Experiences in Messaging Middleware for High-Level Control Applications

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

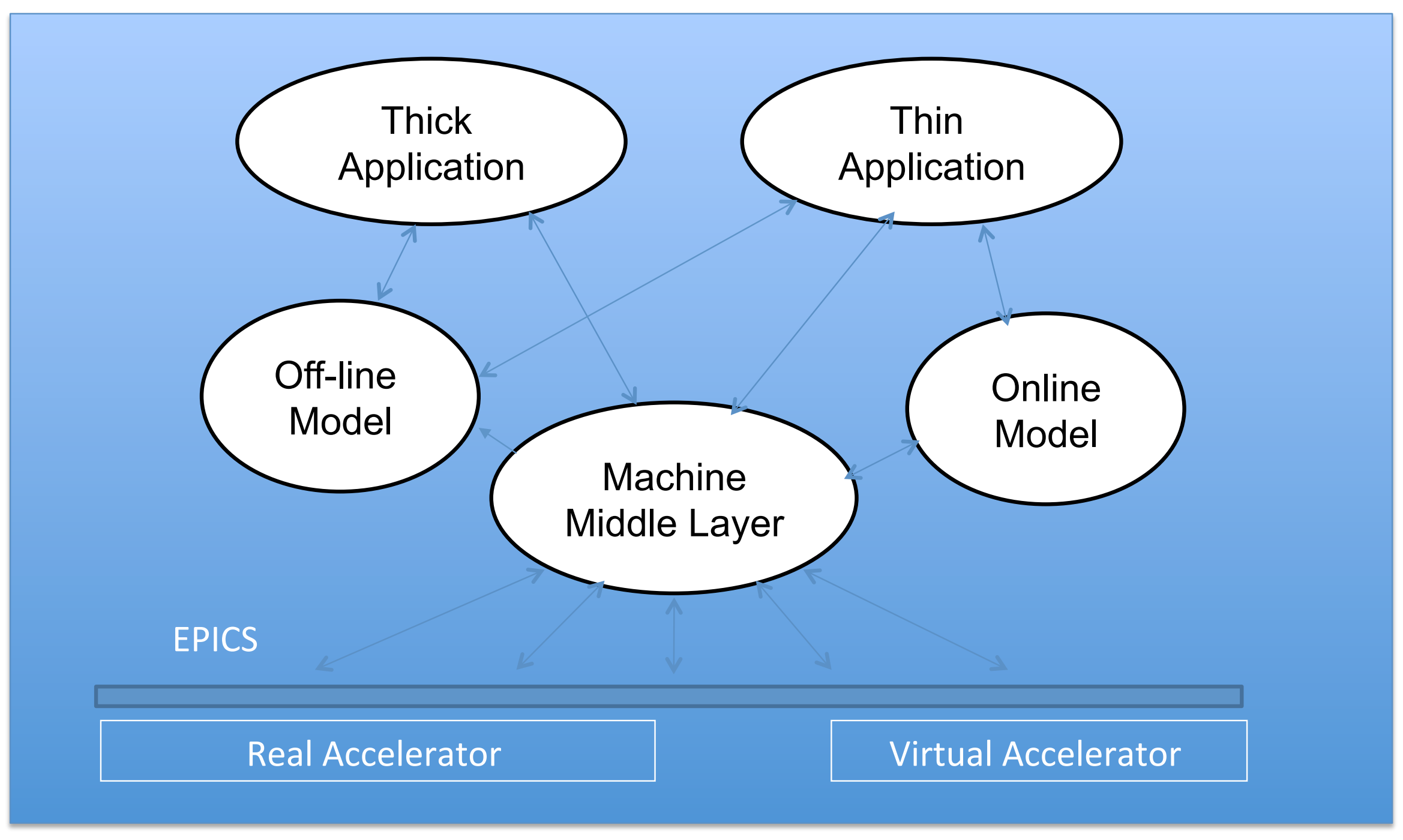
Abstract

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/data 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 v4 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.

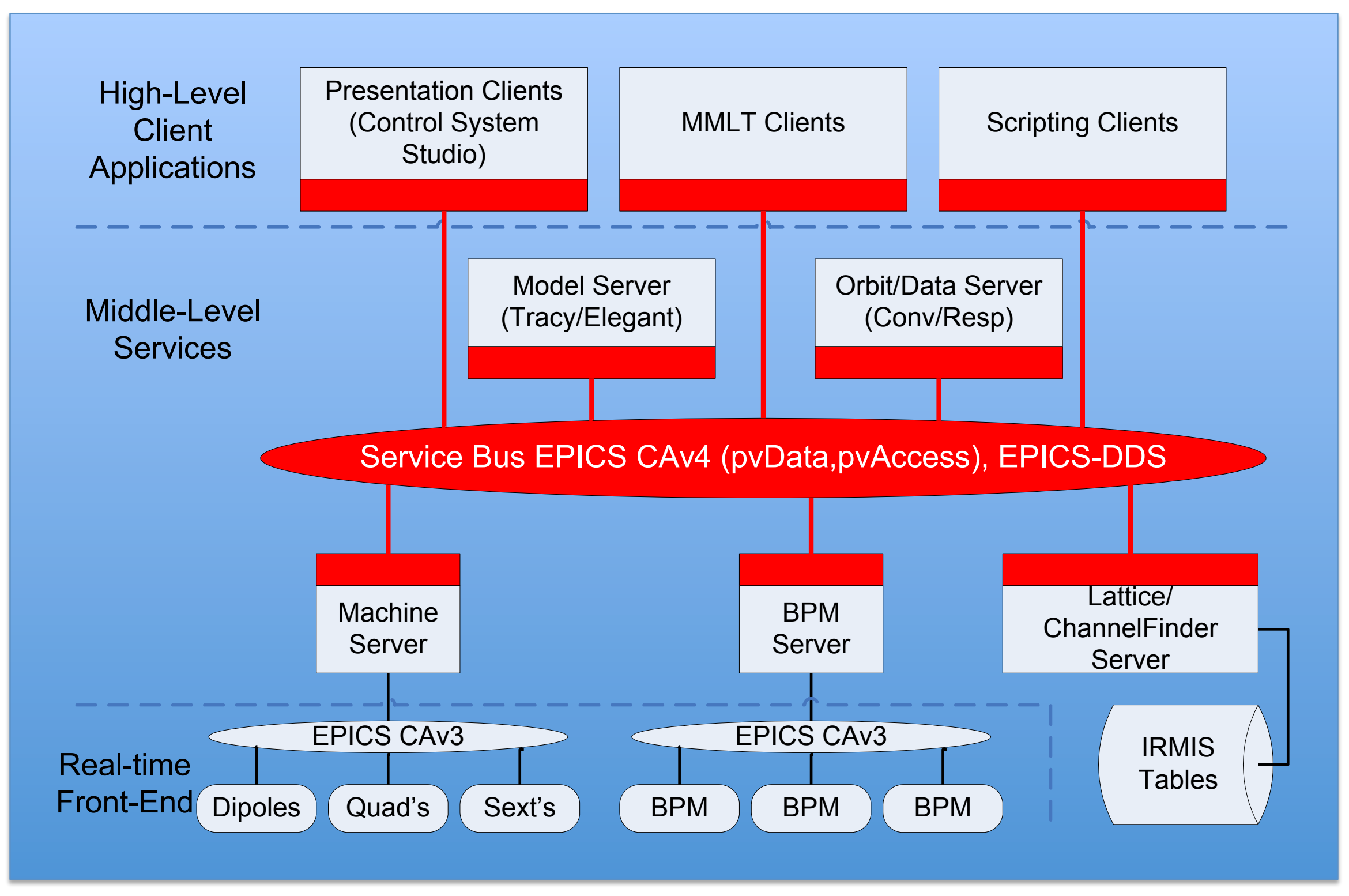
Motivations

- Modern accelerators have greatly increase complexity and scale
  - More devices and sub-devices to control, configure, monitor simultaneously
- With more and more features and automations
  - High-level client and physics applications
  - Centralized control panels for users of different roles
  - Distributed display for off-site users
- There exist many standard environments for ACS
  - EPICS, Tango, Tine, ACNET, etc.
- Limited interfacing supports
  - Hard to expand system by dynamically adding components



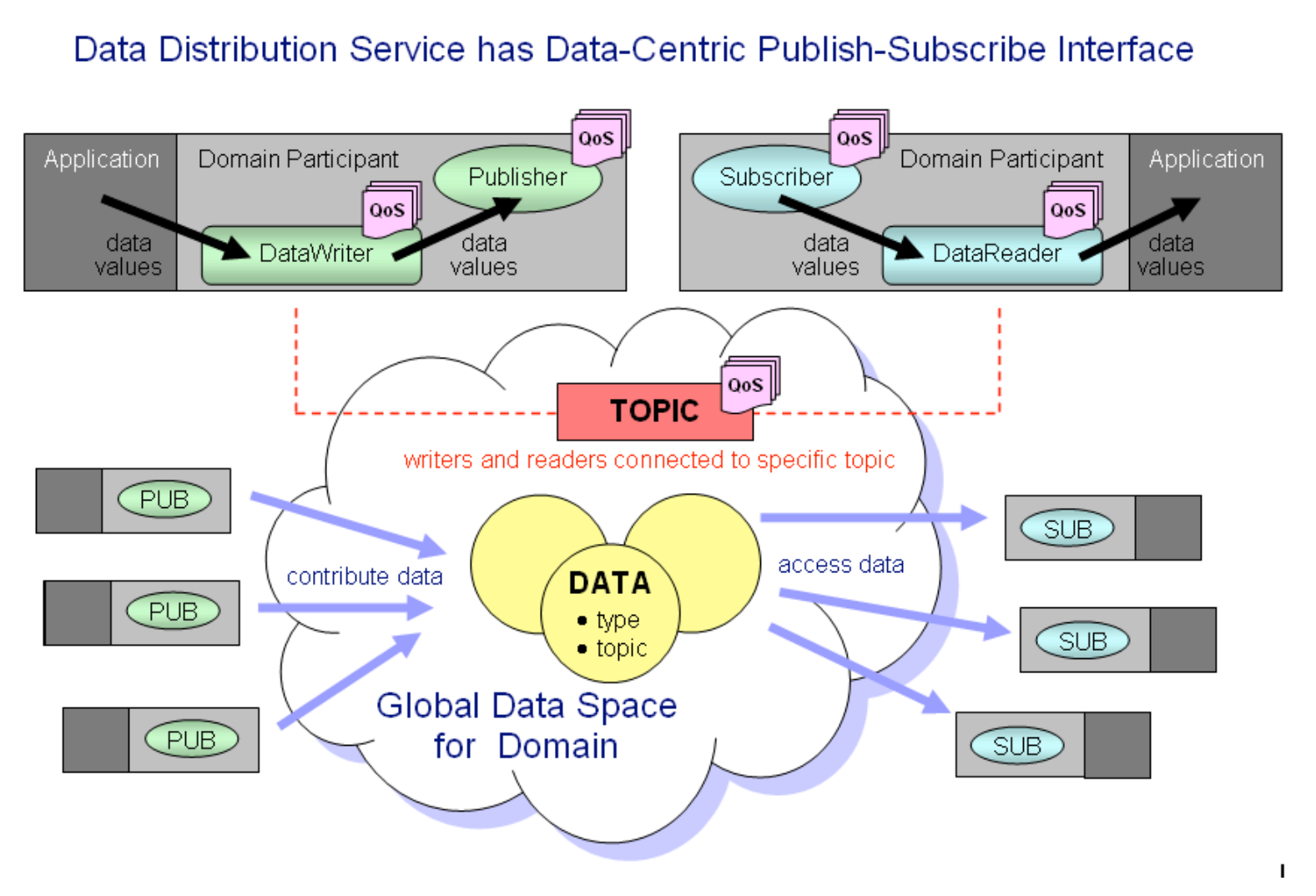
Service-Oriented Architecture (SOA) for Control Systems

- 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
- Solutions: Adding a publish/subscribe messaging protocol
  - OMG DDS
  - Java Messaging Service
  - EPICSv4 pvData, EPICS-DDS


High-Level Client Applications
Presentation Clients (Control System Studio)
MMLT Clients
Scripting Clients
Middle-Level Services
Model Server (Tracy/Elegant)
Orbit/Data Server (Conv/Resp)
Service Bus EPICS CAv4 (pvData,pvAccess), EPICS-DDS
Machine Server
BPM Server
Lattice/ ChannelFinder Server
EPICS CAv3
EPICS CAv3
Real-time Front-End
Dipoles
Quad’s
Sext’s
BPM
BPM
BPM
IRMIS Tables

Why Data Distribution Service?

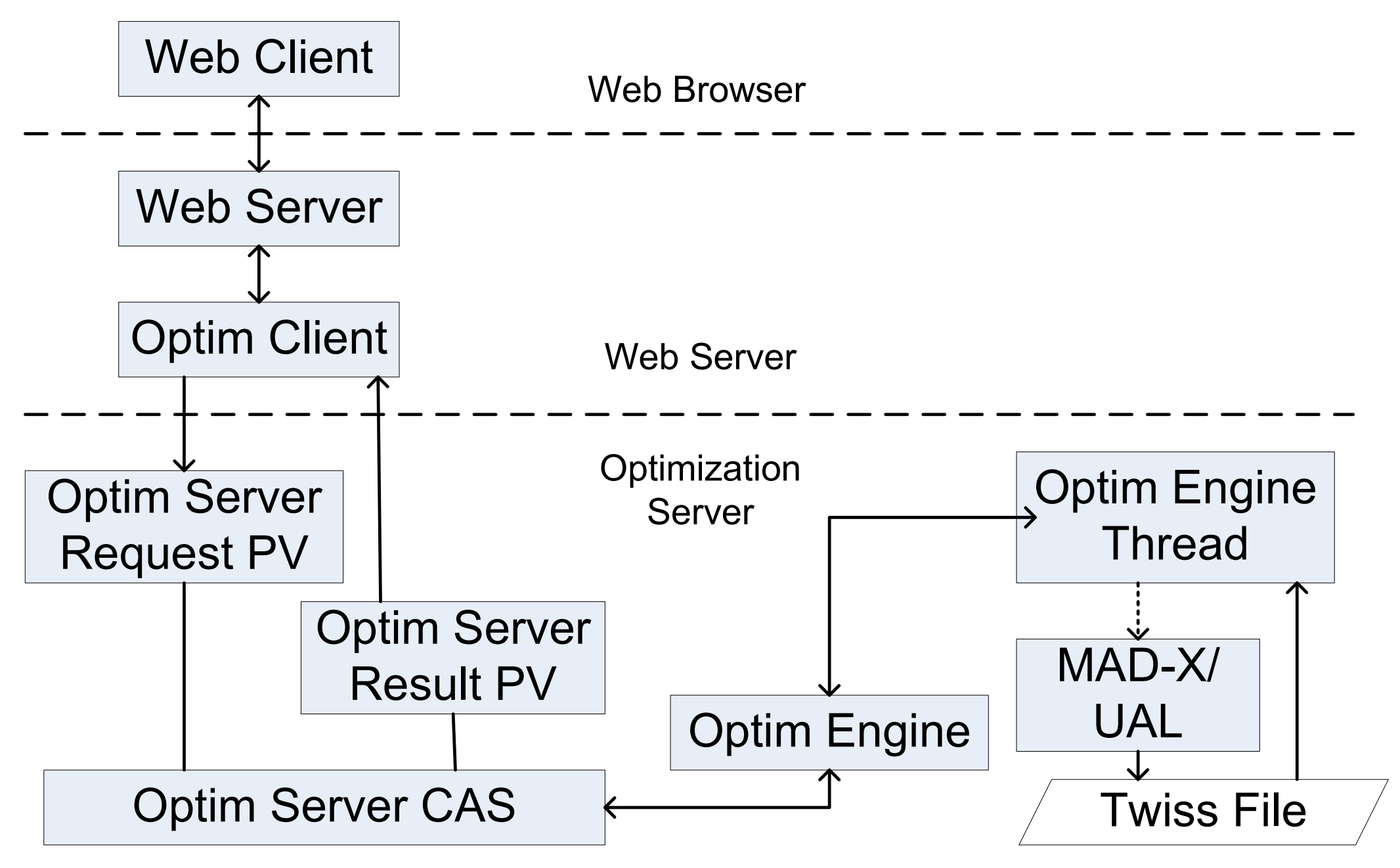
Data-centric publish-subscribe middleware provides more flexible coupling between data information producers and consumers. Furthermore, DDS has built-in Quality-of-Service policies that are crucial to control systems.


Data Distribution Service has Data-Centric Publish-Subscribe Interface
Application
Domain Participant
Publisher
QoS
DataWriter
data values
Subscriber
DataReader
TOPIC
writers and readers connected to specific topic
PUB
SUB
contribute data
access data
DATA
• type
• topic
Global Data Space for Domain

- Deadline: Establishes contract regarding rate at which periodic data is refreshed
- Latency: Establishes guidelines 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 QoS (Best-effort/reliable)
- History: Control how many messages are kept by the middleware (keep last n/keep all)
- Durability: Control the lifecycle of data (volatile, transient, persistent)

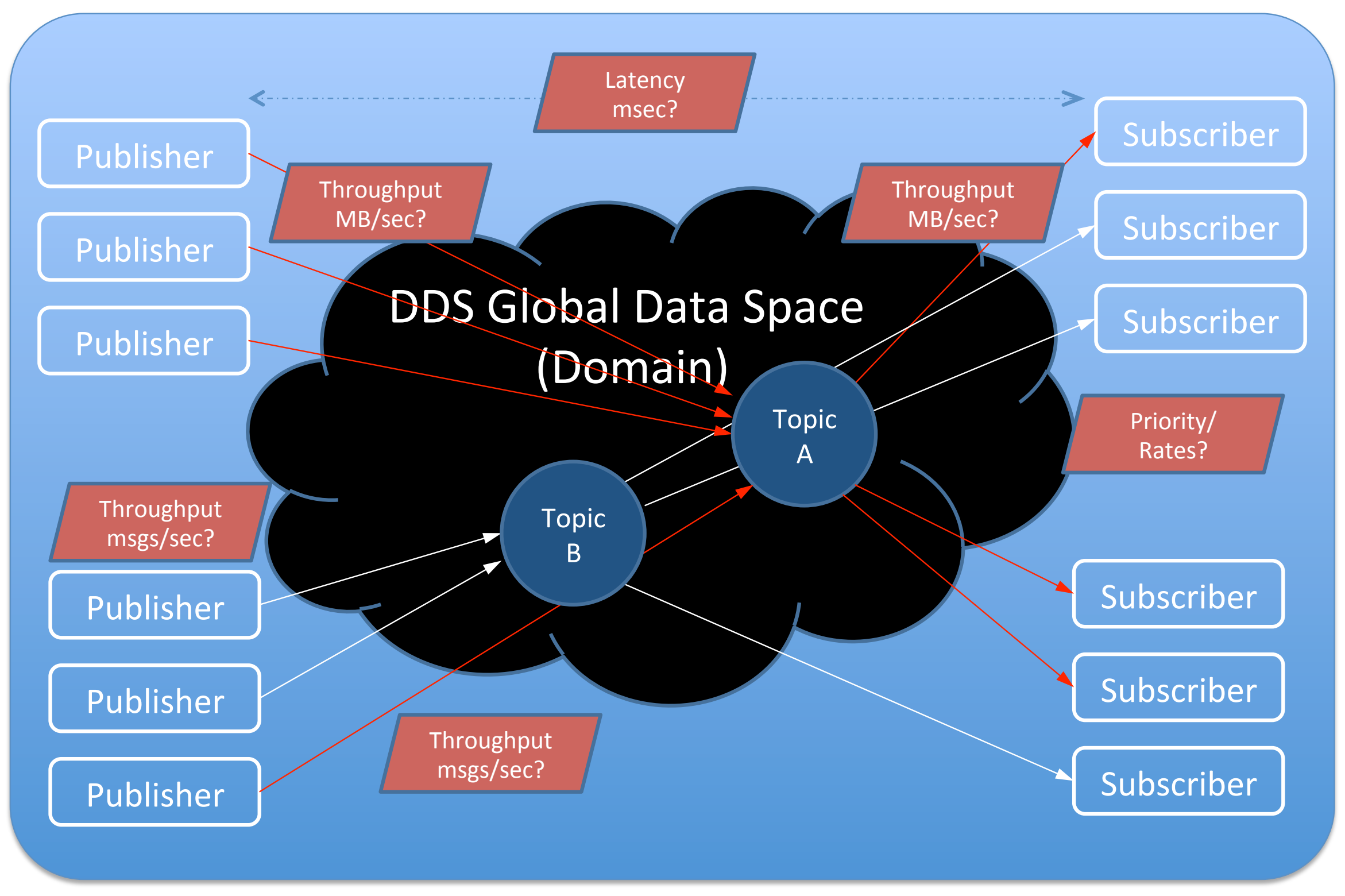
Middle-Layer Server Examples

We developed an example middle-layer server on top of two different DDS implementations:
- Web Browser runs UI
- Web Server connect to the actual Optim Server using pub/sub 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
- The 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 Intensity Frontier experiment employs both EPICS and DDS in the instrument. It is necessary to monitor information and send control signals over both protocols.

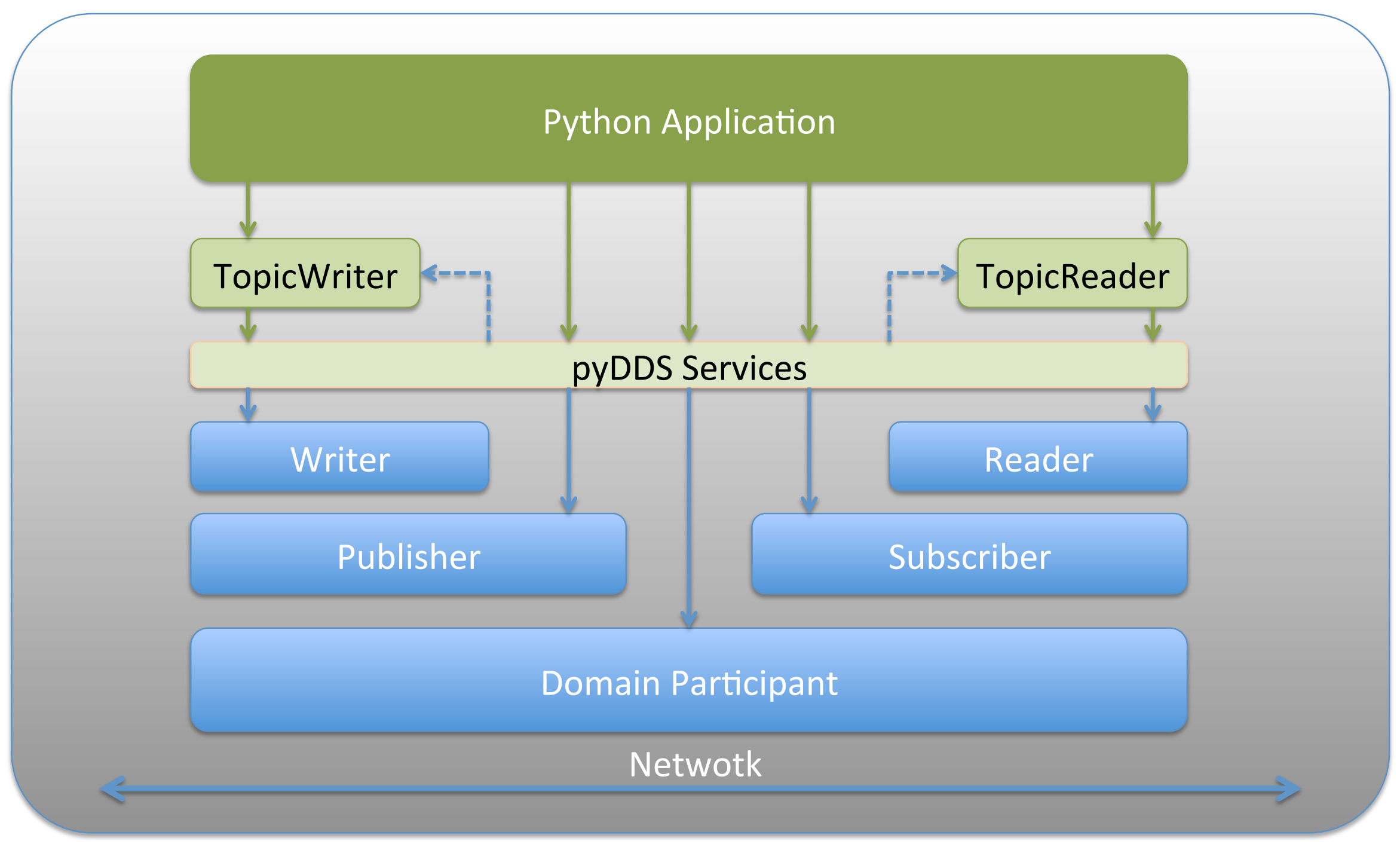
- We modeled after org.csstudio.platform.libs.epics and implemented a set of plug-ins to present DDS topics as PV’s:
  - com.txcorp.soaac.css.platform.libs.dds
  - com.txcorp.soaac.css.platform.libs.dds.ui
  - com.txcorp.soaac.css.pv.dds
- The added plug-ins enable CSS applications to subscribe/publish (get/set) a DDS topic (structured like a PV)
  - For example, an OPI widget can be associated to a PV `dds://temperature3`

Current Status:
- Topics (with their associated QoS policies) need to be precompiled into jar files and loaded thru com.txcorp.soaac.css.pv.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, `dds://Booster/Quad/Magnet2#current`
- Add support for dynamic topic/QoS 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/interpretive language nature



- With 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 wrappings for every topic structure
  - Take advantabe of Python’s dynamic language features and fit into its development flow

Joining a Data Domain

```
# A one-stop interface into the pydds global
# factory methods
import pydds;

# Uers defines their dataspace/runtime and
# pass it in as an argument to various other
# operations that need it.  (See later)
myDataspace =
   pydds.connect_dataspace
     (“Domain name”, “Partition name”)

# A Dataspace object should contains some
# default subscriber/publisher objects with
# some default QoS policies.
```

Manipulate QoS Policies

```
myQoS = pydds.create_qos()
myQoS.set_reliable (3000000)
myQoS.set_transient()
myQoS.set_keep_last (3)
```

Create Topic Reders/Writers

```
# Creating/Finding a topic in global data
# space. Last argument specifies the URI
# of topic structure definitions
helloTopic = pydds.createTopic
  (“TopicName”,
    myDataspace,
    myQoS,
    file:///HelloWorld.idl#HelloTopic);
# Now create reader/writer objects
helloReader = helloTopic.create_reader
  (readerQoS);
helloWriter = helloTopic.create_writer
  (writerQoS);
```

Writing and Reading Samples

```
# Creating a sample
helloSample = helloTopic.create_sample
  (message = “John Smith”, repeat = 3);
# Publishing the sample
status = helloWriter.write (helloSample);
# Simple read/take
[samples, infos] = helloReader.read();
sys.stdout write(samples[0].message);
```

Listen-based Read Modeled after Twist/Trellis

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.

Acknowledgment: The authors wish to thank Nikokay Malitsky 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 Corpoation and by the Office of High-Energy Physics, Office of Science, U.S. Department of Energy under Contracts #DE-FG02-08ER85043 and #DE-SC0000842

The 13th International Conference on Accelerator and Large Experimental Physics Control Systems, October, 2011