ADVANCED MONITOR/SUBSCRIPTION MECHANISMS FOR EPICS*

Ralph Lange, Helmholtz-Zentrum Berlin / BESSY II, 12489 Berlin, Germany
Andrew Johnson, Argonne National Laboratory, Argonne, IL 60439, USA
Leo Dalesio, Brookhaven National Laboratory, Upton, NY 11973, USA

Abstract

Publish/subscribe systems need to handle the possibility that there are subscribers requiring notification at an update rate much lower than the publisher's natural frequency, or synchronized to external events. Feedback or pulse-to-pulse diagnostics are processed at rates in the 100Hz or even multi kHz range, while many subscribers will not be able to process the data at this rate: e.g. archiving, visualization, and processing clients each require specific, different update rates. Sending more updates than required wastes processor and network bandwidth. A subscriber should be able to specify rate limiting factors or filters that are instantiated and guaranteed by the publisher. Many accelerators, especially pulsed machines, are using a hardware event system to distribute fiducials and events from a central event and/or frequency generator. These events should be integrated into the publish/subscribe system to support posting event synchronous updates to subscribers that require synchronized data. This paper investigates several approaches to provide these functionalities in the EPICS architecture.

MOTIVATION

With front end computers becoming more and more powerful, data processing rates have been consistently increasing over the years. Today, modern diagnostics and feedback related systems are processing data at rates up to the multi kHz range. On accelerators, especially pulsed machines, many of these fast systems are connected to site-wide event and timing systems, which provide accurate synchronous nanosecond resolution time stamps, and distribute fiducials as well as synchronized events.

Variable Update Rates

Some clients of this front end data will require every update and need to subscribe at the full rate, e.g. distributed feedback loops, or turn-by-turn diagnostics clients. Other clients, e.g. archiving and visualization tools, will have completely different requirements and are only interested in low or medium update frequencies. Any fixed update rate will not need the various clients' requirements.

Event Related Updates

If an event distribution system is used, a number of clients will be interested in subscribing to event-correlated updates, to obtain coherent synchronized data sets from more than one front end computer at the time of a certain event.

CURRENT LIMITATIONS

The EPICS (Experimental Physics and Industrial Control System) toolkit [1] facilitates Channel Access (CA) as its IP based network protocol [2], using a publish/subscribe mechanism for data updates to clients. There are a few limitations in this area, though, that need to be addressed.

Fixed Update Rates

Mechanisms implemented in the EPICS real-time distributed database determine the rate of data updates to clients. Clients can specify the types of events they are interested in as a combination of “regular data update”, “archive data update”, and “alarm status/severity change”. All EPICS records will send updates for all their fields on alarm status/severity change. Most records can be configured to send data and archive updates on value change, or whenever the record is processed. For numerical values, two separate configurable dead bands are supported, one for regular data, one for archive data updates. All these configurations are done at the EPICS record level – Channel Access clients cannot specify per subscription update rates or update rate limits.

Currently, as a workaround at the application level, additional alternative records with predetermined useful processing rates may provide updates at a limited rate.

Limited Event Correlation

The existing EPICS event system drivers connect to the EPICS time stamp mechanism and specialized EPICS record types. E.g., if configured correctly, an event record would be *processed* on receiving a specified hardware event, and could start a chain of other records in the IOC (Input Output Controller) database to be processed. The event system drivers also allow all IOC *time stamps* being supplied by the timing/event system hardware [3,4].

Other than for record processing and generation of time stamps, there is no mechanism to filter data updates from otherwise unrelated records on system state information derived from receiving timing system hardware events.

DESIGN CONSIDERATIONS

A number of considerations have to be taken into account when designing an extension to overcome these limitations.

*Work supported by U.S. Department of Energy (under contracts DE-AC02-06CH11357 resp. DE-AC02-98CH10886), German Bundesministerium für Bildung und Forschung and Land Berlin.

CA Protocol Compatibility

The existing Channel Access protocol is widely used within the large number of existing EPICS installations. Previous developments have shown that extreme care has to be taken: changes to the on-wire protocol should be avoided if possible as they cause a noticeable increase in code size and must be thoroughly tested for compatibility with a wide range of EPICS versions.

Modularity

To facilitate running EPICS IOCs on small systems, any extensions should be modular, and add only minimal object code and run-time memory consumption when not being used.

API Compatibility

Some internal APIs of the EPICS database are heavily used by code outside the scope of EPICS Base, such as locally added device drivers and record types. Such APIs should be left unchanged if possible, to avoid breaking 3rd party code.

EXISTING UPDATE MECHANISM

The existing mechanism for updates is shown in Fig. 1. The example shows two Channel Access clients, one with three subscriptions to different fields of a record, the other with one subscription to a field of the same record.

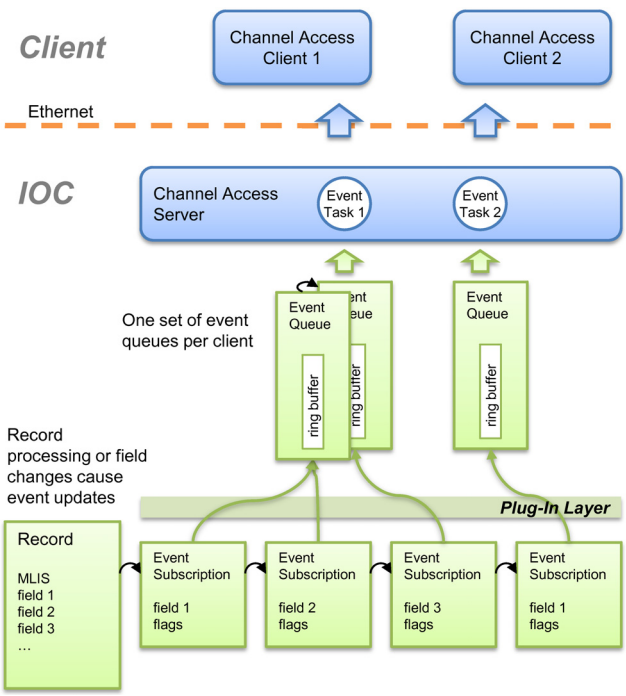

Figure 1: Event update mechanism.

Network Connection

After channel discovery, Channel Access creates one bidirectional TCP connection for each combination of client application and server (IOC). All traffic between the client and the IOC is routed through that connection.

Event Queues and Subscriptions

On the IOC, Channel Access creates an *event queue* (ring buffer) for each client connection, to buffer data updates whenever the database generates events faster than the CA server is able to send to the client. For clients with many subscriptions, additional event queues are generated and linked to the first queue as needed.

One *event task* is started for every client, that reads updates from the client's event queue(s), converts the data to the Channel Access network format, puts it into a network buffer, and sends the filled buffer through the TCP connection to the client.

Every record instance keeps a linked list of *event subscription* structures, one for each subscription to one of its fields. The event subscription contains the field of the record that is being subscribed to, additional configuration flags for the subscription, as well as a pointer to the event queue that updates should be put into.

Data Updates

The routine db_post_events(), which is being called when setting a record's field or as part of record processing for every changed field, iterates through the event subscription list and writes data to the appropriate event queue on a match.

Records will usually send updates by calling db_post_events() on change, or whenever they are processed. Numerical records keep two separate configurable dead bands to limit the update rate, as described above. All records send updates on alarm severity or status changes.

SERVER-SIDE PLUG-INS

Most of the described shortcomings can be resolved by introducing a layer of client-configurable per-subscription plug-ins on the server side.

Mechanism

Additional processing of updates, e.g. event filtering and synchronization, is done in plug-in modules, that can be inserted between the event subscription structures and the event queues when the connection is made. These plug-ins have APIs both for receiving and generating event data, so they can be stacked. Fig. 1 shows the location of the proposed plug-in layer in the existing update scheme.

Instantiation and Configuration

A recent addition to EPICS is the ability for clients to add modifiers to channel names, that are transparently forwarded to the IOC [5]. A JSON (JavaScript Object Notation) parser has been added to the IOC, allowing channel modifiers that use JSON notation to specify arbitrary structures.

The object code for plug-ins is loaded on the IOC. All plug-ins register themselves at boot time, providing a name tag and a set of callbacks for the JSON parser. When the client request for a subscription is processed,

the IOC parses the client-supplied modifiers for the channel. Upon finding a plug-in tag, it creates and connects an instance of the plug-in and lets the plug-in parse its configuration by forwarding all configuration data callbacks.

This design allows extension of the IOC functionality by adding plug-ins without any change to EPICS Base or recompiling the IOC software. It may even be possible to add plug-ins to a running IOC, as long as the necessary registrations are done.

POSSIBILITIES

Server-side plug-ins can be used for a variety of per-connection event processing, including, but not limited to:

- Update rate limiting: The client specifies a maximum update rate for the subscription.
- Update correlation with timing systems: The client specifies a timing system derived system state to determine when updates should be sent.
- Event value filtering: E.g. sliding average of values.
- Server-side buffering of array data.

SUPPORT LIBRARIES

Event Correlator

To support plug-ins that correlate event updates with system states provided by external systems, a library will be provided to handle system state flags.

The external event system driver will be able to simply set or reset a system state based on incoming hardware events or as part of a timer callback. These actions will be stored with their corresponding time stamps, so that an event plug-in can query the correlator with the record processing time stamp to determine if record processing happened during a certain system state being true. This check will work even if the state was true for only a short period and update processing has been delayed.

Memory Allocator

As plug-ins are instantiated per subscription, they are expected to cause a significant number of memory allocations and deallocations for plug-in internal structures of unknown size.

To avoid a performance hit and system memory fragmentation, adding an universal allocator library underneath of or in addition to the existing free list allocators will be considered. After a first plug-in freed all its structures, such an allocator should be able to reuse the memory for instances of other plug-ins using structures of a different size. Depending on the type of target system, the allocator implementation could facilitate the widely used SLAB [6,7], SLUB [8], or SLOB [9] algorithms.

STATUS

This project is in its design phase. It is part of a 1-year effort to add features needed by the NSLS-II project to the EPICS toolkit, and is expected to be finished within that time. The required first set of plug-ins will include update rate limitation and filtering based on external event driven system states as described above.

CONCLUSION

The addition of server-side plug-ins to the EPICS toolkit adds valuable functionality to the EPICS IOC. It will help to overcome limitations of the existing design, and allow the future addition of new functionality without creating a major impact on performance, memory footprint, or complexity.

New features of EPICS facilitate run-time plug-ins with arbitrary configuration supplied by the client application.

REFERENCES

[1] Experimental Physics and Industrial Control System, http://www.aps.anl.gov/epics.

[2] J. Hill, R. Lange, “EPICS R3.14 Channel Access Reference Manual”, http://www.aps.anl.gov/epics/base/R3-14/11-docs/CAref.html.

[3] J. Winans, J. Kowalkowski, A. Johnson, “EPICS 3.14 Support for the APS Event System”, http://www.aps.anl.gov/epics/modules/timing/apsEvent.

[4] B. Kalantari, “generalTime – EPICS R3.14 Support for Clock Time”, http://epics.web.psi.ch/software/generalTime.

[5] A.N. Johnson, R. Lange, “Evolutionary Plans for EPICS Version 3”, WEA003, this conference.

[6] J. Bonwick and Sun Microsystems, “The Slab Allocator: An Object-Caching Kernel Memory Allocator”, USENIX Summer 1994 Technical Conference, 1994, p. 87-98.

[7] B. Fitzgibbons, “The Linux Slab Allocator”, 2000.

[8] C. Lameter, “SLUB: The Unqueued Slab Allocator V6”, 2007, http://lwn.net/Articles/229096.

[9] M. Mackall, “SLOB: Introduce the SLOB Allocator”, 2005, http://lwn.net/Articles/157944.