Data Distribution Service (DDS)

Tutorial

Gerardo Pardo-Castellote
Real-Time Innovations, Inc.
The DDS Standard

• Data Distribution Service for Real-Time Systems
  – Adopted in June 2003
  – Finalized in June 2004 (pending)
  – Joint submission (RTI, THALES, MITRE, OIS)
  – Specification of API required to facilitate the Data-Centric Publish-Subscribe communication environment for real-time distributed systems.
What is DDS

• **DCPS = Data Centric Publish Subscribe**
  – Purpose: Distribute the data

• **DLRL = Data Local Reconstruction Layer**
  – Purpose: provide an object-based model to access data ‘as if’ it was local

![Diagram showing Application, DLRL, and DCPS layers]
What is DDS

- **DCPS** = Data Centric Publish_Subscribe
  - Purpose: Distribute the data
- **DLRL** = Data Local Reconstruction Layer
  - Purpose: provide an object-based model to access data ‘as if’ it was local
Data Distribution Service - DCPS
What is DCPS?

- Data-Distribution for Real-Time Systems
  - Just declare your intent to publish or receive data.
  - No need to make a special request for every piece of data.

Applications just send or receive data with a standard API.

DDS does the addressing, data conversion, sending, receiving, and retries.
DDS/DCPS

• Provides a “Global Data Space” that is accessible to all interested applications.
  – Subscriptions are decoupled from Publications
  – Contracts established by means of QoS
  – Automatic discovery and configuration
Why DDS/DCPS?

- Augment existing distributed object services

Note: Most applications will need both capabilities

CORBA/
RMI

Distributed
Node

Request

Reply

Invoke
Service or
Function

DDS

Distributed
Node

Global Data Space

Distributed
Node

Distributed
Node
Benefits of DDS/DCPS

• Predictable and/or reliable data distribution with minimal overhead
  – No reply confirmation needed.
  – Reliable ↔ acyclic messages
  – QoS parameters used to set up communication determinism / reliability.

• Plug & Play
  – Dynamic, automatic discovery
The DDS/DCPS Model

• Data Domains
• Topics
• Publications & Subscriptions
  – One to One, One to Many, Many to One
  – State propagation
• Quality of Service
  – Reliability
  – Predictability
  – Fault Tolerance
  – Scalability
A DomainParticipant is the entry-point for the service and isolates a set on applications that share a physical network.
DCPS Entities

- **DomainParticipant** ~ Represents participation of the application in the communication collective
- **DataWriter** ~ Accessor to write typed data on a particular **Topic**
- **Publisher** ~ Aggregation of DataWriter objects. Responsible for disseminating information.
- **DataReader** ~ Accessor to read typed data regarding a specific **Topic**
- **Subscriber** ~ Aggregation of DataReader objects. Responsible for receiving information
Topic-based publish-subscribe

- DataWriter is bound (at creation time) to a single Topic
- DataReader is bound (at creation time) with one or more topics (Topic, ContentFilteredTopic, or MultiTopic)
- ContentFilteredTopic and MultiTopic provide means for content-based subscriptions
Topic-based publish-subscribe

Pressure
Temperature

- **DataWriter** is bound (at creation time) to a single **Topic**
- **DataReader** is bound (at creation time) with one or more topics (**Topic**, **ContentFilteredTopic**, or **MultiTopic**)
- **ContentFilteredTopic** and **MultiTopic** provide means for content-based subscriptions
Domains and Participants
Domains: Topic view

Domain 0

- Topic A
  - DW
  - DR

- Topic B
  - DW
  - DR

- Topic C
  - DW
  - DR

Domain 1

- Topic A
  - DW
  - DR

- Topic D
  - DW
  - DR

Domain 2

- Topic E
  - DW
  - DR
Pub/Sub Scenarios

One to One
One to Many
Many to One
Many to Many

Topic

Data Writer
Publisher

Data Writer
Publisher

Data Reader
Subscriber

Data Reader
Subscriber

Data Reader
Subscriber

© Copyright 2004

Real-Time Innovations – Shaping the Future of Real Time
DCPS Publication Objects

User Application:
- Creates all DCPS entities
- Configures entity QoS
- Associates DW with Topic
- Provides data to DW

Domain Participant

Topic

Data Writer

Publisher

Data Sample

S
Example: Publication

Publisher publisher = domain->create_publisher(
    publisher_qos,
    publisher_listener);

Topic topic = domain->create_topic(
    "Track", "TrackStruct",
    topic_qos, topic_listener);

DataWriter writer = publisher->create_datawriter(
    topic, writer_qos, writer_listener);
TrackStructDataWriter twriter =
    TrackStructDataWriter::narrow(writer);

TrackStruct my_track;
twriter->write(&my_track);
DCPS Subscription Listener

User Application:
- Creates all DCPS entities
- Configures entity QoS
- Associates DR with Topic
- Receives Data from DR using a Listener

Listener: read,take

S

S

Real-Time Innovations – Shaping the Future of Real Time

© Copyright 2004
Example: Subscription

```cpp
Subscriber subs = domain->create_subscriber(
    subscriber_qos, subscriber_listener);

Topic topic = domain->create_topic(
    "Track", "TrackStruct",
    topic_qos, topic_listener);

DataReader reader = subscriber->create_datareader(
    topic, reader_qos, reader_listener);

// Use listener-based or wait-based access
```
How to get data (listener-based)

Listener listener = new MyListener();
reader->set_listener(listener);

MyListener::on_data_available( DataReader reader )
{
    TrackStructSeq received_data;
    SampleInfoSeq sample_info;
    TrackStructDataReader treader =
        TrackStructDataReader::narrow(reader);

    treader->take( &received_data,
                   &sample_info, ...)

    // Use received_data
}
DCPS Subscription Wait-Set

User Application:
• Creates all DCPS entities
• Configures entity QoS
• Associates DR with Topic
• Receives Data from DR using Condition + WaitSet

Application: read, take

Domain Participant

Wait for Data

Real-Time Innovations – Shaping the Future of Real Time
How to get data (wait-based)

Condition  foo_condition =
    treader->create_readcondition(...);

waitset->add_condition(foo_condition);

ConditionSeq active_conditions;
waitset->wait(&active_conditions, timeout);
...
FooSeq received_data;
SampleInfoSeq sample_info;

treader->take_w_condition(&received_data,
    &sample_info,
    foo_condition);

// Use received_data
NDDS 4.0 Topics

- MultiTopic (Optional)
  - "Subscription Expression"
  - "Expression Params"
- ContentFiltered Topic (Optional)
  - "Filter Expression"
  - "Expression Params"
- Get Inconsistent Topic Status
  - "TypeName"
  - "Name"
Keys (data-object identification)

- Multiple instances of the same topic
  - Used to identify specific instances
  - Do not need a separate Topic for each data-object instance

Topic key can be any data-type within the Topic.

Data Reader

Subscriber

Temp sensor 1

Temp sensor 2

Temp sensor 3
DCPS Content Filtered Topics

Topic Instances in Domain

- Instance 1: Value = 249
- Instance 2: Value = 230
- Instance 3: Value = 275
- Instance 4: Value = 262
- Instance 5: Value = 258
- Instance 6: Value = 261
- Instance 7: Value = 259

Optional

The Filter Expression and Expression Params will determine which instances of the Topic will be received by the subscriber.

Real-Time Innovations – Shaping the Future of Real Time
DCPS Subscription Objects (MultiTopic)

MultiTopics can combine, filter and rearrange data from multiple topics.

Optional

** Listeners Wait-Set or conditions available
QoS Contract “Request / Offered”

QoS Request / Offered:
Ensure that the compatible QoS parameters are set.

QoS: Durability
QoS: Presentation
QoS: Deadline
QoS: Latency_Budget
QoS: Ownership
QoS: Liveliness
QoS: Reliability

Communication not established

QoS not compatible
QoS: Reliability

**BEST_EFFORT**
- Sample delivery is not guaranteed

**RELIABLE**
- Sample delivery is guaranteed

Publisher
- Data Writer:
  - Sample S1
  - Sample S2
  - Sample S3
  - Sample S4
  - Sample S5
  - Sample S6
  - Sample S7

Publisher
- Data Writer (BE):
- Data Reader (BE)

Subscriber
- Data Reader (R)

Missed samples:
- S1
- S2
- S3
- S4
- S5
- S6
- S7

© Copyright 2004
QoS: History: Last x or All

**KEEP_LAST**: "depth" integer for the number of samples to keep at any one time

**KEEP_ALL**: Publisher: keep all until delivered
Subscriber: keep each sample until the application processes that instance
State propagation

- **System state**
  - Information needed to describe future behavior of the system
    - System evolution defined by state and future inputs.
  - Minimalist representation of past inputs to the system
- **State variables**
  - Set of data-objects whose value codifies the state of the system
- **Relationship with DDS**
  - DDS well suited to propagate and replicate state
  - Topic+key can be used to represent state variables
  - KEEP_LAST history QoS exactly matches semantics of state-variable propagation
- **Significance**: Key ingredient for fault-tolerance and also present in many RT applications
QoS: Deadline

- **Topic**: Commits to provide data each deadline period.
- **Publisher**: Data Writer
- **Subscriber**: Data Reader
- **Listener**: Failed to get data

**DEADLINE “deadline period”**

[Diagram showing the relationship between the components and the deadline period]
QoS: Liveliness – Type, Duration

Type:
AUTOMATIC = Infrastructure Managed
MANUAL = Application Managed

Failed to renew lease

Publisher
Data Writer

Subscriber
Data Reader

Listener

Topic

Domain Participant

Domain Participant

lease_duration

Liveliness Message

Real-Time Innovations – Shaping the Future of Real Time
QoS: Time_Based_Filter

“minimum_separation”: Data Reader does not want to receive data faster than the min_separation time.
QoS: Ownership

Ownership: Specifies whether more than one Data Writer can update the same instance of a data-object

Ownership = EXCLUSIVE
• Only highest-strength data writer can update data-instance

Ownership = SHARED
• All data-writers can update data-instance
QoS: OwnershipStrength

Ownership Strength: Specifies which writer is allowed to update the values of data-objects

OWNERSHIP_STRENGTH
• Integer to specify the strength of an instance

Note: Only applies to Topics with Ownership=Exclusive
QoS: Destination Order

**BY_RECEPTION_TIMESTAMP**
- Order is determined by the DR timestamp

**BY_SOURCE_TIMESTAMP**
- Consistent order for delivery is guaranteed

Absolute order of writes:
S4 T3 S3 T2 S2 T1 S1

Order of updates as received by subscribers:
S4 T3 T2 S3 T1 S2 S1

Real-Time Innovations – Shaping the Future of Real Time
QoS: Latency_Budget

- Intended to provide time-critical information to the publisher for framework tuning where possible.
- Will not prevent data transmission and receipt.

Latency = t1 + t2 + t3
QoS: Resource_Limits

• Specifies the resources that the Service can consume to meet requested QoS

max_samples_per_instance: max # data samples per instance

max_instances: max # instances for a single DW or DR

max_samples: max # data samples for a single DW or DR, across all instances
QoS: USER_DATA

Definition: User-defined portion of Topic metadata

Example: Security Authentication

Entity:
Domain Participant (user_data)
DataReader (user_data)
DataWriter (user_data)

User data can be used to authenticate an origination entity.

Remote Application
Authenticate Origin
Accept Entities

yes
no

ignore_participant()
ignore_publication()
ignore_subscription()
ignore_topic()

Note: USER_DATA is contained within the DDS metadata.
QoS: Partition

Partition: Logical “namespace” for topics

Topic A
Topic B
Topic C

Domain Participant

Data Writer
Data Writer
Data Writer

Publisher
Partition U,W

Data Reader
Data Reader
Data Reader

Subscriber
Partition U,Z

Data Reader

Subscriber
Partition X,Y

** Partition string names must match between publisher and subscriber
QoS: Durability

Durability determines if/how instances of a topic are saved.

Durability Kind:
- VOLATILE – No Instance History Saved
- TRANSIENT – History Saved in Local Memory
- PERSISTENT – History Saved in Permanent storage

# saved in Transient affected by QoS: History and QoS: Resource_Limits
## QoS: Quality of Service

### Table: QoS Policy

<table>
<thead>
<tr>
<th>QoS Policy</th>
<th>Concerns</th>
<th>RxO</th>
<th>Changeable</th>
</tr>
</thead>
<tbody>
<tr>
<td>USER_DATA</td>
<td>DP,DR,DW</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>DURABILITY</td>
<td>T,DR,DW</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>PRESENTATION</td>
<td>P,S</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>DEADLINE</td>
<td>T,DR,DW</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>LATENCY_BUDGET</td>
<td>T,DR,DW</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>OWNERSHIP</td>
<td>T</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>OWNERSHIP_STRENGTH</td>
<td>DW</td>
<td>N/A</td>
<td>YES</td>
</tr>
<tr>
<td>LIVELINESS</td>
<td>T,DR,DW</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>TIME_BASED_FILTER</td>
<td>DR</td>
<td>N/A</td>
<td>YES</td>
</tr>
<tr>
<td>PARTITION</td>
<td>P,S</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>RELIABILITY</td>
<td>T,DR,DW</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>TRANSPORT_PRIORITY</td>
<td>T,DW</td>
<td>N/A</td>
<td>YES</td>
</tr>
<tr>
<td>DESTINATION_ORDER</td>
<td>T,DR</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>HISTORY</td>
<td>T,DR,DW</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>RESOURCE_LIMITS</td>
<td>T,DR,DW</td>
<td>NO</td>
<td>NO</td>
</tr>
</tbody>
</table>

**RxO:**

**Request / Offered**
QoS: Presentation

• Governs how related data-instance changes are presented to the subscribing application.

• Type: Coherent Access and Ordered Access
  - Coherent access: All changes (as defined by the Scope) are presented together.
  - Ordered access: All changes (as defined by the Scope) are presented in the same order in which they occurred.

• Scope: Instance, Topic, or Group
  - Instance: The scope is a single data instance change. Changes to one instance are not affected by changes to other instances or topics.
  - Topic: The scope is all instances by a single Data Writer.
  - Group: The scope is all instances by Data Writers in the same Subscriber.
DDS-DCPS Conclusion

- DDS-DCPS targets applications that need to distribute data in a real-time environment
- DDS-DCPS provides a shared “global data space” where any application can publish the data it has & subscribe to the data it needs
- DDS-DCPS provides automatic discovery (plug & play) and facilities building fault-tolerant systems
- DDS-DCPS is highly configurable by means of QoS settings. Heterogeneous systems can be easily accommodated
Thank you for your time today.

gerardo@rti.com

http://www.rti.com