Evolution of the EPICS Channel Access Protocol

Klemen Žagar <klemen.zagar@cosylab.com>
Matej Šekoranja
Marty Kraimer
Bob Dalesio
Overview

- The role of the EPICS Channel Access (CA) protocol

- What is new in EPICSv4’s CA?
  - Clean design with few dependencies
  - Asynchronous API and design
  - Support for structured process variable (PV) data – pvData
  - Connecting to several fields of a pvData structure
  - Client-specified filters
  - Flow control for monitors
  - Remote procedure calls

- Plans

- Conclusion
The role of the EPICS Channel Access (CA) protocol

- Enables communication between EPICS clients, input/output controllers and other nodes.

A record is an addressable, self-consistent unit of data. E.g., represents a process variable. Consists of fields (e.g., value, alarm status, etc.).

A channel:
- (Virtual) connection between client and server.
- System-wide unique name.
WHAT IS NEW FOR EPICSV4?
Clean design with few dependencies

- Depends only on pvData (more info later).
  - No dependencies on third-party software, not even middleware such as CORBA, DDS, ICE, etc.
  - Network and concurrency management can be reused from previous projects (e.g., Channel Access for Java).
- Clean separation of interface and implementation reduces coupling.
  - Extensive use of factory design pattern to allow changing of implementations without requiring modifications of code.
Asynchronous API and design

- When dealing with input/output or communication, asynchronous API allows for better application design.
  - Able to launch operations in parallel without having to spawn threads.
- Drawback: code is more complex even for simple operations.
  - A synchronous helper API will be provided.
- Example:
  - create a get request:

```java
ChannelGet createChannelGet(
    ChannelGetRequester channelGetRequester, // Whom to notify
    PVStructure pvRequest, // A structure describing the desired set of fields
    String structureName, // The name to give to the created PVStructure.
    boolean shareData, // On the remote side should the companion
    // PVStructure share data with the PVRecord.
    boolean process, // Process before getting data.
    PVStructure pvOption // Additional options (e.g. triggering).
);
```

- when the request is complete, callback on the requester is called:

```java
void channelGetConnect(
    Status status,
    ChannelGet channelGet,
    PVStructure pvStructure,
    BitSet bitSet
);
```
Support for structured process variable (PV) data

- In EPICSv3, each record had fields, which were scalars or arrays.

- With `pvData`, a field can also be a structure.
  - Thus, record is a top-level field.

- Example:
  ```
  powerSupply
  alarm
  timeStamp
  power
  value
  alarm
  voltage
  value
  alarm
  current
  value
  alarm
  output
  value
  ...
  ```
Connecting to several fields of a pvData structure

- In EPICSv3, each CA channel allowed connection to a single field.

  Channel ch = context.createChannel("RECORD.VAL");

- In EPICSv4, a channel can connect to a subset of record’s fields:

  ChannelGet getReq = channel.createChannelGet(...,
  ChannelAccess.createRequest("alarm, timestamp"),
  ...);

- Once a ChannelGet object is created, the same get request can be re-issued:

  getReq.get(false);
  getReq.get(false);
  ...
  getReq.get(true); // last request
Client-specified filters

- Client can configure behavior of monitors to filter-out any changes that do not affect it.
  - The filtering is done already at the server-side (no network traffic).
  - The filters are client-specific: e.g., different clients might have different dead-band tolerances.

- These monitoring algorithms are available:
  - **onPercentChange**: if change is within deadband percent of the last reported value, it is not reported.
  - **onAbsoluteChange**: if change is within deadband of the last reported value, it is not reported.
  - **onChange**: report any change.
  - **onPut**: report whenever the value of the field is set.

- Additional monitoring algorithms can be registered at the server.
  - E.g., monitoring on an external trigger.
  - Client code must use the name with which the monitor is used at the server, and provide any needed parameters.
Flow control for monitors

- If changes are too frequent, server can generate monitors more frequently than they can be handled.
  - Bottleneck: network or client’s CPU.

- Flow control allows the server to detect when the client can no longer accept monitor notifications:
  - TCP flow control.
  - Server monitors clients’ receive buffers.

- Server has a queue for monitors. Queue size is configurable:
  - Shared: 0 – the server sends the data directly from the record
  - Cached: 1 – the server maintains the latest copy of the data
  - Queued: $n > 1$ – $n$ last versions of data are kept in a FIFO
Remote procedure calls

- In EPICSv3, remote procedure calls (RPC) were impossible to perform, unless in some special cases or with special approaches:
  - E.g., marshal invocation data in a waveform, and un-marshal in device support…
  - How to get the return value or completion status?
  - How to correlate return value with invocations (e.g., if several concurrent invocations are in progress).

- CA for EPICSv4 has a special provision for RPC-style communication: the `PutGet` request:
  - First put data to some subset of record’s fields.
  - Wait until processing at the server completes.
  - Then get data from a subset of record’s fields.
- Decreased beacon traffic
  - Server does not send beacons.
  - When server receives an echo request from the client, it responds.
  - Client only echoes a server if the server doesn’t send data for a period of time.

- A note on compatibility
  - EPICSv4 CA protocol is not compatible with EPICSv3.
  - However, it is possible to use CA libraries simultaneously.
    - Access to JavaIOC database with EPICSv3 and EPICSv4 clients.
    - Also, JavaIOC can talk with EPICSv3 or EPICSv4 servers.
Plans

- **Distributed queries**
  - Queries such as: “find all beam position monitors”

- **IP multicast monitors**
  - When several clients subscribe for same data, it would be possible to send the data to all of them.
    - A single send operation for the server.
    - The dispatching performed by network infrastructure (switches).
  - Technique: IP multicasting.
  - Requires UDP protocol.
    - Difficult to ensure reliable delivery and flow control.

- **Access control**
  - Presently, no access control checks are done.
  - At what level of granularity should access control be applied?
  - Ideally compatible with EPICSv3.

- **TCP transport improvements**
  - A stream-like API to upper levels…
  - …taking into account MTU size when transmitting data
The development focuses on Java
  - Quick and efficient prototyping and development

Other implementations are planned
  - In particular, C++

Implementation will commence when Java design and implementation are stable
Conclusion

- Development of EPICSv4’s Channel Access has reached a point where it can be used for first applications.
  - Concept-wise backward compatible with EPICSv3.
  - EPICSv4 applications can talk to EPICSv3, and vice-versa.
  - New features (such as RPC) reduce the “feature gap” with CORBA/ICE/RMI/SOAP and similar middleware.
  - Foreseen improvements (TCP transport improvements, multicasting) likely to reduce “performance gap” with state-of-the-art middleware (e.g., commercial DDS implementations).
  - Remains concentrated to control systems (monitor quality of service, etc.).

- JavaIOC
  - Infrastructure is in place.
  - No device support drivers yet, but suitable for high-level applications, integration with other systems, etc.

- Your input?
  - Now is a great time for considering new features and adjusting priorities.

- Have an application?

- Available on Sourceforge
Thank You for Your Attention