UPGRADING THE FERMILAB FIRE AND SECURITY REPORTING SYSTEM

C. King, R. Neswold
FNAL[†], Batavia, IL 60510, U.S.A.

Abstract

Fermilab's homegrown fire and security system (known as FIRUS) is highly reliable and has been used nearly thirty years. The system has gone through some minor upgrades, however, none of those changes made significant, visible changes. In this paper, we present a major overhaul to the system that is halfway complete. We discuss the use of Apple's OS X for the new GUI, upgrading the servers to use the Erlang programming language and allowing limited access for iOS and Android-based mobile devices.

INTRODUCTION

FIRUS is an acronym for **F**ire **I**ncident **R**eporting and **U**tility **S**ystem. It was developed at Fermilab in the early eighties using the technology available at the time resulting in consoles that run on MS-DOS based PCs using Digital Research's GEM graphical user interface and servers running Motorola's VersaDOS. Consoles communicate with servers via ARCNET and are segregated from Fermilab's main network for security reasons. This has resulted in a reliable, high-availability system that is still in use today.

OVERVIEW

FIRUS is comprised of front-ends, minis and consoles which all communicate via a private ARCNET network. The network is divided into eight trunks as shown in Figure 1. The front-ends are connected to all eight trunks, while the minis and consoles are only connected to one.

Front-Ends

At the heart of FIRUS are the two front-ends, "blue" and "red". The blue front-end is considered the primary and the red front-end is the backup. Under normal conditions, both blue and red divide the load of scanning minis and communicating with consoles equally. They each hold a full copy of the device database, which contains hardware addresses and alarm limits. Edits to the database go to blue, which then forwards the changes to red's copy.

FIRUS is designed to be fault-tolerant against a single front-end failure. The two systems monitor each other and, if one determines the other is unresponsive, it will assume responsibilities for all minis and consoles until the other front-end returns to service. This feature also allows us to take one system down for maintenance without impacting Fermilab's security and fire departments.

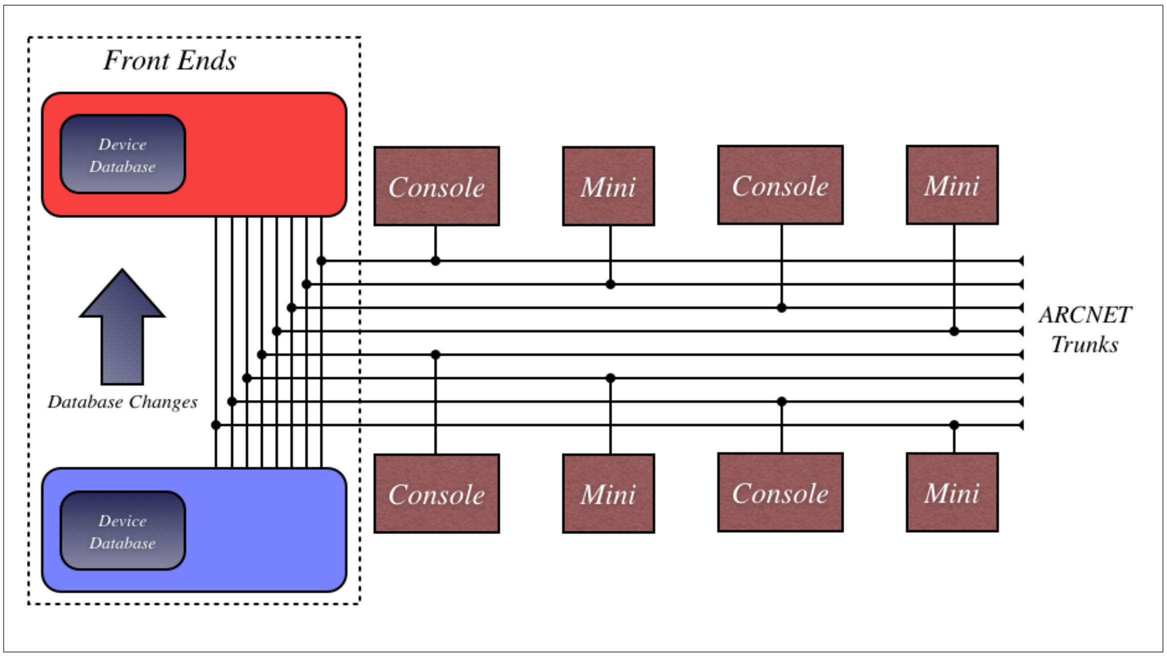

Figure 1: FIRUS Topology.

[†]Operated by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the United States Department of Energy.

Minis

"Minis" are small, embedded systems distributed site-wide. There are 150 minis currently active in FIRUS. Each mini is controlled by an 80186 embedded processor which uses an ARCNET interface to communicate with the front-end systems. A system can contain up to seven daughter cards each of which provides analog or digital inputs. The analog cards provide 16 channels of 16-bit A/D conversion while the digital cards supply 16 bits of input. Each digital input is instrumented to detect shorts, open circuits and grounded signals. Minis are embedded software modules that connect to device hardware.

Consoles

All user interaction with FIRUS happens on the console. The important features of a FIRUS console are as follows:

- Alarm acknowledge and display
- Alarm logging
- Device database management
- Real-time parameter page display
- Data logging at multiple rates
- Real-time and logger plotting
- Synoptic picture displays
- Fully configurable
- Password protection for sensitive items

UPGRADING THE CONSOLE

The FIRUS console hardware hasn't had a refresh since it's original inception in the mid eighties. We are now reaching the point of obsolescence, thus creating a nightmare maintenance scenario. GEM compatible PC hardware was becoming increasingly hard to come by. GEM requires EGA which is fairly low resolution by today's standards and only supports a limited sixteen colours. Contrast that with the ultra high resolution, million colour displays of today and it's not hard to see that change is needed.

The combined memory issues with MS-DOS and GEM was also a source of frustration. Adding features and making changes were almost impossible. Eventually the FIRUS application had to be split into two separate executables because of GEM's 64K maximum resource data segment size, resulting in a stripped down FIRUS application and a utility FIRUS application known as UFIRUS. Because MS-DOS is not a multitasking environment, this split basically removed some features from FIRUS as a whole since most users would never run UFIRUS.

Mac OS X

As it became more of a challenge to obtain replacement PC hardware, more time was invested in determining a new platform for the FIRUS console. Windows XP, Linux with QT, Java and OS X were all possible, but OS X won out because of it's modern graphical interface, it's Unix core and it's powerful set of **free** development tools[1].

OS X has the benefit of being a direct descendant of NeXT Inc.'s OPENSTEP operating system[2]. It inherited from OPENSTEP the feature-rich, object-oriented interface as well as the beginnings of a build environment that Apple has masterfully crafted into what is now known as Xcode. Xcode presented a learning curve but proved to be well worth the effort. It was clear that not only was Apple catering to the end user, they were also making OS X attractive to the developer.

Design Goals

FIRUS has been in use by Fermilab since the early eighties, hence our most important design goal was to minimize the user's learning curve. User acceptance was very integral to the success of the upgrade. Most people are annoyed and/or frightened by change, especially with changes without fundamental improvement to the workflow. Obviously changing the graphical user interface poses some changes that may require the users some time to adjust, but overall we strove to minimize most changes unless deemed value added.

Other design goals included:

- No changes to front-end code so both GEM and new consoles could run side by side for verification purposes
- FIRUS will be one cohesive application (GEM was two separate applications)
- Add a kiosk mode to keep unprivileged users from switching away or quitting the FIRUS application
- Automatic software update distribution with versioning

ARCNET Challenge

After deciding on Mac OS X, iMacs and Mac minis were chosen as the hardware we would use to replace the current consoles. Both the iMac and Mac mini are the least expensive of the Macintosh line, yet were powerful enough for our needs. They also both have a very small footprint and are able to physically fit where any GEM console currently resides.

One of the more challenging aspects of the FIRUS console upgrade was how to get the desired Macintosh computers connected to ARCNET. Since iMacs and Mac minis were the chosen hardware, ARCNET cards were not an option. The search was on for a device that would bridge the gap. Our search turned up such a device. The USB22-CXB from Contemporary CONTROLS is an external USB to ARCNET bridge that is powered by the USB port thus eliminating the need for separate power, and it connected to directly to our coaxial-based ARCNET topology.

Alas, our excitement was short lived when we discovered that the USB22 was virtually unsupported on OS X. Contemporary CONTROLS had no plans to add support, so if we wanted to use the device we would have to do it ourselves. Our first thought was to make the USB22 a true network device in order take advantage of built in network configuration in OS X. While this approach worked well (we were able to copy large files, stream video and browse web pages over ARCNET), it was not compatible with the FIRUS minis. The minis were unable to ignore the multicasted TCP/IP traffic and would become unresponsive. Since updating the software in the FIRUS minis was out of the scope of the upgrade, we had to abandon the network driver for a more FIRUS-specific solution. The final decision was made to use OS X's application level USB driver interface. Code for supporting the USB22 is built in to the FIRUS consoles, and as a result, the application can notify the user of network or device problems in a more timely manner.

Development Notes

Overall, the development of the new FIRUS console was an exciting project to work on. The Xcode integrated environment make easy work of designing the user interface elements. The source code editor aided code production by providing context sensitive help and autocompletion. The debugger worked flawlessly, and the instrument tools for memory issues assured that the high -availability goal for the console could be attained.

Xcode, however, wasn't our only learning curve. We also received lessons on usability. Just because certain standard features of OS X are indispensable in some applications, doesn't mean they will automatically be well received in the FIRUS console. One specific incident pertained to the basic alarm screen. We assumed that giving the users ad-hoc sorting by any displayed column would be a feature they couldn't live without. As it turned out, that was one feature they couldn't live with. FIRUS operators were so used to alarms lists that are sorted by decreasing time that they didn't want to even have the ability to change it and possibly misread the alarm screen. From that point on, we became more FIRUS user- centric in our user interface design.

FUTURE DEVELOPMENT

Now that the FIRUS console upgrade is complete and in the process of certification, we look toward future development for FIRUS. During development of the FIRUS console upgrade we discovered a number of front-end deficiencies as well as inaccurate behaviour.

Front-End Upgrades

In the lifetime of the FIRUS system, the front-ends went through one major upgrade. Right before Y2K, we upgraded the hardware from a 68K-based VME board to a multi-gigahertz Intel-based PC. The FIRUS software was ported -- not rewritten -- from Motorola Unix to a BSD Unix. The upgrade gave us a lot of CPU horsepower and much more memory to use than the original system. None of these extra resources have been exploited, yet. Now that the consoles have been updated, we look to modernize the front-ends again.

The software in the front-end was written in a time of limited CPU resources and memory constraints. The operating systems didn't support threads and the FIRUS processes weren't written to fork parallel processes to handle multiple requests. We feel that, with the resources available on the new consoles, a greater number of requests will be placed on the front-ends. They need to scale gracefully with the extra load, which the single threaded design won't be able to do.

Another shortcoming is that the tasks on the front-end don't communicate with each other when some global, system state has changed. The database task, for instance, doesn't inform the alarm task that an entry has changed. Instead, a console will see the change when the alarm task eventually rereads the database. It would be nice if state changes, like this, are synchronized better within the front-end.

Our decision is to rewrite the front-end software using the soft real-time, functional programming language, Erlang. Erlang was developed by Ericsson to use in their telephony equipment[3]. The language features concurrency primitives as part of the language and uses very light-weight processes to break a problem into simple pieces. Processes communicate with each other using message queues. The Erlang runtime is very rich and includes an ACID-compliant, distributed database (which will get leveraged in the upgrade.) There is also a web server module so the front-ends could generate and deliver web pages displaying status.

Remote Access/Mobile Devices

In today's well-connected world, WIFI and cellular data access are the norm. People have come to expect universal access to their data. Now the question arises. How do we securely deliver remote access to FIRUS.

We are currently testing a web-based remote alarm display that gives users access both on and off site. This is done securely by routing all traffic though a proxy server using a secure socket layer (SSL). All on-site access is automatically allowed. Off-site access is allowed only after password verification. This is currently in use by our on-site fire technicians. When testing contact circuitry, they are able to view alarm status on the web browser running on their portable scanning device. This has been very successful by reducing the testing time in the field.

We believe the future of the FIRUS console could include application development on mobile devices. Today's mobile devices are essentially ultra portable computers capable of running complex applications. With proper security considerations, we feel that many of the console features could be incorporated into applications for the iPhone and iPad and possibly some Android-based devices.

CONCLUSION

FIRUS has been a reliable, high-availability system that has been in use since the mid eighties at Fermilab. But over the last few years, FIRUS started showing signs of it ageing architecture. In this paper, we presented the highlights of the FIRUS console upgrade as well as detailed what remains to be done to take FIRUS into the future.

REFERENCES

[1] “iOS Dev Center – Apple Developer“, 2011, Apple Inc. http://developer.apple.com/devcenter/ios/index.action

[2] “Erlang Programming Language”, 2011, Ericsson AB. http://www.erlang.org/

[3] “Mac OS X – Wikipedia, the free encyclopedia”, 2011, http://en.wikipedia.org/wiki/OS_X