LESSONS LEARNED ENHANCING EPICS CA FOR LANSCE TIMED AND FLAVORED DATA*

J. Hill, LANL, Los Alamos, NM 87544, U.S.A.

Abstract

A previous paper [1] described an upgrade to EPICS enabling client side tools at LANSCE to receive subscription updates filtered selectively to match a logical configuration of LANSCE beam gates, as configured by the control room. The upgrade required fundamental changes in the EPICS core components. First, the event queue in the EPICS server was upgraded to buffer record (function block) and device specific parameters accessed generically via software interfaces for introspection of 3rd party data. In contrast, event queues in previous versions of EPICS were strictly limited to buffering only value, timestamp, and alarm status tuples. Second, the Channel Access server is being upgraded to filter subscription updates. In this follow on paper some necessary design changes mid-project and the lessons learned during the software development will be described.

LANSCE

The Los Alamos Meson Physics Facility (LAMPF) was originally designed to be a versatile machine for medium-energy (800 MeV) nuclear physics experiments. It had three injectors and could simultaneously accelerate positive hydrogen ions (H+), negative hydrogen ions (H-) and polarized negative hydrogen ions (P-). These three beams could all have different intensities, duty factors, and even different energies - depending on experimental needs. Today LANSCE can simultaneously generate four H- beam types and two H+ beam types. It services several experimental facilities including a proton storage ring, a low-intensity neutron research facility, proton radiography, ultra-cold neutron source, isotope production, and a proposed materials test station.

Developed during the infancy of computer control systems, the architecture of the original LAMPF / LANSCE control system (LCS) has elements of data acquisition along with elements of traditional computer control system architectures. It employs a locally designed centralized hardware IO system called RICE (Remote Instrumentation and Control Equipment). One of the more interesting and useful features of RICE is its ability to do "Timed" and "Flavoured" reads.

A "Timed Read" refers to sampling the signal at any point within the 8.2 millisecond machine cycle. The time to sample is normally specified relative to the start, middle, or end of a particular beam gate, with the default being the start of the cycle.

A "Flavoured Read" refers to the ability to schedule the read for a particular machine cycle containing a desired configuration of beam-gates. A “Flavour” is configured by specifying for each of 96 timing system beam gates whether it must be present, must be absent, or is not relevant. Therefore, there can be up to 3^{96} possible flavor combinations, but in practice only roughly a dozen “Flavours” are regularly used. These represent various (meaningful) combinations of the six beam destination beam-gates along with a handful of diagnostic-trigger-gates, but more esoteric flavours for diagnostic and experimental purposes are considered to be essential.

Work supported by US Department of Energy under contract DE-AC52-06NA25396.

EPICS CONTROL SYSTEM

An EPICS Input Output Controller (IOC) is configured with Database Records implementing function blocks for various purposes including logical IO, numerical calculation, and ordered sequencing. The EPICS Channel Access (CA) internet communication subsystem is based on a publish-and-subscribe communication model where clients subscribe for updates, servers publish updates to subscribed clients, and records post state change events to servers. A channel is a virtual communication link between a client application program and a process variable (PV) exported by a service. EPICS clients issue asynchronous read, write, and subscribe requests to the process variable in the service. Clients are notified when the connectivity of a channel changes.

One of the most essential requirements underlying the original EPICS design was that regular periodic processing of EPICS Records should not be disturbed by influences from outside of an IOC. This guarantees that time periodic algorithms such as PID loops are properly maintained, and that there will be proper time deterministic response by EPICS Records to state changes detected in the sensors. This design recognizes that the load induced by Record processing is measurable when the IOC starts up, and remains fixed thereafter. In contrast, the externally induced load on the CA server by its clients is less predictable. It is therefore necessary for EPICS Record processing to execute at relatively higher priorities and for the CA Server to execute at relatively lower priorities. An event queue containing subscription updates communicates between the two entities.

UPGRADE REQUIREMENTS

At LANSCE we would like to upgrade the RICE based hardware components, of the control system with modern superset capability hardware. To interface the new hardware with EPICS we must consider how to preserve capabilities to view data in the control room selectively based on timing and flavouring parameters. Tuning operation work flows are typically experimental in nature where there are too many beam flavouring and timing permutations for all of the necessary flavours to be preconfigured as records in the EPICS Input Output Controller (IOC) *a-priori*. Therefore, it is necessary to upgrade EPICS at the Channel Access (CA) protocol level so that logical configurations of beam gates, and the necessary timing parameters, can be specified when a client subscribes for process variable (PV) updates.

An EPICS IOC must also be upgraded to allow site specific companion data to be specified when posting subscription updates. For LANSCE purposes the site specific companion data will be the set of beam gates present when the data were captured.

Furthermore, it is necessary to add new array indexing metadata to EPICS. When connecting with a PV the client must learn the magnitude of the zero-eth element, the magnitude of a one element increment and the units for these magnitudes. Time delay units will be typical.

All of the above changes must be accomplished without impacting the original guarantee; periodic processing of EPICS Records will be immune to external influences.

BASIC DESIGN

The software upgrade for the event queue has closely followed a design described in the previous paper [1]. In summary, it was necessary to upgrade the IOC's event queues to carry device and record specific payloads. Each event queue entry contains one smart pointer [2] linking to a call back method, a C++ member function, to be invoked when the event is processed, and one smart pointer linking to the device or record specific payload. The core software components in the system remain generic because device payloads are queried through a data introspecting interface we call Data Access [3].

Efficient memory management for these payloads is best delegated to EPICS record and device specific 3rd party modules that produce them. Therefore, reference-counting smart pointers are necessary so that when the last reference to a payload is consumed from the event queue proper cleanup can be efficiently delegated to the device and record specific code of the producer.

DESIGN CHANGES

Early in the design it was clear that the payload producing record and device specific codes must operate in different mutual exclusion locking domains from the event consumer. Therefore, a record and device specific lock may need to be taken before interrogating the data using the Data Access interface, and before invoking the call back methods which will process the event. What wasn't clear was the proper approach to manage this complexity in generic code. Initial efforts were similar to Figure 1 below – where a Guard object acquires the mutual exclusion lock in its constructor, and releases it in its destructor. Passing a reference to the Guard guarantees that the target object is protected.

```
Mutex mutex;
Guard guard (mutex );
Service & service = createService (guard);
Channel & channel = Service.createChannel (guard,…);
```

Figure 1: Initial mutual exclusion enforcement.

The downfall of the approach in Figure 1 is inability for the generic code to know which mutual exclusion locking primitive should be used to protect code in 3rd party modules. To eliminate this issue, and also unsightly clutter in the public callable interfaces, the project has transitioned to the approach in Figure 2. With this approach the C++ `operator->` is overloaded to return an upgraded type of guarded smart pointer object acquiring the mutual exclusion lock in its constructor and releasing it in its destructor. The approach in Figure 2 is a substantially better design because, we will see later, mutual exclusion locking and reference counting are delegated to the record and device specific 3rd party codes.

```
Ptr < Service > pService = createService ();
Ptr < Channel > pChannel = pService->createChannel (…);
```

Figure 2: Subsequent mutual exclusion enforcement.

Smart Pointer Choices

A wide range of custom smart pointer implementations are in use within the C++ community, but there are also some carefully crafted versions available in open-source libraries. In particular, the boost [4] library `shared_ptr`, and the loki [5] policy-based smart pointer are close matches to requirements. After some deliberation the boost `shared_ptr` wasn't adopted because of lack of automated mutual exclusion for each pointer invocation. The ultimate efficiency and flexibility of the policy-based smart pointer in the Loki library was also not adopted because it is taxing the limits of C++ compiler standard compliance. Unfortunately, one of the primary limiting factors being portability to the aged gcc 2.x cross compiler still used for building legacy embedded targets at several sites.

In the end, after careful research an in-house smart pointer design which exactly fits our needs was chosen. This has increased our code size by about 500 lines. There were some basic requirements; the implementations of mutual exclusion and reference counting must be runtime polymorphic. Therefore, the implementation of these features is delegated to the producer of the smart pointer target, the consumer of these smart pointer targets shall not be dependent on which implementation was chosen for a particular target, and the choice can be made after the base components are compiled. Furthermore, the choice of an efficient intrusive (in target object), or less efficient but architecturally decoupled non-intrusive implementation is also enforced to be runtime polymorphic. The design has also determined that (unlocked) smart pointers shall consume the same space as ordinary pointers, and therefore each (unlocked) smart pointer contains a reference to the abstract handle interface seen in Figure 3.

```
class HandleIntf {
    virtual TargetMutexPair targetMutexPair () = 0;
    virtual HandleIntf & clone () = 0;
    virtual void release () throw () = 0;
};
```

Figure 3: Smart pointer target handle.

To assist with efficient implementation of reference counting, new Symmetric Multiprocessing (SMP) safe atomic increment and decrement operators have been added to the EPICS base distribution. Implementations have been stubbed out currently for Windows, Solaris, and vxWorks operating systems. Also, any code built with version 4.1 or higher of the GNU gcc compiler is supported using built-in atomic intrinsic functions. A typical spinlock based mutual exclusion primitive takes about three times longer when compared to the atomic increment / decrement operations in straight line tests.

LESSONS LEARNED

As this manuscript is being prepared many parts of the project are complete but there is still some work left to be done. About 40 undefined symbols remain, and it's clear that the project has required more time than originally anticipated. It is also not possible to provide performance numbers in this paper as originally predicted. Perhaps some underlying causes can be identified.

A year ago, I was willingly assigned to another project [6] for which I was a principal software developer. My role in this project rapidly expanded until it consumed all available time for more than a year, and later this summer I spent about a month and a half preparing a bug fix release of EPICS – release R3.14.11. These projects delayed my progress due to context switching, and loss of momentum. From my perspective both of these issues are frequently behind loss of productivity with software development projects. Returning to this project has brought a positive opportunity to review my work objectively and to re-examine core design choices. That has led to the improved design described previously, but has also introduced additional delays.

For additional causes I have to look closer at my work processes. At the start of the project the scope was set for an incremental upgrade, but in the end the library was completely rewritten. This will certainly make the code simpler, better organized, and easier to maintain in the future but this has introduced some delays. A rewrite is certainly time consuming and expensive, perhaps even justifiable in the end due to increased reliability and lower maintenance costs, but the cost in terms of loss of momentum in the EPICS community is more difficult to rationalize. In retrospect breaking large projects into smaller components is a good idea because more authors can be involved, feedback from the user community can be more rapidly obtained, and upgrades can occur predictably.

I must also admit that the combination of object oriented and multithreaded development has taken the author some time to master. Each can be readily understood on-their-own but the combination of two has caused some considerable time spent on a learning curve with multiple twisting paths. Changing the public interface locking model midstream in the project has had some considerable impact on the rest of the code.

Finally, the impacts of Symmetric Multiprocessing (SMP) memory barriers, which cause CPU stalls but not consumption, on throughput haven't been determined. That lesson can only learned upon project completion.

BENEFITS FOR LANSCE

We will soon have LANSCE style dynamic, on-the-fly and ad-hoc, beam flavoring and beam timing specifying experiments in the control room, but now with a homogeneous EPICS-based and modern-hardware-based, system. At LASNCE we can transition to a tool-based approach to high level applications. This implies that high level applications will interact with an abstract model of the hardware which will facilitate incremental upgrades. Use of the well-defined EPICS network communication model can make on-call fault isolation much easier.

BENEFITS FOR THE EPICS COMMUNITY

With the new system we will place flexible device and record 3rd party module specific snapshots on the IOCs event queue. Parameters other than alarm status, time stamp, and scalar value will be correlated in time together as a single event. Array updates will also now be buffered on the event queue. The subscription filtering feature of the upgrade, being implemented generically, should provide equal benefit for all EPICS sites while being minimally invasive for legacy client side tools. In the new system we will have array index metadata – a major omission in the original EPICS specification. Finally, we will have an increasing intersection of EPICS capabilities and the needs of data acquisition systems.

CONCLUSION

The project is nearing completion, some initial design decisions needed revision, and I can identify some areas in which to revise my work flow on future projects. Nevertheless, after some delays, I hope that the EPICS community will find that the new features are useful.

REFERENCES

[1] J. Hill, "EPICS CA Enhancements for LANSCE Timed and Flavored Data," ICALEPCS'07, Knoxville, Oct 2007, WPPA24, p. 365, (2007).
[2] Herb Sutter, "The New C++:Smart(er) Pointers," "http://www.ddj.com/cpp/184403837/.
[3] J. Hill, "Next Generation EPICS Interface to Abstract Data," ICALEPCS'01, San Jose, 27-30 Nov 2001.
[4] http://www.boost.org/.
[5] A. Alexandrescu, *Modern C++ Design* (Addison-Wesley, 2001).
[6] Martin Pieck, Jeff O. Hill, John F. Powers "System Integration effort on MagViz a Liquid Explosive Detection Device," This conference.