EVOLUTION OF THE EPICS CHANNEL ACCESS PROTOCOL

K. Žagar*, M. Šekoranja, Cosylab d.d., Ljubljana, Slovenia
M. R. Kraimer#, ANL, Argonne, Illinois, USA
L.B. Dalesio, BNL, Upton, Long Island, New York, USA

Abstract

Experimental Physics and Industrial Control System (EPICS) is one of the most widely deployed control system infrastructures in the large experimental physics community. At EPICS' foundation are 1) the real-time process database, which allows integrators to build the control system from reusable building blocks (e.g., device drivers) into a coherent whole without much coding or other kind of development, and 2) the Channel Access protocol, which allows the database to be distributed across several computers in a scalable way. In this contribution, we describe the objectives of the next major EPICS release (v4). In particular, we focus on the improvements of the Channel Access protocol that will allow it to support additional functionality, such as structured process variable data (pvData) and client-specified filters. We also describe how this functionality is implemented while simultaneously further improving the Channel Access' performance (no-copy get, flow control improvements, beacon traffic reduction, zero-length queues, etc.). We also discuss potential for future improvements, such as use of IP multicast and a layer for implementing remote-procedure call style of communication.

INTRODUCTION

EPICS [1] is a software framework for development of distributed control systems. At its core is the concept of a ***process variable***, which represents either an input (e.g., a readout from a digital input) or an output (e.g., a reference setpoint for an analog output).

Process variables are modelled as ***records***, which contain several ***fields***, each having a name, type and well-defined meaning. E.g., the `VAL` field is the value of the process variable, `STAT` the current alarm status, etc.

Records can be ***processed***. When a record is processed, it may take some data from other records' fields (via ***links***), perform data acquisition from hardware, send commands to hardware, change the value of its fields, and trigger processing of other records that are linked to it.

EPICS provides for so called ***soft records***, which do not have any physical device assigned to them, but can perform operations on data obtained from their input links and update their value as required – triggering processing along their output links. This allows for implementation of (soft) real-time control loops, control sequences, etc.

EPICS also has provisions for development of client applications such as human-machine interfaces and central services (e.g., archiving). These applications are able to read field values of fields (***get***), or send values to fields (***put***).

As client applications and process variables are usually not on the same computer, a middleware layer is provided that makes the communication between nodes across network links transparent. This is the role of the ***Channel Access* (CA)** protocol.

The CA protocol has a server-side component (the ***CA Server***) and its client-side counterpart (the ***CA Client***) – see Figure 1. The CA Server brokers requests received across the network to the locally hosted records. The CA Client locates fields across the network (using UDP broadcasts) and transfers data to/from them via so-called ***channels*** across TCP connections.

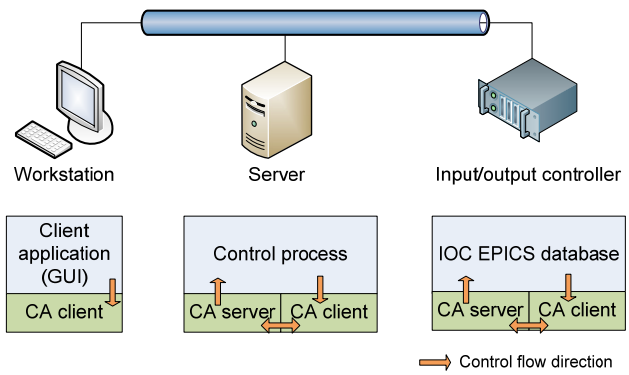

Figure 1: Architecture and data flows in EPICS.

In this paper, we present how the next major release of EPICS (version 4) which is currently under development [2], will improve the CA protocol of EPICS version 3, which is currently widely used. In the text that follows, we shall refer to the existing CA protocol as "EPICSv3 CA".

EPICS V4 CA IMPROVEMENTS

Clean Design with Few Dependencies

The EPICSv3 CA implementation has no dependencies on third party libraries, middleware, toolkits and frameworks. The only dependency it has is to a lightweight operating system abstraction library which allows it to be platform-independent.

Consequently, from early 1990's till today, CA implementation was relatively immune to technological evolution in the field of operating systems and middleware technologies. A recent study [3] has shown that despite EPICSv3 CA uses a *roll-your-own* approach to middleware, in terms of performance it is not at a disadvantage compared to more recent and even state-of-the-art middleware solutions such as CORBA [4], ICE [5] and DDS [6].

* klemen.zagar@cosylab.com
# on leave of absence

Prior to commencing with design of EPICSv4 CA, we have extensively evaluated commercial off-the-shelf (COTS) as well as open source middleware solutions. We have decided that the longevity, flexibility and maintainability risks associated with adopting a third party middleware do not outweigh the additional effort required to develop the protocol atop of the socket and threading APIs, especially considering domain as well as technical experience we have gained analyzing and developing EPICSv3 CA protocol [7][8], and low risk exposure due to external factors as already experienced by the EPICSv3 CA implementation.

EPICSv3 CA implementation evolved from an initial set of requirements into its present form. As flexibility was not on top of the priority list for the initial implementation, the present design does not allow for easy addition of new functionality without running at a risk of breaking existing one.

Therefore, to make future adaptations easier, we have leveraged the experience gained by the software engineering community in the past two decades (e.g., the design patterns [9]) and put a clean design as a top priority. For example, extensive and consistent use of the factory design pattern allows replacement of any software component without affecting the rest of the implementation – see Figure 2.

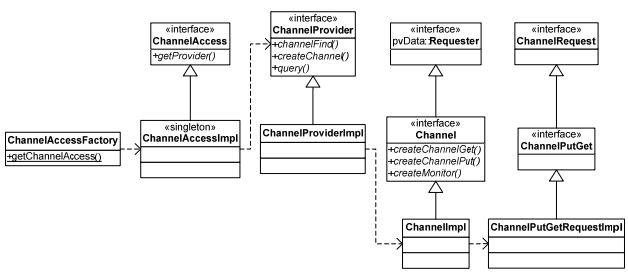

Figure 2: Class diagram showing how classes reference each other only through interfaces, leaving choice of actual implementation to configurable factories.

In EPICSv3, the reference implementation of channel access is written in the C++ programming language. In EPICSv4, we have decided to develop and consider as reference the Java version, because nowadays Java offers development tools that allow for shorter development cycles, resulting in faster prototyping and development.

Asynchronous API and Design

The easiest way to work with input/output operations is by using synchronous semantics, such as the following:

```
sendCommand("QUERY CURRENT");
current = receiveResponse();
```

This approach, while very intuitive, will cause the second call to block until the response is received – which can also be a very long, or even infinite, time. Also, waiting for several responses cannot be scheduled simultaneously, significantly reducing the performance.

To allow parallel execution, threads can be used, but then the developer needs to take precautions to prevent unanticipated concurrent access to shared data structures. Also, threads consume resources and reduce performance as they cause frequent task switching.

A better approach is to use asynchronous semantics:

```
requestResponse(responseHandler, errorHandler);
…
void responseHandler(char *response) { … }
void errorHandler(char *error) { … }
```

In this case, `responseHandler` is called when the response is ready, and `requestResponse` call completes immediately (no blocking). Also, unexpected conditions can be handled in the `errorHandler`.

In EPICSv4 CA, the API is primarily asynchronous, and the underlying design is optimized for asynchronous operation. Synchronous API will be created atop of the asynchronous one.

Support for Structured Data

In EPICSv3, a record can consist of a set of fields. Each field can be a scalar or an array of scalars of the same type. More complex data types, such as fields, were not foreseen in the design (Figure 3, left).

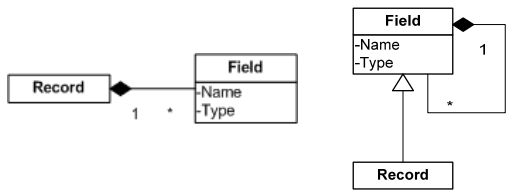

Figure 3: In EPICSv3 (left) fields can be only scalars or arrays. In EPICSv4/pvData (right) fields can contain other fields as well (structures).

EPICSv4 also allows for structures (Figure 3, right). In EPICSv4 data separation is cleanly separated from other components (CA, input/output controller – IOC), and has no dependencies. The component for data representation is called ***pvData***.

Simultaneous Access to Several Fields

In EPICSv3, CA Client is able to access fields only individually. Thus, getting value of two fields of the same record requires two GET CA commands to traverse the network. For example:

```
Channel ch1 = context.createChannel("RECORD.VAL");
Channel ch2 = context.createChannel("RECORD.STAT");
ch1.get();
ch2.get();
```

Apart from being less efficient, this approach also prevents the client from obtaining a consistent snapshot of the server's fields, because the second field's value could have changed since the retrieval of first field's value.

In EPICSv4, CA Client first defines a subset of fields of a record it is interested in by defining a `ChannelGet` request object. This can be as small as a single field, or as

all-encompassing as all fields, including those that are deep in the structure. This information is shared between the client and the server, and client can request the same subset several times, without having to communicate all the fields with each request, but just identifying the request.

This approach not only allows the client to retrieve a consistent snapshot of the field values, but it also conserves required bandwidth and does not result in excessive creation of objects at CA client and server.

```
ChannelGet cg = channel.createChannelGet(
  …,
  ChannelAccess.createRequest("alarm, timestamp"),
  …);
cg.get(false);
cg.get(true); // the last call - will dispose the ChannelGet
```

Client-Specified Filters

In EPICSv3, the monitoring policy of a record is specified when configuring the database (e.g., the `SCAN` field to specify the update rate of a record).

This was found to be too limiting, therefore in EPICSv4 the client can specify a monitoring policy on the level of a monitor, which applies to a subset of record's fields.

Presently, on-percent-change, on-absolute-change, on-change and on-put algorithms are supported. Also, a mechanism exists to add additional algorithms without needing to change the EPICSv4 CA code.

Monitor Flow Control

It may happen that the server dispatches monitors to the client faster than the client can handle them. This can occur because the network link does not provide sufficient throughput, because the client's CPU is overloaded, or because of improper handling of monitors in the client application.

If improperly implemented at the server side, the server might block when sending data to the client, as the TCP send would block until TCP flow control detects enough free buffer at the client side.

In EPICSv4 CA, a FIFO queue is provided at the server to send data. The queue size is configurable, and the following special sizes are of a consequence:

- 0: the data is sent to the network directly from the server's data structure. This is very efficient because no data copying is needed, but if the server changes the data while it is being sent, it might arrive to the client in an inconsistent state (e.g., first part of an image – an array of bytes – belonging to the first frame, and the second part of the image belonging to the second frame).
- 1: a single element in the queue. The data is cached prior to sending.
- More than 1: a queue. When adding data to the queue, and if the queue is full, the last element of the queue is replaced with the added data, and an overrun flag is set.

Remote Procedure Calls

In EPICSv3 CA, it is not possible to implement the remote procedure call semantics correctly. This semantics calls for data being provided to the server, the server processing the data, and returning the result of the processing back to the server.

EPICSv4 CA provides a mechanism called `PutGet` request. Here, the data is first provided to the server (the put part), the server record is processed, and the resulting data is returned.

CONCLUSION

EPICSv4 CA provides significant new features to the concepts of EPICSv3 CA: improved handling of monitors, which is configurable by the client when requesting data, ability to perform remote procedure calls, etc.

At present, the implementation of Channel Access and the Java IOC is sufficiently complete to allow for applications to be developed. The source code is freely available on SourceForge [10], and those interested are invited to evaluate the suitability of EPICSv4 for their applications.

Though on the wire EPICSv4 CA is not compatible with EPICSv3, EPICSv4 applications (either clients or servers) are able to talk with EPICSv3 applications, assuring for seamless integration of EPICSv4 solutions in EPICSv3 environments.

REFERENCES

[1] EPICS web page: http://epics.aps.anl.gov
[2] M. Kraimer, "JavaIOC Status", EPICS Collaboration Meeting, Kobe, Japan, October 2009
[3] K. Zagar, "ITER Control System Technology Study", EPICS Collaboration Meeting, Vancouver, Canada, April 2009
[4] Object Management Group, "Common Object Request Broker Architecture – CORBA", http://www.corba.org
[5] ZeroC: "Internet Communications Engine – ICE", http://www.zeroc.com
[6] Object Management Group, "Data Distribution Service for Real-time Systems (DDS)", Revision 1.2
[7] M. Sekoranja, "Native Java Implementation of Channel Access for EPICS", ICALEPCS'05, Geneva, Switzerland, October 2005
[8] A. Pucelj, M. Sekoranja, K. Zagar, "Channel Access Protocol Specification", Revision 1.4, February 2008
[9] E. Gamma et al., "Design Patterns: Elements of Reusable Object-Oriented Software", Addison Wesley, November 1994
[10] EPICSv4 Channel Access and Java Input/Output Controller implementation: http://epics-pvdata.sourceforge.net/