Argonne NATIONAL LABORATORY

... for a brighter future

Alternative Control Protocols for EPICS – Data Distribution Service (DDS)

Claude Saunders

Argonne National Laboratory – Advanced Photon Source

EPICS Collaboration Meeting – DESY, April 2007

Motivation

1. Learn more about details of rsrv codebase, and therefore also about the behavior of channel access.
2. Learn about OMG (Object Management Group) DDS (Data Distribution Service) specification.
   - Actually, more interested in underlying wire protocol called RTPS
3. Learn what some of the tradeoffs of a UDP-based protocol are.
4. I believe it would be useful to have a more broadly known and standards-based protocol in EPICS.
5. Get some idea of what it would take to allow for a “pluggable” protocol in EPICS.

Note: ZeroC Embedded-ICE also of interest (www.zeroc.com)

Shifu Xu at Argonne looking at full ICE, but nothing to report yet

Object Management Group (www.omg.org)

From the home of the CORBA specification comes:

- Data Distribution Service (DDS) for Real-Time Systems V1.2
  - Important!
    - *DDS is NOT A PROTOCOL - an API specification only*
    - *Implementations can be completely non-interoperable*

- DDS Interoperability Protocol (not final yet, working doc dated Aug-2006)
  - This is a “wire protocol” called RTPS, also specified elsewhere as:
    - *IEC PAS 62030 Digital data communications for measurement and control – Fieldbus for use in industrial control systems*
      - Section 1: MODBUS Application Protocol Specification V1.1a
      - Section 2: Real-Time Publish-Subscribe (RTPS) Wire Protocol Specification Version 1.0
    - *Odd, since RTPS seems to have no relation to MODBUS*
  - OMG specification is an independent specification now, though…

Implementations of DDS and/or RTPS

1. RTI (Real-Time Innovations) www.rti.com
   - Was NDDS 4.0, now called RTI Data Distribution Service
   - More or less the “reference” for DDS and RTPS, and prime contributor to OMG specifications.
   - Proprietary product
2. PrismTech OpenSplice 2.0 www.prismtechnologies.com
   - Proprietary, not open-source
   - Implements DDS specification, not DDS Interoperability Protocol
3. OCI (Object Computing, Inc.) DDS for TAO
   - OCI – home of Java JacORB and TAO C++ ORB
   - Implements DDS specification using CORBA ORB
   - Open-source
4. Ocera Project ORTE
   - Implements core of RTPS wire protocol, but not DDS API
   - Open-source

Choosing an Implementation for Prototyping

- I'm interested in an interoperable protocol, so DDS API-compliance only isn't an option:
  - This eliminates PrismTech OpenSplice and OCI DDS for TAO
- While I've heard good things about RTI's NDDS, I don't have budget or inclination to acquire proprietary product.

- So I grabbed Ocera Project ORTE to try out protocol.
  - Not a big deal, since RTPS protocol entities mostly map one-to-one with DDS API abstractions anyways. There is an OpenDDS sourceforge project, but it seems to have died off…
  - ORTE is a user-space implementation of RTPS
  - Definitely not zero-copy, and has malloc all over the place
  - Builds and runs easily on linux, have not tried elsewhere

DDS Interoperability Protocol Specification (aka RTPS)

- Specification contains 3 main parts:
  1. PIM (Platform Independent Model)
     - *Structure Module*
     - *Messages Module*
     - *Behavior Module*
     - *Discovery Module*
     - *Versioning and Extensibility*
     - *Implementing DDS QoS and advanced DDS features*
  2. PSM (Platform Specific Model) for UDP/IP
     - *Contains wire representations of messages for UDP*
  3. Data Encapsulation
     - *Note: how to serialize/deserialize payload data not specified as part of RTPS*
     - *For interoperability, all implementations must at minimum support OMG CDR data encapsulation.*

Advantages of UDP/IP transport

- RTPS has many QoS features, so best-effort transport is best match to allow RTPS to control delivery options and timeout parameters.
- UDP provides predictable behavior – RTPS not dependent on vagaries of TCP implementations, such as timeout settings
  - More portable as a consequence
- UDP multicasting a good match for RTPS publish/subscribe behavior

Note: I'm parroting others opinions here, but these points make sense to me.

Essence of RTPS

- **Publisher** creates named topics to which pre-defined messages are published. Can have multiple publishers to one topic.
- **Subscriber** subscribes to named topic(s) (with various QoS parameters and optional topic/message filter parameters) and subsequently receives asynchronous message deliveries.
  - Think “channel access monitors”
- Topics are one-way communication only, from application perspective.

- There is no topic “nameserver”
  - Each node on network maintains a “topic cache”
  - Publish/Subscribe on “built-in” topics used to distribute application topics to all topic caches.
  - Note: custom “discovery protocols” are allowed as extensions
- Connection management
  - Equivalent to channel access connection “heartbeat” exists for topics
- QoS parameters can control both message “timeout” behavior as well as policies when resources such as x-mit/recv buffers run out. Lots here.

Prototype Goal

- Use softIIOC on linux for server-side
- Use caget and camonitor command line utilities for client-side
- Gut rsrv module of all channel access, replace with RTPS
- I did not attempt to emulate channel access API

Attempt #1: Topic as Process Variable

- Create a topic per process variable at iocInit() time
  - Since topics are one-way communication, need 2 topics per PV
  - One <pvname>Writer topic and one <pvname>Reader topic
  - <pvname>Writer topic for clients to send commands to IOC
    - *Put, synchronous get, monitor request*
  - <pvname>Reader topic for server to respond to commands and send monitor events

- Problems
  - Unscalable, given default RTPS discovery protocol
    - *Means every PV (topic) communicated to every client cache at IOC startup*
  - Even worse, since you can request monitors on individual fields

Attempt #2: Topic as Virtual Circuit

- Create “ioc name” topics at iocInit() time
  - Client publishes to all known <iocname>Writers with desired PV-name plus unique client ID. Client subsequently receives response from IOC with desired PV on “reader”.
- When PVs found, create pair of topics for each client-IOC virtual circuit (ie. 2 topics per connection from a client).
  - Use these 2 topics for command/response and monitors
  - Can clean up IOC resources if subscriber (client) goes away

- Problems
  - Still wind up transmitting all topics to all clients, although far fewer than in Attempt #1. Wasteful and pointless, nonetheless.
  - Need to generate network-wide unique topic names for “virtual circuit” topics.
  - Just seems unnatural to abuse notion of topics to emulate virtual circuits.

Conclusions

- RTPS/DDS is not a good substitute for what EPICS channel access does.
  - Publish/subscribe is one-way communication only. Much of what we do is 2-way. Using topics to implement 2-way communication is possible, but would require adding custom API on top of DDS.
  - The required default discovery protocol in RTPS does not seem well suited for managing Process Variables or Virtual Circuits.
- Creating a custom discovery protocol (allowable by specification) would be a major effort, and would break interoperability.
- The core message structure and behavior of RTPS is interesting, especially the integration of QoS parameters and use of UDP transport. I did not have time to look at this closely, but it would be an interesting task to compare this to EPICS channel access and Fermilab ACNET messages and behavior. No small task…

Conclusions

- **DDS and RTPS are complex**. It’s entirely possible that I’ve missed some usages that would improve the fit with EPICS channel access. My time was limited.
  - Example: The specifications allow for subscription “filter expressions” on both topic names and message content. Implementations are free to choose whether to implement these on reader-side (easy), or writer-side (much harder).
  - The ORTE implementation I looked at only had reader-side filters, which are not very useful.
  - An implementation with writer-side filters could be used to manage problems with “topic explosion” mentioned earlier.