DDS

Data Distribution Service

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

Data Distribution Service for Real-Time Systems

• Adopted in June 2003
• Finalized in June 2004
• Joint submission (RTI, THALES, MITRE, OIS)
• API specification for Data-Centric Publish-Subscribe communication for distributed real-time systems.

RTI’s role

• Member of OMG since 2000
• Co-authors of the original DDS RFP
• Co-authors of the DDS specification adopted in June 2003
• Chair of the DDS Finalization Task Force completed March 2004
• Chair of the DDS Revision Task Force
• Providers of a COTS implementation of the specification (NDDS.4.0)
OMG Middleware standards

**CORBA**

* Distributed object
  - Client/server
  - Remote method calls
  - Reliable transport

* Best for
  - Remote command processing
  - File transfer
  - Synchronous transactions

**DDS**

* Distributed data
  - Publish/subscribe
  - Multicast data
  - Configurable QoS

* Best for
  - Quick dissemination to many nodes
  - Dynamic nets
  - Flexible delivery requirements

**DDS and CORBA address different needs**

Complex systems often need both…
More Complex Distributed Application

- New nodes are not dynamically “Discovered”
- Socket connections needed for each path
- Future upgrades require “re-design”
- App SW must perform endian conversion
The net-centric vision

Vision for “net-centric applications”
Total access to information for real-time applications
This vision is enabled by the internet and related network technologies

Challenge:
“Provide the right information at the right place at the right time… no matter what.”
Challenges: Factors driving DDS

Need for speed
- Large networks, multicast
- High data rates
- Natural asynchrony
- Tight latency requirements
- Continuously-refreshed data

Complex data flows
- Controlled QoS: rates, reliability, bandwidth
- Per-node, or per-stream differences
- Varied transports (incl. Unreliable e.g. wireless)

Dynamic configurations
- Fast location transparency

Fault tolerance
- No single-points of failure
- Transparent failover
DDS

Provides a “Global Data Space” that is accessible to all interested applications.

- Data objects addressed by **Topic** and **Key**
- Subscriptions are **decoupled** from Publications
- Contracts established by means of **QoS**
- Automatic **discovery** and configuration
Data object addressing: Keys

Address in Global Data Space = (Topic, Key)

Multiple instances of the same topic

- Used to sort specific instances
- Do not need a separate Topic for each data-object instance

Topic key can be any field within the Topic.

Example:
```
struct LocationInfo {
    int LocID; //key
    GPSPos pos;
};
```
Publisher declares information it has and specifies the Topic
  • … and the offered QoS contract
  • … and an associated listener to be alerted of any significant status changes

Subscriber declares information it wants and specifies the Topic
  • … and the requested QoS contract
  • … and an associated listener to be alerted of any significant status changes

DDS automatically discovers publishers and subscribers
  • DDS ensures QoS matching and alerts of inconsistencies
**DCPS Entities**

- **Publisher**
- **DomainParticipant**
- **Subscriber**

**DataWriter**

**Publisher**

**DataReader**

**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
Domains and Participants
**User Application:**

- Creates all DDS entities
- Configures entity QoS
- Associates DW with Topic
- Provides data to DW
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);
**DDS Subscription Listener**

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

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)

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

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

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

    // Use received_data
}
```
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

QoS not compatible

Communication not established
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

Data Writer Keep Last 2
- Publisher: S1, S2, S3, S4, S5, S6, S7

Data Reader Keep All
- Subscriber: S3, S4, S5, S6, S7

Data Reader Keep Last 4
- Subscriber: S4, S5, S6, S7

Data Writer Keep All
- Publisher: S1, S2, S3, S4, S5, S6, S7
**QoS: Deadline**

- **Publisher**
  - Data Writer
  - Commits to provide data each deadline period.

- **Subscriber**
  - Data Reader
  - Expects data every deadline period.

- **Listener**
  - Failed to get data

**DEADLINE “deadline period”**
**QoS: Liveliness – Type, Duration**

- **Type:**
  - AUTOMATIC = Infrastructure Managed
  - MANUAL = Application Managed

- **Failed to renew lease**

- **Domain Participant**

- **Topic**

- **Data Writer**
  - Publisher

- **Data Reader**
  - Subscriber

- **Listener**

- **Lease Duration**
  - lease_duration

- **Liveliness Message**
QoS: Time_Based_Filter

“minimum_separation”: Data Reader does not want to receive data faster than the min_separation time.
### QoS: Quality of Service (1/2)

<table>
<thead>
<tr>
<th>QoS Policy</th>
<th>Concerns</th>
<th>RxO</th>
<th>Changeable</th>
</tr>
</thead>
<tbody>
<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>READER DATA LIFECYCLE</td>
<td>DR</td>
<td>N/A</td>
<td>YES</td>
</tr>
<tr>
<td>WRITER DATA LIFECYCLE</td>
<td>DW</td>
<td>N/A</td>
<td>YES</td>
</tr>
<tr>
<td>TRANSPORT PRIORITY</td>
<td>T,DW</td>
<td>N/A</td>
<td>YES</td>
</tr>
<tr>
<td>LIFESPAN</td>
<td>T,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>RELIABILITY</td>
<td>T,DR,DW</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>DESTINATION ORDER</td>
<td>T,DR</td>
<td>NO</td>
<td>NO</td>
</tr>
</tbody>
</table>
### QoS: Quality of Service (2/2)

<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>YES</td>
</tr>
<tr>
<td>TOPIC DATA</td>
<td>T</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>GROUP DATA</td>
<td>P, S</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>ENTITY FACTORY</td>
<td>DP, P, S</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>PRESENTATION</td>
<td>P, S</td>
<td>YES</td>
<td>NO</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>PARTITION</td>
<td>P, S</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>DURABILITY</td>
<td>T, DR, DW</td>
<td>YES</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>
Summary

**DDS targets applications that need to distribute data in a real-time environment**

**DDS is highly configurable by QoS settings**

**DDS provides a shared “global data space”**
- Any application can publish data it has
- Any application can subscribe to data it needs
- Automatic discovery
- Facilities for fault tolerance
- Heterogeneous systems easily accommodated
Thank you

References:

OMG DDS specification:
http://www.omg.org/cgi-bin/doc?ptc/04-04-12

General material on DDS and RTI’s implementation:
http://www.rti.com/dds

Comments/questions: gerardo@rti.com