Experiences in Messaging Middleware for High-Level Control Applications

Nanbor Wang*, Svetlana Shasharina, James Matykiewicz, and Rooparani Panduleeka
Tech-X Corporation
nanbor@txcorp.com

Abstract

Existing high-level applications in accelerator control and modeling systems leverage many different languages, tools and frameworks that do not interoperate with one another. As a result, the accelerator control community is moving toward the proven Service-Oriented Architecture (SOA) approach to address the interoperability challenges among heterogeneous high-level application modules. Such SOA approach enables developers to package various control subsystems and activities into "services" with well-defined "interfaces" and make leveraging heterogeneous high-level applications via flexible composition possible. Examples of such applications include presentation panel clients based on Control System Studio (CSS) and middle-layer applications such as model/database servers.

This poster presents our experiences in developing a demonstrative high-level application environment using emerging messaging middleware standards. In particular, we utilize new features such as in EPICS v.4 and other emerging standards such as Data Distribution Service (DDS) and Extensible Type Interface by the Object Management Group. We briefly review examples we developed previously. We then present our current effort in integrating DDS into such a SOA environment for control system. Specifically, we illustrate how we are integrating DDS into CSS and showcase our other DDS efforts.

Why Data Distribution Service?

Data-centric publish-subscribe middleware provides more flexible coupling between data information producers and consumers. Furthermore, DDS has built-in Quality-of-Service policies that are crucial to control systems.

- Deadline: Establishes contract regarding rate at which periodic data is refreshed
- Latency: Establishes guideline for acceptable end-to-end delay
- Time-based filter: Mediates exchanges between slow consumers and fast producers
- Resource limit: Controls resource utilization by DDS entities
- Reliability: Controls message delivery (best-effort/reliable)
- History: Controls how many messages are kept by the middleware (keep last n/keep all)
- Durability: Controls the lifetime of data (volatile, transient, persistent)

Middle-Layer Server Examples

We developed an example middle-layer server on top of two different DDS implementations:

- Web Browser runs UI
- Web Server runs MAD-X or UAL
- Web Server connects to the actual Optim Server using publish protocols (DDS and EPICS-DDS)
- Optim Server runs MAD-X or UAL

DDS Performance Test Suite

- We extended the open source TouchStone performance test suite to work with most DDS implementations
- Test suite helps developers to explore different ways to configure the overall system QoS policies using different operational scenarios before actually developing the applications

Integrating DDS into CSS

Motivation: The Internex Frontier experiment employs both EPICS and DDS in the instrument. It is necessary to monitor information and send control signals over both protocols.

- We modeled after org.csstudio.platform.libm.epics and implemented a set of plug-ins to present DDS topics as PVs:
  - com.bxcorp.soaccs.platform.libm.dds
  - com.bxcorp.soaccs.platform.libm.dds.ui
  - com.bxcorp.soaccs.cpp.dds
- The added plug-ins enable CSS applications to subscribe/publish (get/set) a DDS topic (structured like a PV)
- For example, an OPI widget can be associated to a PV "/temp/temperature3"

Current Status:

- Topics (with their associated QoS policies) need to be precompiled into jar files and loaded through com.bxcorp.soaccs.cpp.dds
- We currently support only DDS topics with structures similar to PVs (with some meta-data)

Ongoing effort:

- Add support for general DDS topic structures such as, pubsub/qualitative DDS topics definitions via XML (eliminate the need to implement a separate jar file)

Python DDS (pyDDS)

- The easiest way for Python code to interact with DDS data space is to wrap up C/C++ generated code with SWIG/Boost python
  - Type-specific wrappers
  - Extra-steps necessary to generate wrappers
  - Need to regenerate wrappers when topic structures change
  - Not compatible to Python's dynamic/decipherable language nature

Middle Layer

EPICS

• Client-Server/RPC-styled/remote object messaging protocols are suitable for command/control and deployment purposes
• They are not suitable for dynamic services
• Dual messaging buses
• Solution: Adding a publish/subscribe messaging protocol
• OMG DDS
• Java Messaging Service
• EPICS 4.5

• We currently support only DDS topics with structures similar to PVs
• The new pyDDS library, a Python application: Dynamically generate type-specific objects right inside Python application using services provided by pyDDS
• Interact with DDS services directly via pyDDS
• Does not require extra tools or separate steps to generate wrappers for every topic structure
• Take advantage of Python's dynamic language features and fit into its development flow

Summary and Future Work

A SOA with dual messaging buses have shown to be an effective approach to address the scalability and interoperability challenges of modern large-scale accelerator control systems. We are developing tools and libraries to simplify the adoption of DDS. We plan to continue to harden these tools and enhance them to support more dynamic features such as run-time type resolution and the emerging extensible type standard to make them more robust and adaptable to new application needs.

Acknowledgement: The authors wish to thank Nikoyak Malikly of Brookhaven National Lab for providing the middle-layer server examples, and Jim Kowalkoski, Marc Paterno, Kurt Biery, and Erik Gottschalk of Fermilab for discussing the needs and requirements of the projects they are working on.

Work partially supported by Tech-X Corporation and by the Office of High-Energy Physics, Office of Science, U.S. Department of Energy under Contracts #DE-FG02-08ER85043 and #DE-SC000842.