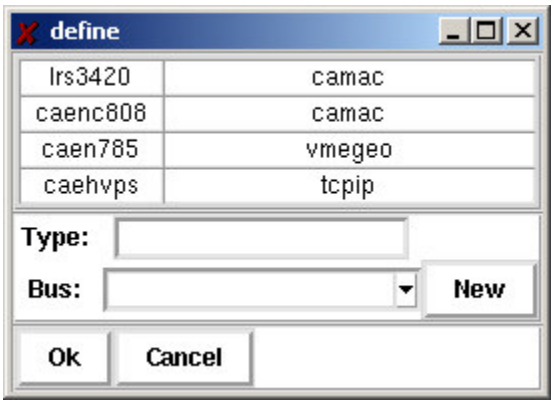

Figure 4 The module definition application

To do this, the experiment designer will interact with an experiment editor. The experiment editor allows the experiment designer to add, remove and edit modules that are in the experiment configuration. The experiment editor is shown in action in Figure 5.

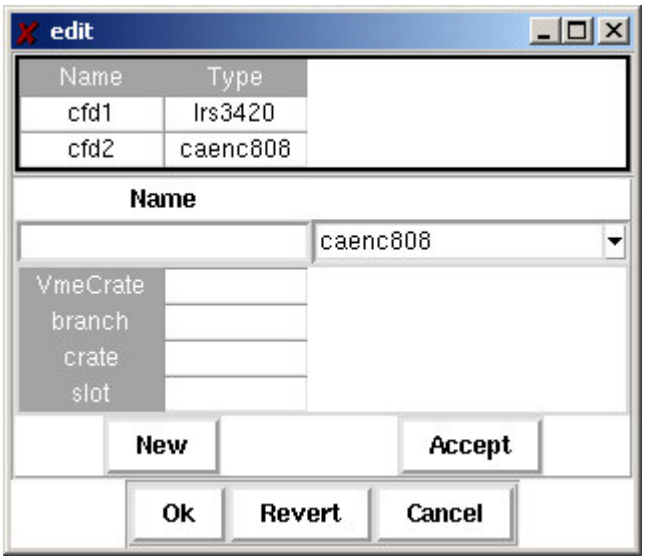

Figure 5 The experiment editor

The modules that are currently configured into the experiment are shown in the table at the top of the GUI. If a line of this table is clicked, the addressing information for the module is loaded into the lower part of the GUI. where it can be edited. Hitting the *Delete* key deletes the selected module. The **New** button allows the user to type a new module definition into the lower table. The module type is selected from the set of supported modules through the drop down list, and the name is typed in the name field. Once the module type is selected,

the appropriate address field names get loaded into the table, and the user can enter values for each field.

The **Accept** button either adds the new module or accepts an edit to an existing module. **Accept** also range checks each address field against the limits defined for that field in the system database, refusing to commit the change if an address field is invalid. Clicking on **Ok** writes the new experiment definition to the experiment database. Clicking on **Revert** reloads the top table from the database, and clicking on **Cancel** dismisses the editor without making any changes to the database.

C. *Shift operators*

Shift operators are the people who actually run the experiment. They will be operating the controls for the experimental electronics defined by the experiment designer. They interact with the electronics through a control panel application.

.When the control panel starts, it reads the experiment database to determine the set of modules, their types and locations in the bus. For each module, a page is created in a BLT[8] tabbed notebook widget. The page widget is handed to the driver for the module along with the bus addressing information. The driver is responsible for establishing communication with the specific device, a drawing the GUI for that device in the page it was given, and responding appropriately to events in the GUI. Existing pages are implemented as Snit[9] mega widgets.

A sample control panel page for the CAEN C808 discriminator is shown in Figure 6.

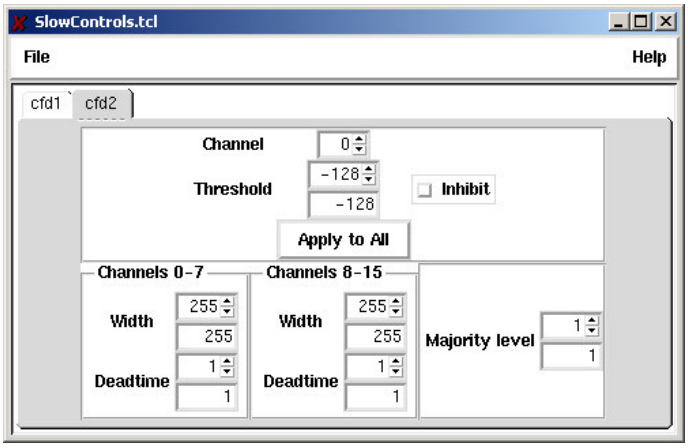

Figure 6 The control panel in action

The pages of the notebook are created with tear-off enabled in case the user wants to view multiple pages simultaneously.

The **File** menu provides access to the saved settings part of the experiment database. Figure 7 shows the dialog that pops up in response to the **File->Restore...** menu entry.

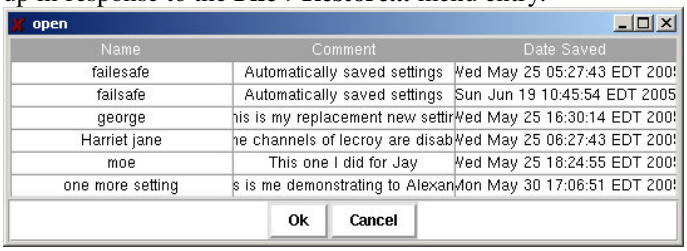

Figure 7 The Settings restoration dialog

The user selects a setting line and clicks **Ok** to load that setting. Note the timestamps and comment fields. The table will grow scroll bars as needed. The **File->Save...** menu entry brings up a similar dialog with the addition of a **New** Button that allows you to create a new setting, rather than overwriting an existing setting.

When settings are saved, the control panel simply asks each page driver to return its property list. These get tagged with the module and settings ids and written to the database. Similarly, when a setting is restored, the control panel application distributes the appropriate property list to each page driver and that driver interacts with the hardware so as to restore the settings described by the property list. Periodically, the control panel automatically saves settings in a "failsafe" setting allowing the user to return to a recent state in the event of system failure or power outage.

IV. Implementation

The system block diagram is shown in Figure 8.

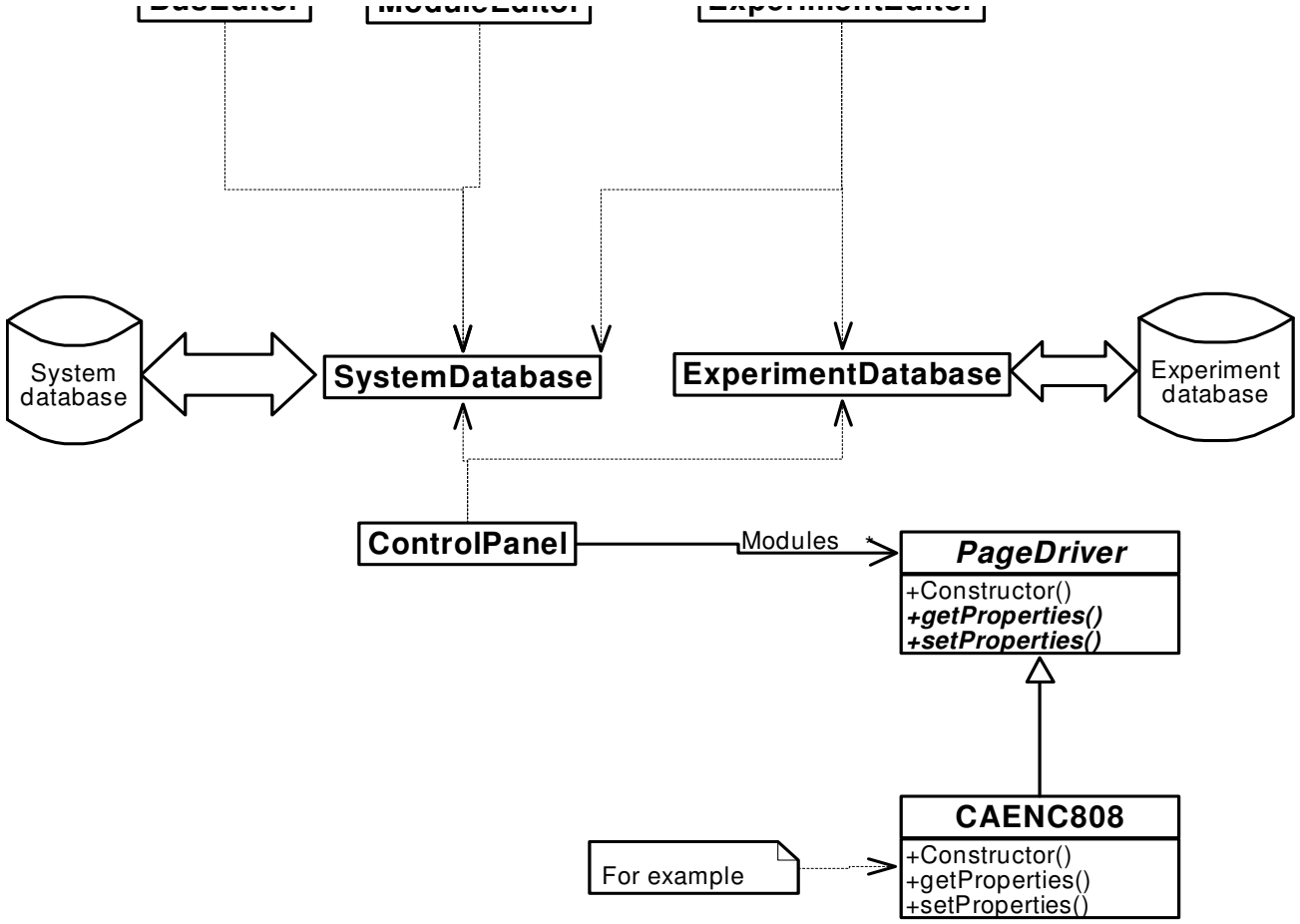

Figure 8 System block diagram

Heavy use was made of Snit types and Snit mega widgets. This is seen most clearly in the interface between the control panel and the Page drivers. Page drivers are Snit mega widgets, Page drivers must implement a constructor that recognizes an **–address** option that will contain the ordered address list, as well the widget path to the page in which they must draw their GUI. They are also responsible for implementing two methods in order to support the settings save/restore subsystem:

- getProperties must return a property list that will restore the device to its current setting.
- setProperties must accept a property list and restore the device to the state described by that list.

While figure 8 only shows a single concrete driver, additional drivers would be added by the system integrator to support other module types.

The other salient feature of figure 8 is the database isolation modules. All database accesses go through these modules insulating the system from changes in both database structure and database package choice.

The entire system consists of about 2800 lines of Tcl/Tk/Snit code including integration tests. Unit tests, for non-graphical modules make up another 1200 lines of tcltest code.

V. Status and conclusions

An implementation of this system is in use. Currently two discriminator modules are supported while the utility and usability of the software are being evaluated.

While it was relatively simple to use Metakit to implement the database, I wonder if an implementation on top of an SQL based system such as SQLite[11] might have made the management of joins between tables much simpler to handle. If the system transitions from its current prototypical implementation to a production implementation, the insulation of the applications from the database via the database adaptor modules should make such a switch relatively painless.

There are several shortcomings in system design that I would remedy in a reimplementation of the system:

- It might be well to formally recognize the split between physical device driver and page driver. This would allow system integrators to provide several alternative GUI's for a single device.
- Regardless, the driver package names should be in the database rather than derived from the module name.
- Some thought should be given to how to handle multiple interface types to the same bus system. For example, we have illustrated the use of CAMAC via a CES CBD 8210 VME branch highway system. Also supported by the NSCL are the Wiener VC32/CC32 VME[11] to CAMAC interface, and support has also been contributed for the Wiener PCDA-CC32[12] PCI to CAMAC interface. It is likely that support will be contributed to for the Wiener USB-CC USB[13] CAMAC interface as well. These all have slightly different addressing schemes, however sufficient insulation of the page driver from the underlying hardware driver should make it possible to re-use the control interface even if the underlying hardware driver for each of these is slightly different.
- The model of addressing via a set of numeric fields is somewhat simplistic. For example, a device that contains an Ethernet interface should be addressable via its DNS name. In this more open model of addressing, more complex validation is required, perhaps requiring a system integrator to either select validation scripts from a set of pre-written scripts or to supply a new script to support unforeseen validation needs.

Things that work well:

- The choice of a property list representation of device settings has made it quite easy to build the save and restore subsystem on top of modules with unknown parameter sets.
- The automatic derivation of the user interface from the experiment configuration database is very friendly to the user.
- The use of Snit mega widgets to produce the user interfaces of both the page drivers and the GUI's of the various configuration utilities simplified the user interface logic.
- While Metakit may not be the best choice of a database package, nonetheless, the limited set of queries required by the system meant that once these queries were written it nice to be able to rely on a debugged database package as a basis for implementing the data model of the system.

All in all this was an enjoyable project. It has been contributed to a group at Oregon State University which uses it as is in the experiment design and shift operator roles, and have been favorably impressed with the system's configurability and stability.

VI. References

[1] *NSCLSpecTcl Meeting the Needs of Preliminary Nuclear Physics Data Analysis.* Proceedings of Tcl 2004 New Orleans available at http://www.tcl.tk/community/tcl2004/Papers/RonFox/fox.pdf

[2] http://www.equi4.com/metakit.html

[3] CAN Specification 2.0 Robert Bosch gmbh available online at http://www.semiconductors.bosch.de/pdf/can2spec.pdf

[4] *Precision High Voltage Supply VME STANDARD series Operators Manual* iseg Spezialelektronik GmbH. Available online at: http://www.iseg-hv.com/download.php/483/file_url_en/vhqx0x_eng.pdf

[5] Technical Information Manual: Mod. C8080 16 Channel Constant Fraction Discriminator Costruzioni Apparechiature Elettroniche Nucleari (CAEN) Available online at http://www.caen.it/nuclear/product.php?mod=C808

[6] IEEE Std 583-1975 IEEE Standard Modular Instrumentation and Digital Interface System (CAMAC).

[7] Creative Electronics Systems CBD 8210 CAMAC Branch Driver CES Document CBD8210.ds Rev 0.01.

[8] http://www.sourceforge.net/projects/blt

[9] http://www.wjduquette.com/snit/

[10] http://www.sqlite.org

[11] VME – to – CAMAC CC32 CAMAC Crate Controller with VC32 VME Interface User Manual W-ie-Ne-R Plein-Baus gmbh available online at http://67.15.74.137/documents/contentdocuments/12.pdf

[12] PCI-CAMAC CC32 CAMAC Crate Controller with PCI Interface available online at Interface User Manual W-ie-Ne-R Plein-Baus gmbh available online at http://67.15.74.137/documents/contentdocuments/8.pdf

[13] CC-USB User Manual W-ie-Ne-R Plein-Baus gmbh available online at http://www.wiener-d.com/documents/contentdocuments/18.pdf