DDS

Data Distribution Service

Tutorial

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

Agenda

Part I. DDS Overview

- Background
- Communication model
- DDS Entities
- Listeners, Conditions, WaitSets
- Quality of Service

Part II. DDS in Action

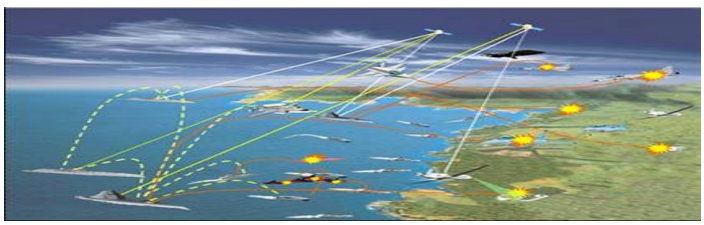
- Examples and Applications
- Topic management
- Data visibility and access control
- Application patterns
- Logging and monitoring
- Legacy applications

Part I DDS Overview

Background
Communication model
DDS Entities
Listeners, Conditions, WaitSets
Quality of Service

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 implementing vision

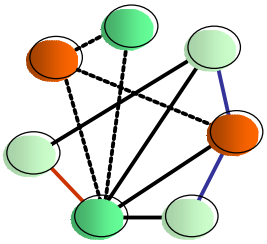
Computers are connected by a variety of "transports"

- Busses (e.g. VME)
- LANs (e.g. ethernet)
- Wireless links

Applications use communication endpoints to send/receive information

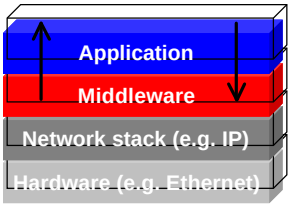
The overall application is a complex dynamic "web" of logical inter-connections

- OK for loose information aggregations World Wide Web
- Extremely challenging for real-time, or safety-critical systems



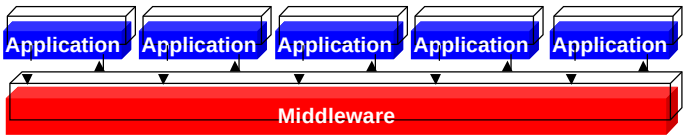
➔ **Middleware must be used to isolate applications from communications infrastructure**

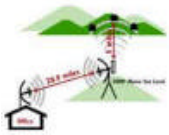
Middleware



Network middleware: A library between the operating system and the application

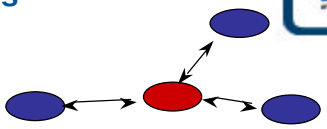
It insulates application from the raw network and provides an easier way to communicate





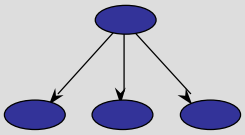
Middleware information Models

Point-to-Point
Telephone, TCP
Simple, high-bandwidth
Leads to stove-pipe systems

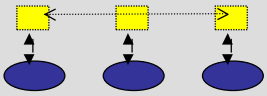


Client-Server
File systems, Database, RPC, CORBA, DCOM
Good if information is naturally centralized
Single point failure, performance bottlenecks

DDS



Publish/Subscribe Messaging
Magazines, Newspaper, TV
Excels at *many-to-many communication*
Excels at distributing *time-critical information*



Replicated Data
Libraries, Distributed databases
Excels at data-mining and analysis

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…

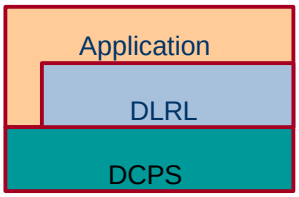
What is DDS

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



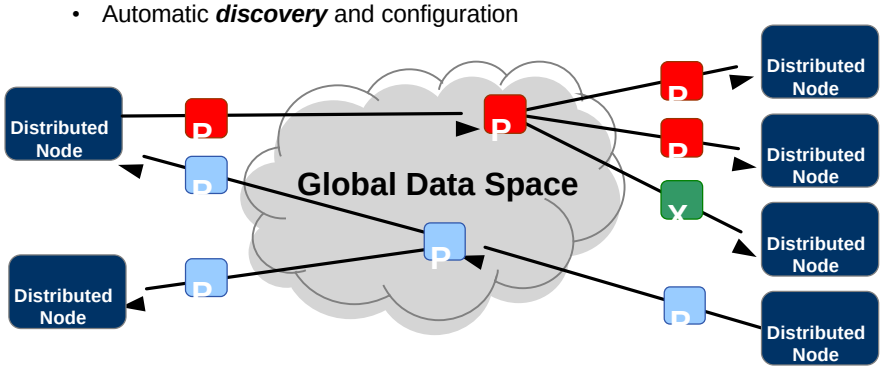
Part I DDS Overview

- Background
- → Communication model
- DDS Entities
- Listeners, Conditions, WaitSets
- Quality of Service

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

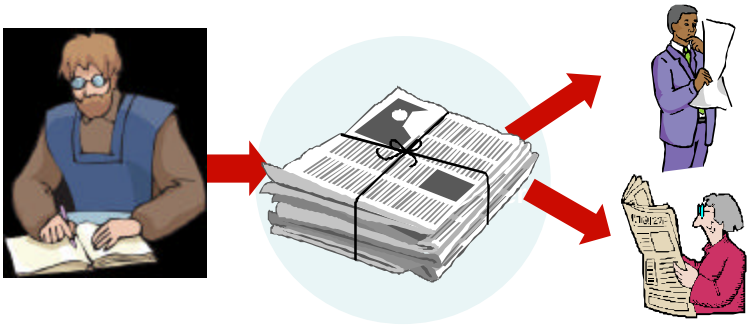


Publish Subscribe Model

- *Efficient mechanism for data communications*

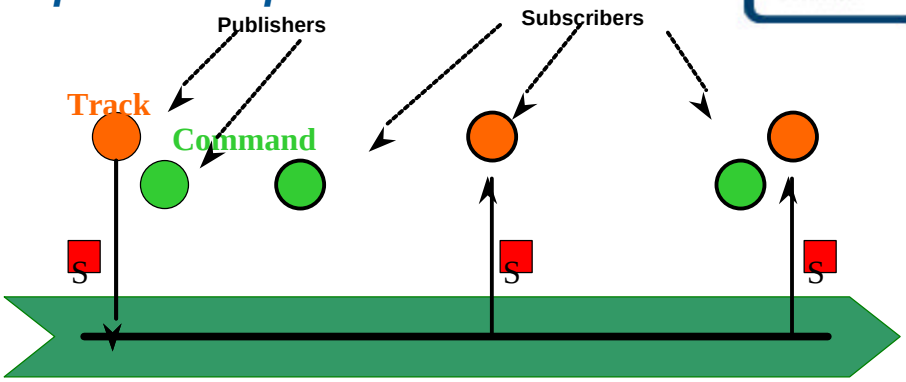
Reporter does not need to know where subscribers live.

Subscribers do not need to know where reporter lives.

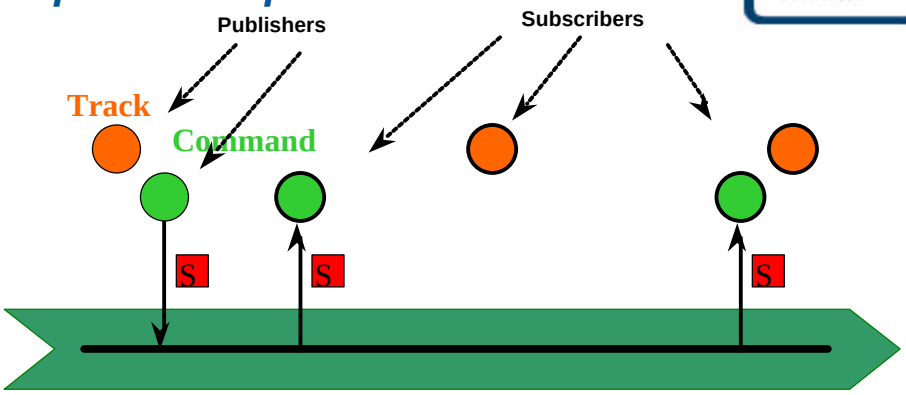


Data Producer | Middleware | Consumers

Topic-based publish-subscribe



Topic-based publish-subscribe



Publish-subscribe allows infrastructure to prepare itself…

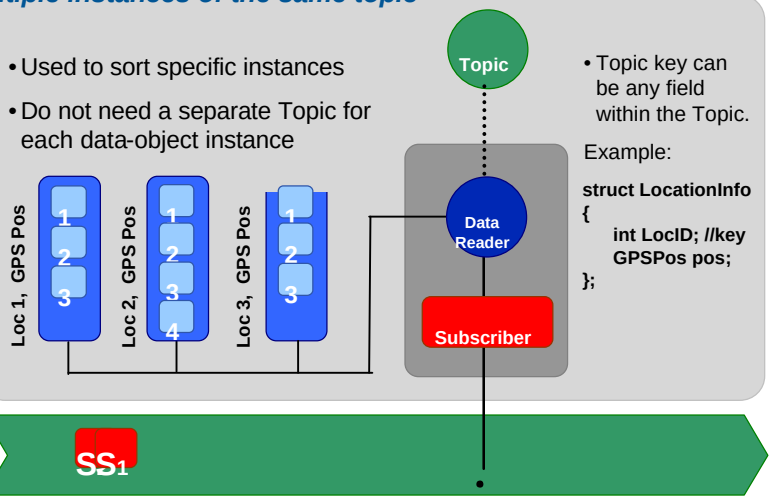
… Such that when the data is written it is directly sent to the subscribers

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

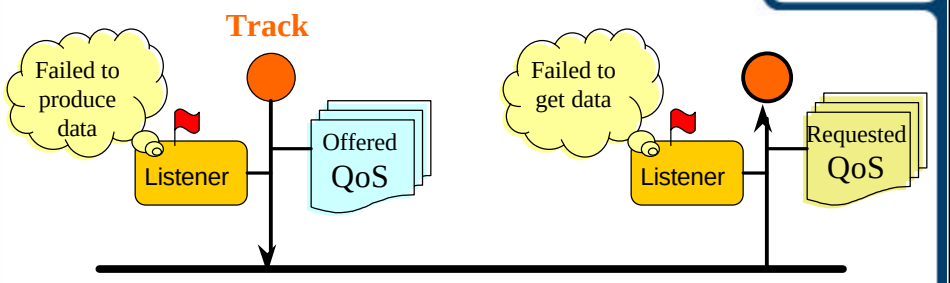


- Topic key can be any field within the Topic.

Example:

```
struct LocationInfo
{
    int LocID; //key
    GPSPos pos;
};
```

DDS communications model



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

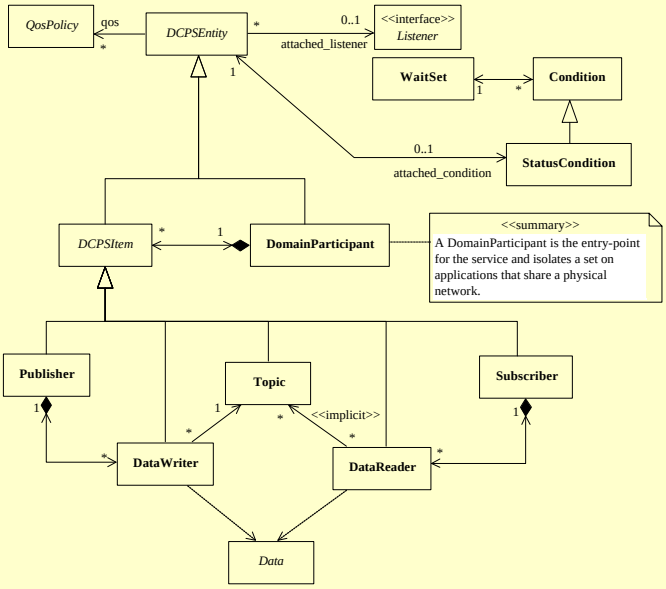
DDS automatically discovers publishers and subscribers

- DDS ensures QoS matching and alerts of inconsistencies

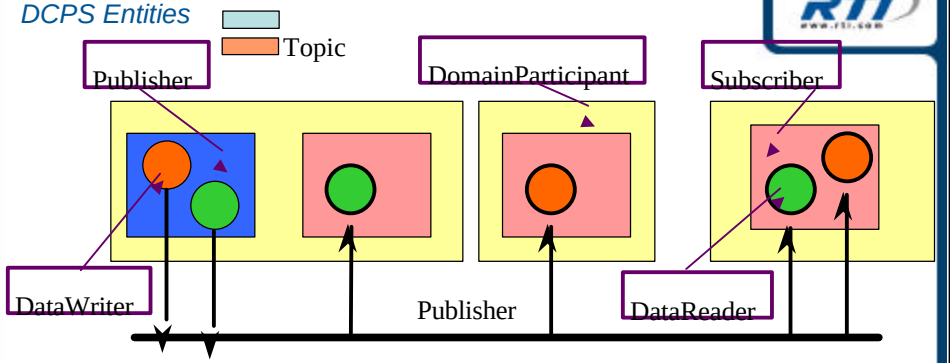
Part I DDS Overview

Background
Communication model
DDS Entities
Listeners, Conditions, WaitSets
Quality of Service

PIM Overview



DCPS Entities



DomainParticipant ~ ***Represents participation of the application in the communication collective***

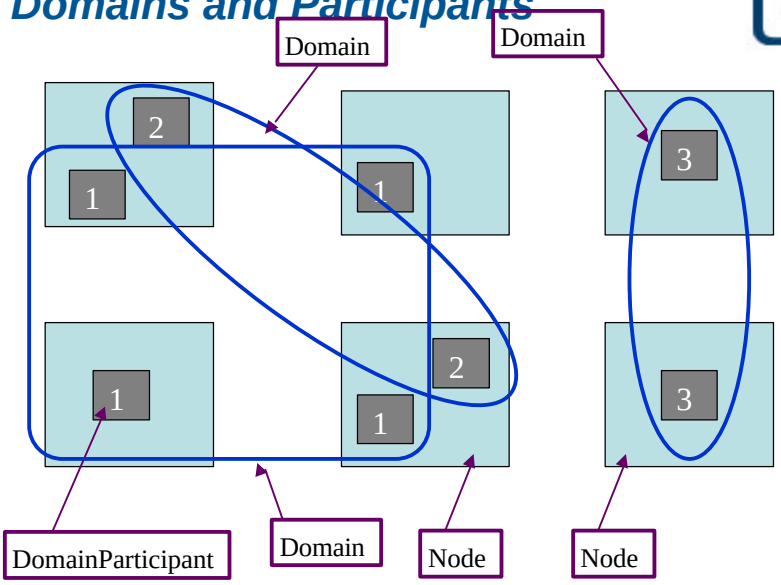
DataWriter ~ ***Accessor to write typed data on a particular*** *Topic*

Publisher ~ ***Aggregation of DataWriter objects. Responsible for disseminating information.***

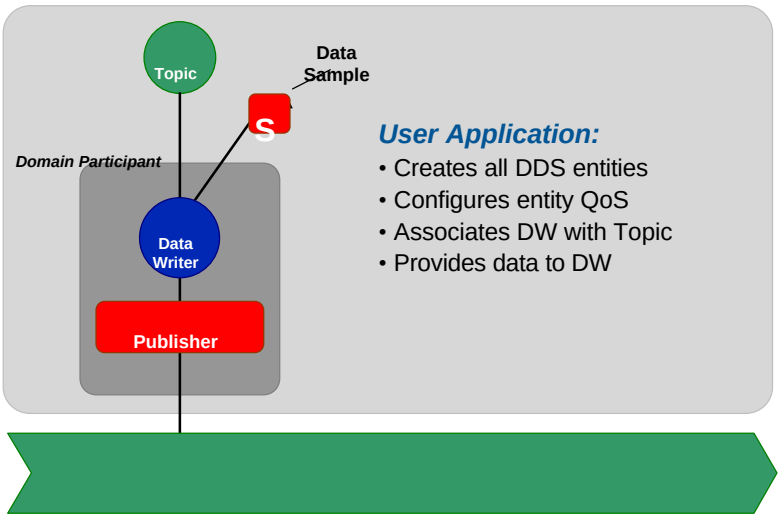
DataReader ~ ***Accessor to read typed data regarding a specific*** *Topic*

Subscriber ~ ***Aggregation of DataReader objects. Responsible for receiving information***

Domains and Participants



DDS Publication



User Application:

- Creates all DDS entities
- Configures entity QoS
- Associates DW with Topic
- Provides data to DW

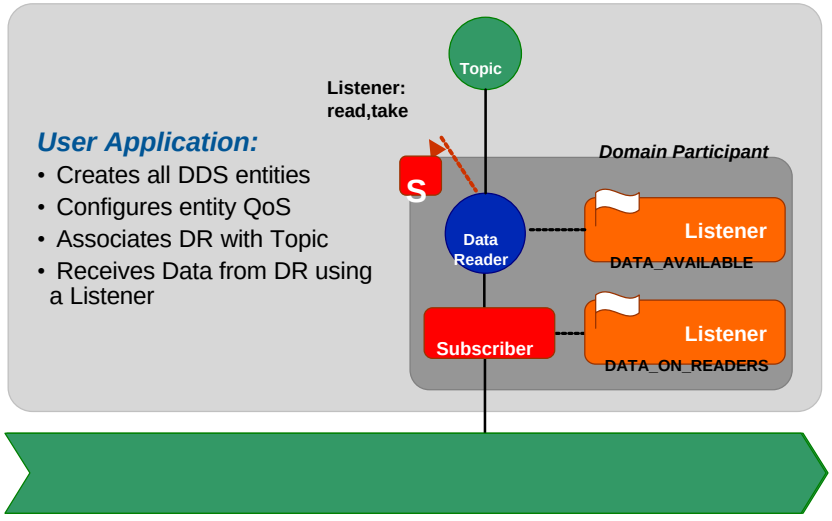
Example: Publication

```
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



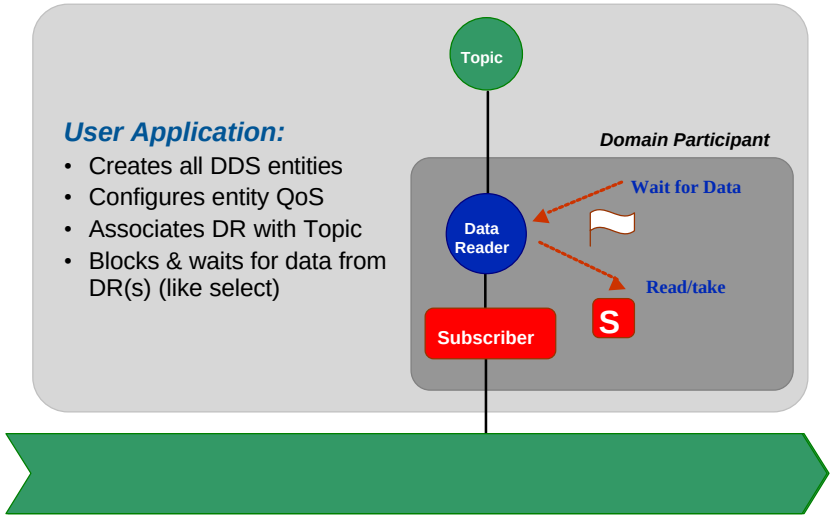
Example: Subscription

```
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)

```
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
}
```

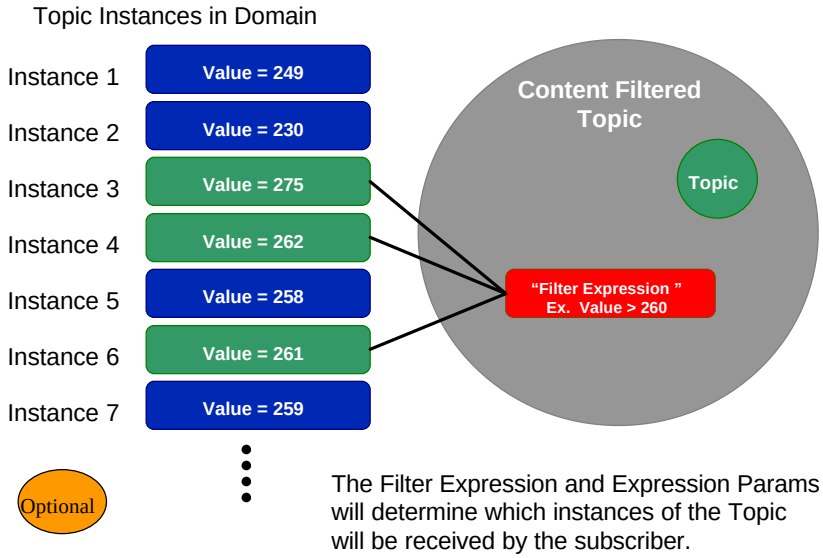
DDS Subscription Wait-Set



How to get data (wait-based)

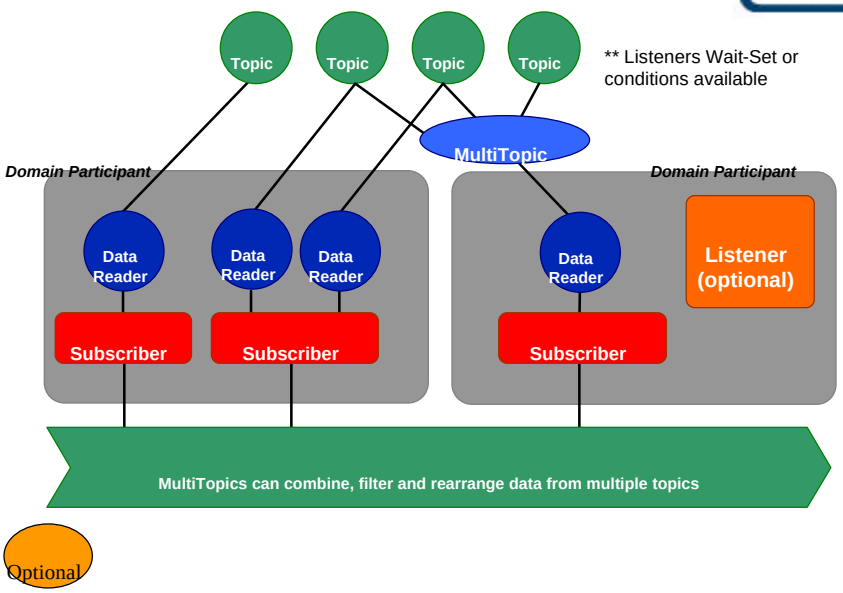
```
Condition  foo_condition =
   treader->create_readcondition(…);

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
```

DDS Content Filtered Topics



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

DDS Subscription Objects (MultiTopic)



Part I DDS Overview

- Background
- Communication model
- DDS Entities
- → Listeners, Conditions, WaitSets
- Quality of Service

Listeners, Conditions & WaitSets

Middleware must notify user application of relevant events

- Arrival of data
- QoS violations
- Discovery of relevant entities
- These events may be detected asynchronously by the middleware
- … Same issue arises with POSIX signals…

DDS allows the application a choice:

- Either get notified asynchronously using a Listener
- Or wait synchronously using a WaitSet

Both approaches are unified using STATUS changes

Status changes

DDS defines

- A set of enumerated STATUS
- The statuses relevant to each kind of DDS Entity

A DDS entity maintains a value for each of related STATUS

STATUS	Entity
INCONSISTENT_TOPIC	Topic
DATA_ON_READERS	Subscriber
LIVELINESS_CHANGED	DataReader
REQUESTED_DEADLINE_MISSED	DataReader
RUQESTED_INCOMPATIBLE_QOS	DataReader
DATA_AVAILABLE	DataReader
SAMPLE_LOST	DataReader
SUBSCRIPTION_MATCH	DataReader
LIVELINESS_LOST	DataWriter
OFFERED_INCOMPATIBLE_QOS	DataWriter
PUBLICATION_MATCH	DataWriter

```
struct LivelinessChangedStatus {
    long active_count;
    long inactive_count;
    long active_count_change;
    long inactive_count_change;
}
```


Listeners, Conditions and Statuses

A DDS Entity is associated with

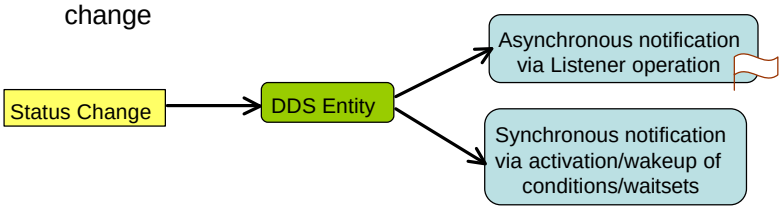
- A listener of the proper kind (if activated)
- A StatusCondition (if activated)

The Listener for an Entity has a separate operation for each of the relevant statuses

STATUS	Entity	Listener operation
INCONSISTENT_TOPIC	Topic	on_inconsistent_topic
DATA_ON_READERS	Subscriber	on_data_on_readers
LIVELINESS_CHANGED	DataReader	on_liveliness_changed
REQUESTED_DEADLINE_MISSED	DataReader	on_requested_deadline_missed
RUQESTED_INCOMPATIBLE_QOS	DataReader	on_requested_incompatible_qos
DATA_AVAILABLE	DataReader	on_data_available
SAMPLE_LOST	DataReader	on_sample_lost
SUBSCRIPTION_MATCH	DataReader	on_subscription_match
LIVELINESS_LOST	DataWriter	on_publication_match
OFFERED_INCOMPATIBLE_QOS	DataWriter	on_offered_incompatible_qos
PUBLICATION_MATCH	DataWriter	on_publication_match

Listeners & Condition duality

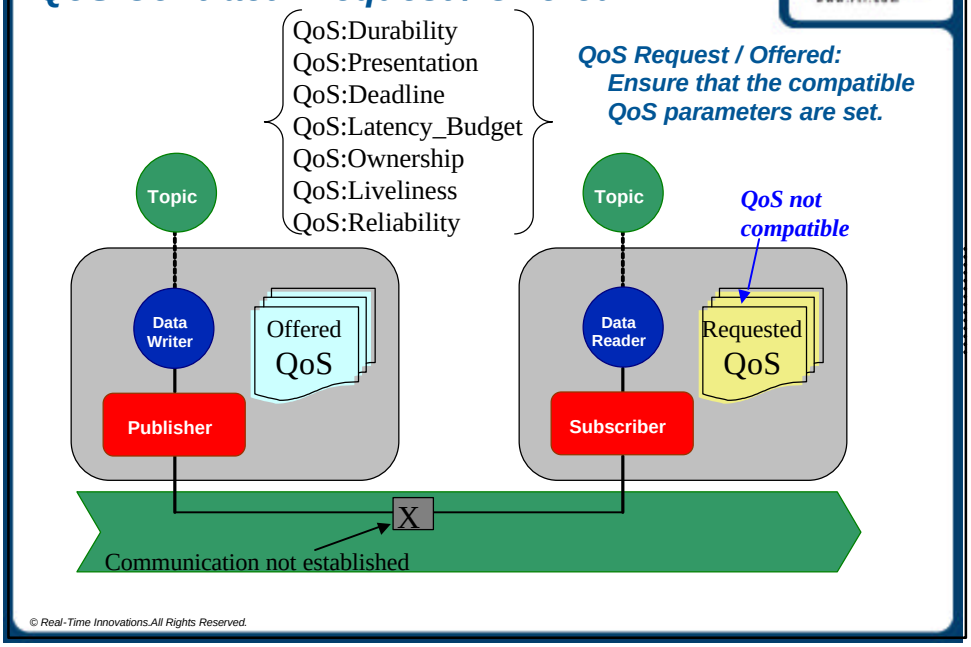
- *A StatusCondition can be selectively activated to respond to any subset of the statuses*
- *An application can wait changes in sets of Status Conditions using a WaitSet*
- *Each time the value of a STATUS changes DDS*
  - Calls the corresponding Listener operation
  - Wakes up any threads waiting on a related status change



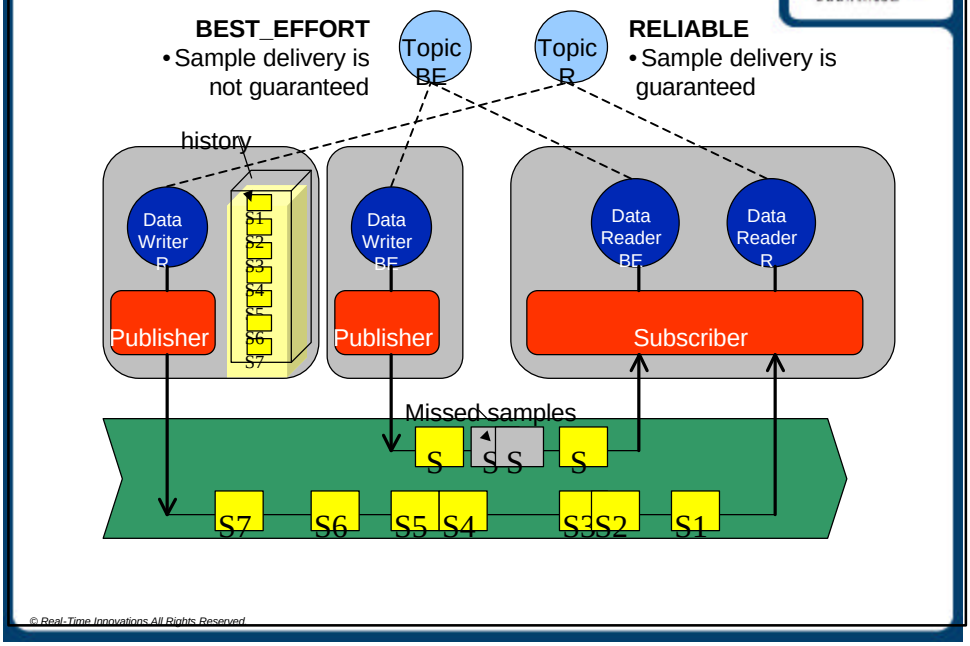
Part I DDS Overview

Background
Communication model
DDS Entities
Listeners, Conditions, WaitSets
Quality of Service

QoS Contract "Request / Offered"



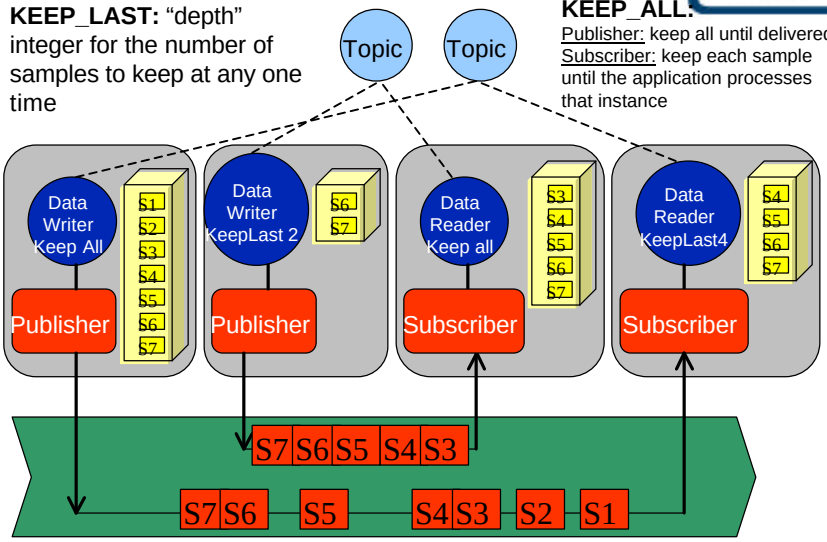
QoS: Reliability



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

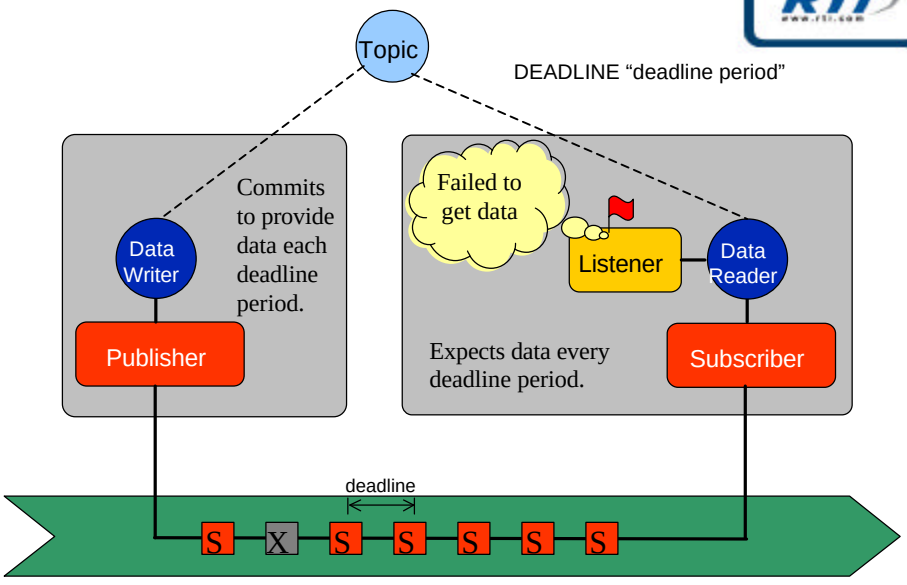
- Set of data-objects whose value codifies the state of the system

Relationship with DDS

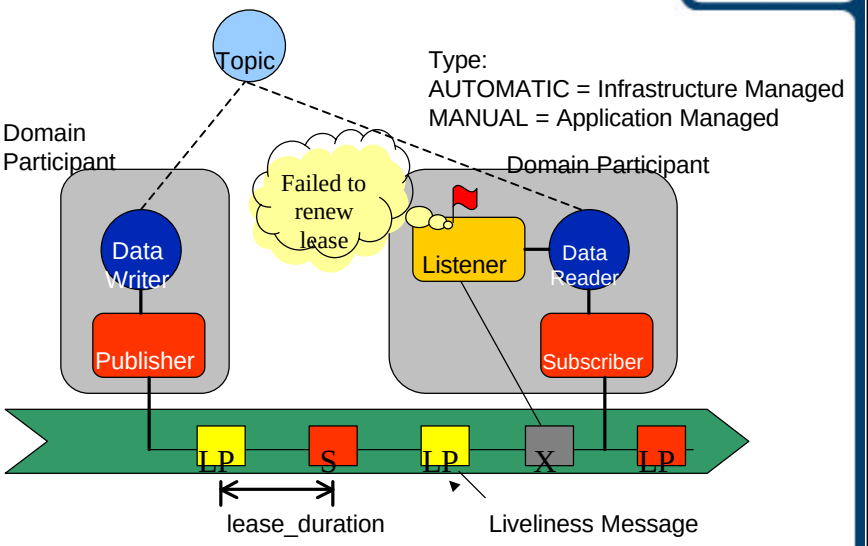
- DDS well suited to propagate and replicate state
- Topic+key can be used to represent state variables
- KEEP_LAST history QoS exactly matches semantics of state-variable propagation

Significance: Key ingredient for fault-tolerance and also present in many RT applications

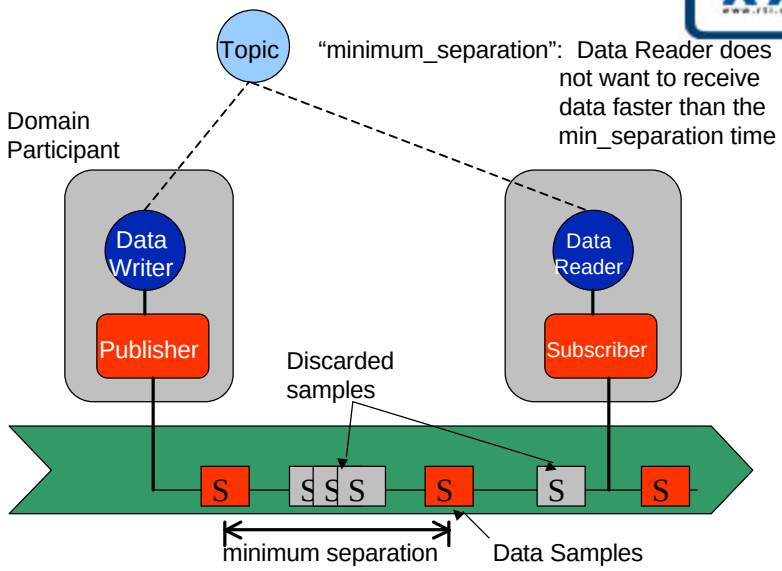
QoS: Deadline



QoS: Liveliness – Type, Duration

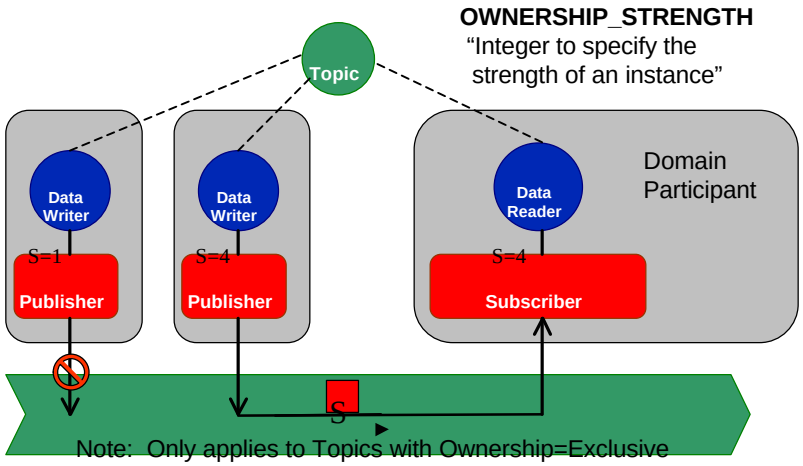


QoS: Time_Based_Filter



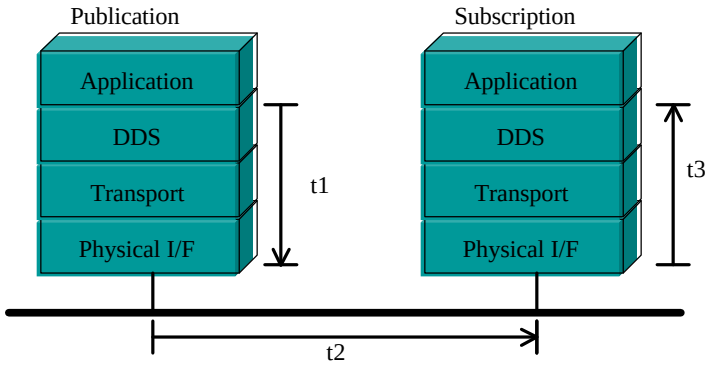
QoS: Ownership_Strength

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



QoS: Latency_Budget

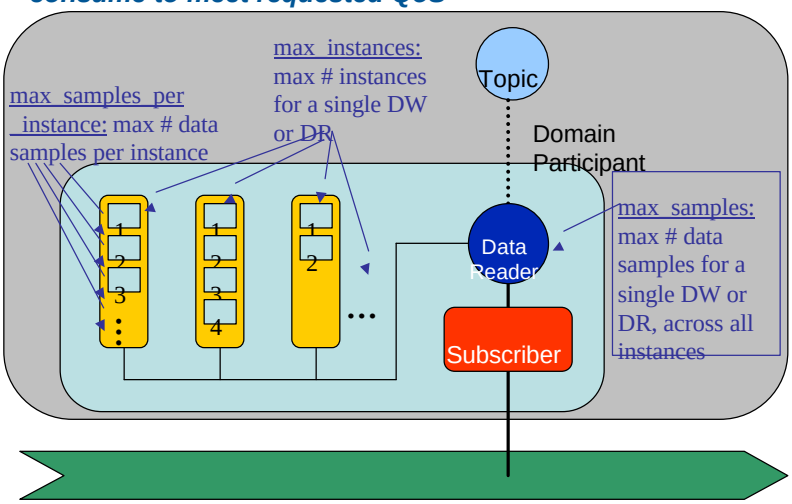
- Intended to provide time-critical information to the publisher for framework tuning where possible.
- Will not prevent data transmission and receipt.



Latency = t1 + t2 + t3

QoS: Resource_Limits

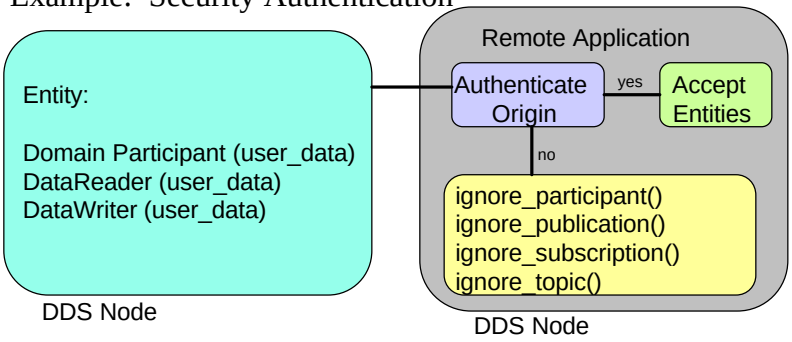
Specifies the resources that the Service can consume to meet requested QoS



QoS: USER_DATA

Definition: User-defined portion of Topic metadata

Example: Security Authentication

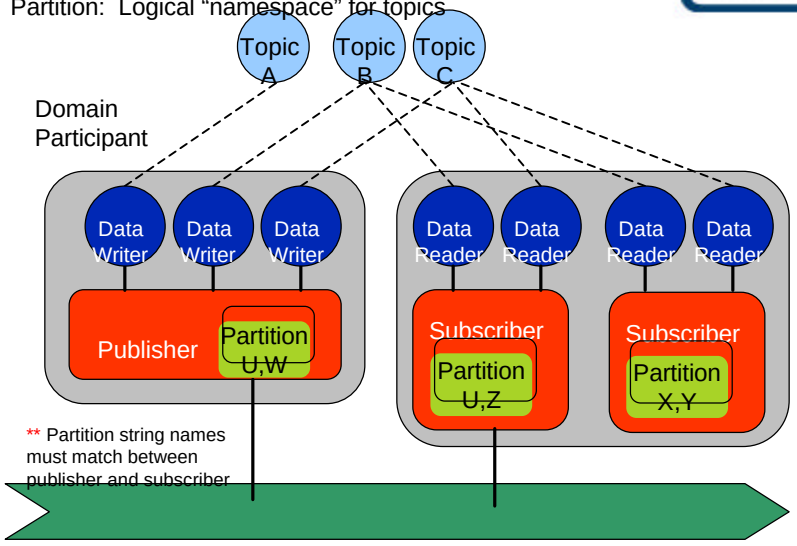


User data can be used to authenticate an origination entity.

Note: USER_DATA is contained within the DDS metadata.

QoS: Partition

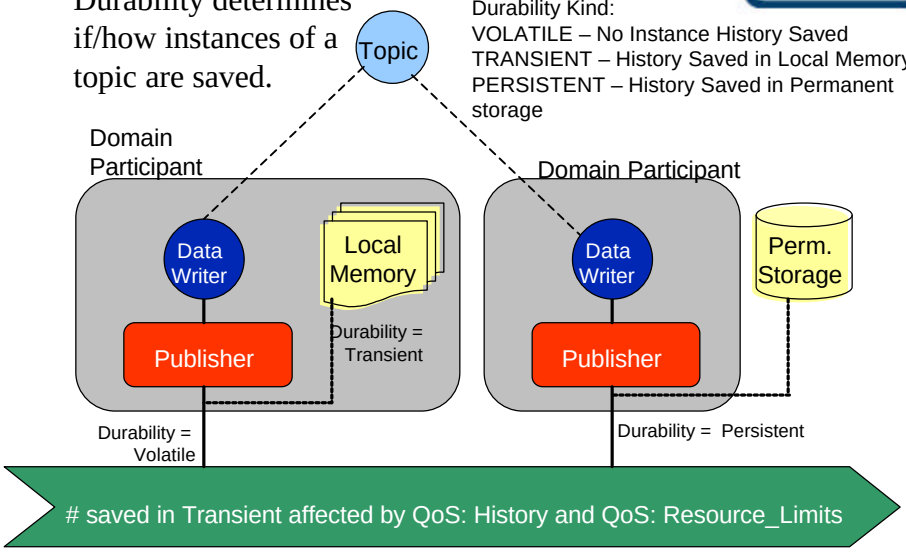
Partition: Logical "namespace" for topics



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



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.

QoS: Quality of Service (1/2)

QoS Policy	Concerns	RxO	Changeable
DEADLINE	T,DR,DW	YES	YES
LATENCY BUDGET	T,DR,DW	YES	YES
READER DATA LIFECYCLE	DR	N/A	YES
WRITER DATA LIFECYCLE	DW	N/A	YES
TRANSPORT PRIORITY	T,DW	N/A	YES
LIFESPAN	T,DW	N/A	YES
LIVELINESS	T,DR,DW	YES	NO
TIME BASED FILTER	DR	N/A	YES
RELIABILITY	T,DR,DW	YES	NO
DESTINATION ORDER	T,DR	NO	NO

QoS: Quality of Service (2/2)

QoS Policy	Concerns	RxO	Changeable
USER DATA	DP,DR,DW	NO	YES
TOPIC DATA	T	NO	YES
GROUP DATA	P,S	NO	YES
ENTITY FACTORY	DP, P, S	NO	YES
PRESENTATION	P,S	YES	NO
OWNERSHIP	T	YES	NO
OWNERSHIP STRENGTH	DW	N/A	YES
PARTITION	P,S	NO	YES
DURABILITY	T,DR,DW	YES	NO
HISTORY	T,DR,DW	NO	NO
RESOURCE LIMITS	T,DR,DW	NO	NO

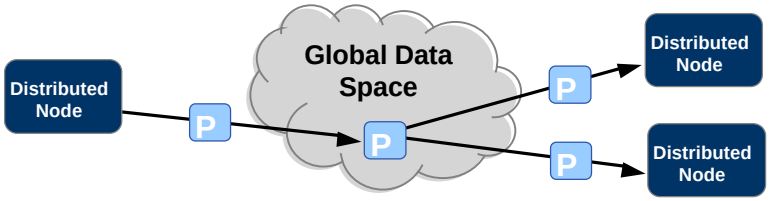
Summary

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



Part II DDS In Action

Examples and Applications
Topic management
Data visibility and access control
Application patterns
Logging and monitoring

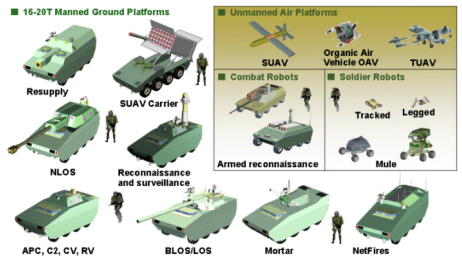
Navy OA projects adopting DDS

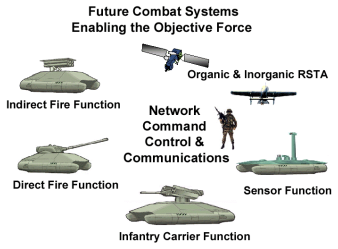
- *DD(X)*
- *Ship Self Defense System (SSDS)*
- *LCS*
- *Spy OA*
- *Sea Slice*
- *LPD 17*

JSF

Future Combat Systems (FCS)

Has specified DDS as part of SoSCOE
Large complex real-time data-distribution
Wireless with intermittent connectivity
Highly heterogeneous
Highly dynamic





Trains

RETF

- Goal: revamp and standardize railroad communications
  - ***Wireless, wired, real-time, enterprise***
- 5 Railroads
- 2 locomotive manufacturers
- A couple of associations
- ARINC hired to coordinate project

Big project

- All trains in US
- 10k trains
- 100k control stations

Early stages

- Releasing RFP

Omron (pre DDS-standard)

Tokyo traffic control

- "Metropolitan highway line"

Architecture

- Control center
- Hundreds of terminals

Goal

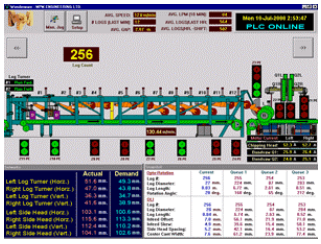
- Show drivers traffic conditions
- Report problems
- Control flow

Industrial Automation (pre DDS-standard)

Schneider Automation

(N)DDS is the primary communication channel for:

- PLC's, MotorDrive, HMI, etc.
- Reduces factory floor wiring
- Provides real time communication for synchronizing multiple devices



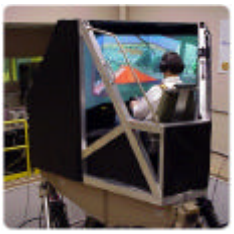
Simulation Systems (pre DDS-standard)

National Advanced Driving Simulator

- Virtual Proving Ground
- DDS ties together the geographically-distributed simulators to create one large simulation

CAE SimXXI Flight Simulators

- DDS for real-time communications between simulator subsystems
- IEEE-1394 (FireWire) physical layer for speed and determinism



Part II DDS In Action

- Examples and Applications
- Topic management
- Data visibility and access control
- Application patterns
- Logging and monitoring
- Legacy applications

Topic Management

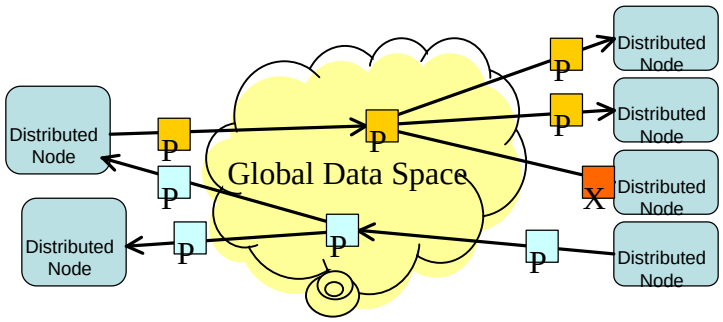
Good topic selection leads to

- Better interfaces
- Easier integration
- Improved scalability
- Decreased system size
- Faster startup and discovery times

How to go about choosing topics?

- By Sender “Role”
- By Receiver “Role”
- By Message ID
- By Data Role
- By Data Type

Topic Management



By Sender “Role” ***- MixerTank3Data***
By Receiver “Role” ***- AirTrackCorrelator***
~~***By Message ID***~~ ~~***- Filter23ToGUI12***~~
~~***By Data Role***~~ ~~***- AAWTracks***~~
(By Data Type ***- CommandString)***

Topic Management

Has the largest impact on system 'openness'…

DDS Elements

- Topic Keys
  - ***Can dramatically decrease system size***
  - ***Used for reliable many-to-one (i.e. ALARM topic)***
  - ***Enables seamless scaling of system***
- Topic QoS
  - ***Convenient way to describe information model***
- Data Types specified in IDL and can be reused
- ContentFilteredTopic and MultiTopic also control scope

Part II DDS In Action

Examples and Applications
Topic management
Data visibility and access control
Application patterns
Logging and monitoring
Legacy applications

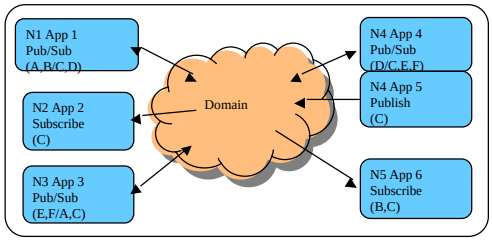
Data Visibility - Domains

Single Domain System

- Container for applications that want to communicate
- Applications can join or leave a domain in any order
- New Applications are “Auto-Discovered”
- An application that has joined a Domain is also called a “Domain Participant”

Multiple Domain System

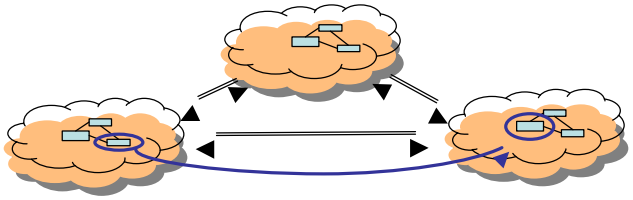
- Use Multiple domains for Scalability, Isolation or Testing



Data Visibility

Bridging – common requirements

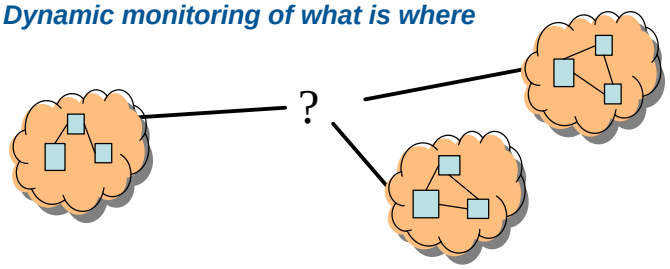
- Desired not to have all applications fully interconnected – (controlling system size)
- Clusters of machines can pub/sub any Topic in another cluster
  - ***Number of clusters can be static or dynamic***
- Often required to respond to dynamic events
  - ***Creation of new topics or appearance of new nodes***



Data Visibility – Bridging

Connecting groups of computers together

- Domains
  - *Static, tied to physical properties of the transport*
- Partitions
  - *Dynamic, can take time to propagate*
- Built-in Topics
  - *Dynamic monitoring of what is where*



Data Visibility – Partition QoS

An additional criteria that must be satisfied

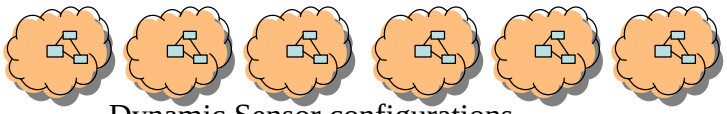
- DataReaders / DataWriters must
  - *Have Same Topic*
  - *Be in the same Domain*
  - *Have Matching PARTITION QoS*

Can be changed at run-time

- Can have propagation delay

Doesn't isolate Discovery traffic

Data Visibility – Bridging Example

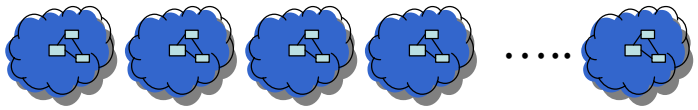


Sensors

Dynamic Sensor configurations…

Compute & Control

May need to switch sensor-sets dynamically



Displays

Displays should be running the same software
Each Display is configured differently
Additional need to manage all displays centrally

Data Access Control

A.K.A: *Security*

- Access Control and Authentication
- In many systems it is desired that some data not be distributed to unintended recipients.
- In many systems it is desired that some data not be distributed to unverified recipients.

Basic Pub/Sub paradigm is such that if a topic is in the Domain, it is available.

- There isn't the concept of a 'connection' that can be easily secured.

Could be needed at many levels

- Participant, Publisher/Subscriber, Topic

Many needs for dynamic security policies

Participants may be allowed some data, but not all.

Data Access Control

DDS doesn't provide security (e.g. encryption) but rather the hooks necessary for applications to use.

Using Built-in topics, you have the ability to selectively ignore objects in the domain

- i.e "DCPSPublication", "DCPSSubscription"
- ignore_xxx() API (can't un-ignore)

Use QoS passed during discovery

- USER_DATA, GROUP_DATA, TOPIC_DATA
- Hierarchical in nature
  - ***Could be allowed as a Participant, but not allowed to subscribe to a certain topic.***
- Can be used to pass PKI certificates

Custom serialization / de-serialization routines

Can implement custom transport

Part II DDS In Action

Examples and Applications
Topic management
Data visibility and access control
Application patterns
Logging and monitoring
Legacy applications

Application Patterns

Applications can be loosely categorized to operate within the following states:

Steady state behavior

- Moving data around subject to desired QoSs

Initialization phase

- Discovery
- Authenticating

Exception handling

- What to do when things go wrong…

Application Patterns

Many applications usually can be categorized into*

- "Commander" application
- "Sensor" application
- "HMI" application

Application 'types' benefit from different QoS policies

- Reliable / Best-effort transmission
- Use of redundancy (OWNERSHIP)
- Transport: Multicast / Unicast (Discovery & Data Delivery)
- Discovery properties (LIVELINESS)
- State propagation (HISTORY, DURABILITY)

As well as DDS implementation specifics (threading)

* A known over-simplification

Application Patterns

"Commander" applications

- Commands often need to be reliable since order matters
- Usually have some concept of state (DURABILITY, HISTORY)
  - ***Late-joiners need the last issued configuration command or status message (TRANSIENT_LOCAL)***
- Care about the health of their DataReaders
  - ***Need to know if someone has stop 'receiving' data (Listeners / WaitSet)***
- For systems wide configuration commands, multicast is very convenient
- Commands have a built-in timeout (LIFESPAN)
- Order of commands may be important:
  - ***May need to order a set of topics when written (PRESENTATION)***
  - ***May need consistent order for commands originating in different computers: BY_SOURCE_TIMESTAMP (DESTINATION_ORDER)***
- Requirements of redundancy and automatic failover (OWNERSHIP)

Application Patterns

"Sensor" Application

- Usually just reporting data
- Access to data can be limited (USER_DATA)
- The most recent value is the only one of interest
  - ***Best-effort transmission (RELIABILITY)***
  - ***Keep only the the last value (HISTORY)***
- Many Sensors reporting of the same kind
  - ***Use Keys for managing instances within a Topic***
- Primary / secondary Sensors for redundancy (EXCLUSIVE OWNERSHIP)
- Alarm sensors may benefit from ordered access:
  - ***INSTANCE/TOPIC/GROUP access_scope (PRESENTATION)***
- Alarms may originate from any source (SHARED OWNERSHIP)
- Detection sensor failures
  - ***Continuous updates: DEADLINE***
  - ***Sporadic updates (e.g. digital I/O): LIVELINESS***

Application Patterns

"HMI" Application

- Process data in a discernable order
  - *take_instance(), take_next()*
  - *Use ViewState (NEW/NOT_NEW), InstanceState (ALIVE/NOT_ALIVE), and SampleState (READ/NOT_READ)*
- For repetitive sensor data, often the most current is all that is needed
  - ***Use RELIABILITY= BEST_EFFORT, HISTORY= KEEP_LAST)***
- For sporadic command data reliability is important
  - ***Use RELIABILITY= RELIABLE, HISTORY= KEEP_ALL)***
- Only certain subsets of data are of interest
  - ***Use ContentFilteredTopics for selecting based on content***
  - ***Use TIME_BASED_FILTER for selecting based on timing***
- Need to know when you lose connectivity
  - ***Take different actions depending on down-time (LIVELINESS)***
- Want to 'batch' processing of received data/threading concerns
  - ***Read/take asynchronously***
- Want to know if specific data missed
  - ***Use DEADLINE***

Part II DDS In Action

Examples and Applications
Topic management
Data visibility and access control
Application patterns
Logging and monitoring
Legacy applications

Logging / Monitoring

Everyone needs it to some degree

- Log Events
- Log Data

Issues

- Sometimes want globally ordered
- Easy to get locally ordered
- Data content extraction, using defined IDL types

Log DDS meta-traffic

Infrastructure can provide additional tools

Logging / Monitoring - Listeners

~~on_inconsistent_topic()~~	~~Topic~~
~~on_liveliness_lost()~~	~~DataWriter~~
~~on_offered_deadline_missed()~~	~~DataWriter~~
~~on_offered_incompatible_qos()~~	~~DataWriter~~
~~on_publication_match()~~	~~DataWriter~~
~~on_data_on_readers()~~	~~Subscriber~~
~~on_sample_lost()~~	~~DataReader~~
~~on_data_available()~~	~~DataReader~~
~~on_sample_rejected()~~	~~DataReader~~
~~on_liveliness_changed()~~	~~DataReader~~
~~on_request_deadline_missed()~~	~~DataReader~~
~~on_request_incompatible_qos()~~	~~DataReader~~
~~on_subscription_match()~~	~~DataReader~~

DDS Listeners – can be monitored, logged, or ignored

Part II DDS In Action

Examples and Applications
Topic management
Data visibility and access control
Application patterns
Logging and monitoring
→ Legacy applications

Legacy Applications and DDS

Emerging needs in existing programs

- Open interfaces
- Interoperability with others
- Maintainability
- Performance issues
- New Transports, hardware, and software

Transition to DDS implementation usually occurs in several steps

- First pass of integration: pass around current messages
- "Middle" ground: Some QoS utilized, porting some topics

Legacy Applications and DDS

Map current communication infrastructure to Topics

- Usually take a MessageID => Topic approach first
- Can further sub-class messages by using both sender and receiver roles – thus reducing number of Topics

Reuse existing serialization/deserialization code, NDDS messages are all "opaque"

QoS settings tend to be very simple

- All messages "reliable" for example

Determine most compatible threading model

- Listener verses WaitSet for the DataReaders

Address issues of scaling and system scope

- Domains, Partitions, and other discovery properties

Legacy Applications and DDS

Further integration efforts involve

- Re-distribution of Topics and DataTypes
- Changing the threading models
- Re-examining QoS properties and their settings
- Tuning Discovery and startup properties
- Adding security and authentication

Conclusion

DDS enables the net-centric vision

The right information

- DDS is based on publish-subscribe technology
- Each application gets exactly what it wants!

To the right place

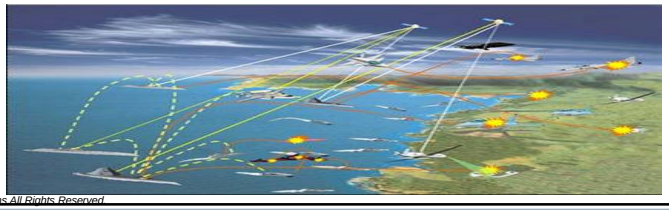
- DDS includes automatic discovery
- System automatically locates publishers and subscribers

At the right time…

- QoS contracts allow applications to specify what they need
- DDS automatically monitors QoS and notifies of violations

… no matter what

- Built-in redundancy, lack of centralized logic, strength-arbitration logic, facilitate development of a fault-tolerant system




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