EPICS CA ENHANCEMENTS FOR LANSCE TIMED AND FLAVORED DATA *

J. Hill, LANL, Los Alamos, NM 87545, U.S.A.

Abstract

Currently, the subscription update event queue in the EPICS server is capable of carrying payloads consisting of a channel's value, time stamp, and alarm state. The complexity of the LANSCE macro pulse beam structure requires unique capabilities from the control system, which is currently a hybrid of EPICS and the original VMS-based LANSCE Data System. A homogeneous EPICS based system with a tool based approach to the development of modular application programs is the favored post upgrade configuration, but this evolves new requirements for EPICS. Specifically, EPICS Channel Access (CA) Clients must dynamically specify the LANSCE macro pulse beam gate combinatorial, and a time window, to be sampled when they subscribe. EPICS upgrades fulfilling these requirements, including generic software interfaces accommodating site specific event queue payloads and client specified subscription update filtering expressions, 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. As time progressed, the facility gained capabilities evolving into the Los Alamos Neutron Science Center (LANSCE). 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 "Flavored" 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.

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

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

Any signal in the RICE system can be sampled either un-timed (at anytime), timed, flavored, or both flavored and timed (a common configuration).

If a particular signal (a current monitor, for example) is typically read at the same time and flavor, its record in the LCS database will contain default timing and flavoring information. If several beam flavors are monitored by the same current monitor, it will usually have multiple entries in the LCS database - each with a different externally visible Process Variable (PV) names. Essential LANSCE physics applications have knowledge of, and are dependent on, the flexibility of the RICE system. The LCS system software interfaces specify a "device read on demand" paradigm. The multiplexed RICE IO system requires that we limit the number of timed/flavored device read interactions to only those that are initiated by active clients. Application programs will typically request scheduling of a time and flavor qualified read from a RICE channel and then block for completion. Completion requires an available time slot on the RICE multiplexer occurring when the requested flavor is available. When comparing this approach with EPICS it is important to observe that this guarantees that the sensor was sampled after the read request was initiated in software.

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.

Thread Based Scheduling within an IOC

The EPICS IOC is implemented using a substantial number of independent threads of execution. This was done originally to manage proper deterministic response to external events, to keep the different components of the system independent, and to manage the ordered degradation of the system under load. This design is also an opportunely synergistic match for modern light weight thread based operating systems running on modern multi-core CPUs.

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 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 is predictably 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 threads to execute at relatively higher priorities and for the CA Server threads to execute at relatively lower priorities.

Event Queue – Theory of Operation

Communication between EPICS Record processing and the CA server occurs via subscription update events stored in the order of their occurrence on an event queue. Considering that potential strongly exists for an EPICS record's subscription update event production rate to significantly exceed the CA Client's event consumption rate, the event queue linking the two subsystems must be designed so that EPICS Record processing *never* blocks for a slow client. When there is a burst of updates, the queuing subsystem needs to avoid discarding intermediate updates. When the sustained production rate exceeds the client's sustained uptake rate, then the queuing subsystem needs to keep subscriptions current while at the same time discarding some of the intermediate updates which would otherwise force the execution of EPICS Records to block when broadcasting updates to the event queues of each client. Another central pillar of the design is that, as the client load on the IOC increases, any induced load on the EPICS Record processing threads shall be minimized.

Event Queue – Existing Design Limitations

Currently a ring buffer data structure is used to implement the event queue. That design is efficient, but very inflexible. Each fixed-sized entry in the ring buffer can store only the channel's scalar value, alarm state, and time stamp. This means that these are the *only* EPICS parameters that can be delivered in an instance-in-time correlated package. For example, snapshots documenting events occurring in the system can't contain multiple process variables, and the EPICS Record and or device specific codes are unable to extend the payload associated with an event. It is not possible to record the current state of the LANSCE beam gates in the event payload. Currently, array value state change events, but not the array value, are stored on the event queue, and we are therefore much less likely to fully document bursts of activity for array channels. Furthermore, reproducing LCS timed read capabilities requires indexing of a subset of the array elements stored in the event payload based on its position in time, but metadata linking array index increments with time aren't visible to clients. These limitations in the legacy EPICS event queue design prohibit on-the-fly, ad-hoc, application driven flavoring and timing specifying experiments that are fundamental to productive beam optimization activities at LANSCE.

The device and project independent view presented by client side programming interfaces is a positive facet of the EPICS tool-based-approach we endeavor to preserve. Modest upgrades to preexisting project portable tools, for example operator interfacing and archival clients, allowing site specific configuration parameters such as the necessary LANSCE specific flavor of beam to be specified will be required. The intent is for such tools to remain generic while allowing project specific subscription update filtering configuration using project portable interfaces. At LANSCE, an abstract view of hardware capabilities will facilitate future upgrades.

Processing capacity of modern CPUs significantly exceeds those of the original EPICS installations, but we must be cognizant of a large installed base of legacy processors and a desire to remain within elemental embedded processor capacity. CPU efficiency consistent with past expectations is therefore a design goal.

Event Queue –Memory Management Upgrade

The efficiency of the original event queue design resulted from identically sized payloads. All of the memory for the queue was pre-allocated, and so a size-generalized (pool of random sized blocks) dynamic memory allocator wasn't used. Size-generalized allocators, while quite adequate in performance for many general purposed programs, must be prudently employed in long lifespan programs running in limited resource embedded environments. There is an inescapable tradeoff; the random sized block allocation pool can either be prone to fragmentation over time, or it can be prone to increasing inefficiency as the number of blocks in use increases [1]. At the opposite end of the efficiency spectrum can be found free list (pool of fixed-sized blocks) based memory allocation schemes [2].

Consequently, there is a dilemma; the best level of efficiency requires size fixed allocation while flexibility requires inefficient size-generalized allocation. We observe that it's typically sufficient for an event payload data structure to be compile time fixed to the EPICS Record and or the Device that is posting an event, or based on a list of fixed-sized blocks (for arrays). The solution is to move the event producer, and its memory allocator, to plug-compatible module specific to implementing the EPICS Record, or to interfacing with a particular Device. Within the EPICS Record and or

Device dedicated module size-dedicated free lists can be used to allocate event data structures efficiently.

A solution is apparent but two issues remain. One, EPICS Record and or Device independent software must somehow efficiently interface with data stored in a plug-compatible module's proprietary format. Two, we must arrive at a mechanism whereby the allocated memory can be returned to the allocator's free list only after the last reference to it on the client dedicated event queues is consumed.

The solution to the first problem is Data Access [3][4] - a plug-compatible cataloging interface for proprietary data containers along with a support library for copying between proprietary data containers. The cataloging interface has functions for traversing all of the parameters in a container, for finding a particular parameter in a container, and for assigning one container to another container. Using this interface, we can implement a messaging system without requiring compile time knowledge of the message's data structures – a unique feature. Consequently, we can transport EPICS Record and Device proprietary event payloads on a generic subscription update event queue.

For a solution to the second issue a prominent design pattern, the reference counting smart pointer [5], is employed. This value added pointer class increments a reference count when a pointer instance is copied, decrements that count when a pointer instance is destroyed, and calls a plug-compatible memory de-allocation interface when that count decrements to zero.

Event Queue – Filter Upgrade

Productive beam tuning and optimization require that timing and flavoring specific experiments be set up ad-hoc and on-the-fly by application programs. A tool based approach to the development of EPICS application programs implies that EPICS clients must somehow specify the LANSCE specific timing and flavoring data capture constraints while at the same time preserving the site independent nature of the EPICS components.

Our solution is to modify the CA client side API and protocol, establishing a subscription, to include an optional character string specifying a generalized filtering logical expression operating on the parameters in the event payload. If the generalized expression tests true then a subscription update is forwarded to the client, and otherwise it is suppressed. At LANSCE, this feature will filter for the flavor of beam that is requested by the client.

Filtering updates against a logical expression character string is certainly a device and project independent approach requiring minimally invasive upgrades to preexisting tools, but performance is a concern. Our intent is to employ one of the high quality open source implementations compiling expressions into efficiently executed byte code.

Miscellaneous Issues

EPICS CA clients query various properties of the process variable such as the units, limits, or display precision. Expansion of this set to include the magnitude of the index of the first array element, the magnitude of a single array index increment, and the units of these index magnitudes is necessary. For LANSCE timed data the two magnitude parameters would specify the time offset of the first element, and the inverse of the array's sample rate.

LANSCE applications expect that the sensor is sampled after the read request is initiated in software, but EPICS doesn't guarantee this. The proposed solution is new support for one-update-only subscriptions.

LANSCE applications expect correlated read capabilities where a hardware read of multiple signals is constrained to occur within the same machine cycle - a challenging requirement for a distributed system such as EPICS where the hardware being read might be on multiple IOCs. The traditional EPICS option is correlating after the fact using the data's time stamp. A new option might be filter expressions constraining such reads to modulo N_{th} occurrences of a flavor in modulo M_{th} super-cycles along with event payloads including the super cycle occurrence index synchronized across IOCs by the timing system. A support library might query the current super cycle index, compute an index offset, schedule the reads to occur after when setup is likely to complete, and reschedule the data take in the unlikely event of unsuccessful on-time setup.

At LANSCE potential exists for the occurrence rate of certain flavors to induce excessive load on the system. The proposed solution is for device specific code to decimate the record processing of commonly occurring flavors. The machine cycles selected for decimation would be identical in all IOCs as synchronized by the timing system.

CONCLUSION

EPICS upgrades accommodating LANSCE data acquisition requirements are being installed. Our implementation is consistent with the EPICS site independent tool based approach, and consequently we hope that these new capabilities will improve the overall utility of EPICS expanding its intersection into the domain of data acquisition systems.

REFERENCES

[1] D. Knuth, "Fundamental Algorithms, Third Edition," Addison-Wesley, 1997, pp. 435-456.
[2] T. Cormen et. al., "Introduction to Algorithms, Second Edition," MIT Press, 2001, pp 210 - 212.
[3] J. Hill, "Next Generation EPICS Interface to Abstract Data," ICALEPCS'01, San Jose, 27-30 Nov 2001.
[4] R. Lange, "Data Access – Experiences Implementing an Object Oriented Library on Various Platforms", ICALEPCS'01, San Jose, 27-30 Nov 2001
[5] S. Meyers, "More Effective C++," Addison-Wesley, 1996, pp 159-213.