DDS TUTORIAL
For OMG RTE workshop July 2007
V3

Hans van 't Hag, DDS product Manager
hans.vanthag@prismtech.com
Agenda

Data-centric Foundation
✓ Into the mood
✓ Net-centric Future: ‘the data is the network’
✓ The ‘information-centric approach’

Driving the Standard
✓ Naval Combat Systems (CMS) example
✓ The OMG DDS specification

DDS ‘By Example’
✓ DDS profiles
✓ Corba Integration

Concept Demo
✓ DDS Chatroom ‘concept’ demo
Into the mood (1): ‘huh ?...’
Into the mood (2): ‘The Problem…’
Into the mood (3): ‘A scalable solution?...’
Into the mood (4): ‘...Client/server vs. Pub-sub: A mind-shift...’
Net-centric Future:
‘The Data is the Network’
© ARMS (Doug Schmidt): R&D Challenge

Critical Infrastructures

- Government Operations
- Emergency Services
- Telecommunications
- Electrical Energy
- Gas & Oil Storage and Delivery
- Water Supply Systems
- Banking & Finance
- Transportation
Develop DRE Systems Technologies to Protect Critical Infrastructure

The “soft underbelly” of commercial, military, & infrastructure DRE systems depend increasingly on **information technology**, making attacks both attractive & lethally effective.

**R&D Challenges:** Create
- efficient,
- scalable,
- reliable,
- secure,
- & predictable

DRE system technologies from nano- to tera-scale
Emerging Trends
• Large-scale DRE system requirements are increasingly more
  • Dynamic
  • Diverse
  • Demanding

1000 – 1,000,000 node fusion of physical & information systems

Total Ship Computing Environments: Global dynamic resource management (1,000 nodes)

Distributed Active Control: Vibration Damping on Delta-4 Rocket Payload Fairing (1,000 nodes)

Gossamer Space Antenna (1,000,000 nodes)

Distributed Network of embedded sensor motes for environment monitoring, tracking, & surveillance (10,000 nodes)

Noiseless sonar on submarines to provide camouflage (3,000 nodes)
Leverage the Power of Information

NET-CENTRICITY:

People, processes, and technology working together to enable timely and trusted:

- ACCESS to information
- SHARING of information
- COLLABORATION among those who need it

Can Only Be Done on The Net!

Connecting People With Information
The Move to Net-Centricity

Current

- Information stovepipes
- "Welded" interfaces
- Predetermined needs
- Fixed display formats
- Need to know
  
Net-Centric

- Shared information
- Unconstrained
- Unanticipated users
- User-defined info and formats
- Need to share; right to know

Rigid

Agile
Net-Centric Framework

- **Data Strategy:**
  - How to "share" the data

- **Enterprise Services:**
  - How to "access" the data

- **Information Transport:**
  - How to "move" the data

- **Network Operations:**
  - How to "operate and defend" the GIG

- **Information Assurance:**
  - How to keep it all "dependable"

**Data:** Discoverable, Accessible, Understandable

Connecting People With Information
### Grids

<table>
<thead>
<tr>
<th>Catch Phrase</th>
<th>System</th>
<th>Collections</th>
<th>Components</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Network Is the computer</td>
<td>Objects</td>
<td>Legacy to the Web</td>
<td>The Computer Is the Network</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Scale</th>
<th>When/Peak</th>
<th>Leaf Protocol(s)</th>
<th>Directory(s)</th>
<th>Session</th>
<th>Schematic</th>
</tr>
</thead>
<tbody>
<tr>
<td>100s</td>
<td>1984/1987</td>
<td>X</td>
<td>NIS, NIS+</td>
<td>RPC, XDR</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1000s</td>
<td></td>
<td>+ CDS</td>
<td>+ CORBA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1000000s</td>
<td></td>
<td>+ LDAP (*)</td>
<td>+ CORBA, RMI</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Net Apps</th>
<th>Net Services</th>
<th>Next</th>
<th>After that</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ HTTP (+JVM)</td>
<td>+ UDDI</td>
<td>+ jini</td>
<td>+ ?</td>
</tr>
<tr>
<td>+ LDAP (*)</td>
<td>+ SOAP, XML</td>
<td>+ RMI/Jini</td>
<td>+ ?</td>
</tr>
</tbody>
</table>

---

**“Information - Backbone” (DDS)**

- **Corba**
- **Web**
- **Java**
- **Information Grids**

---

*Copyright © PrismTech 2007 Proprietary information subject to non-disclosure agreement.*
Dr. Richard Soley (OMG Chairman & CEO):

“The DCPS publish/subscribe model stands as a natural complement to the object-centric client/server model provided by CORBA”

(Consumer Electronics, September 13, 2004)

Doc Allen (OMG Co-Chair & Mitre):

“DDS clearly has the potential to become ‘THE’ dominant (real-time) standard in the Net-centric environment”

(OMG-RTESS plenary meeting June’04)

Dave Sharp (Boeing FCS chief-architect):

“The (Army) SOSCOE environment will be based on MDA/UML and OMG-DDS”

(OMG sponsor-presentation, June’04)
The Idea: Reduced complexity

- pub/sub already patented in 1987
- information backbone
- “Right info, Right place, Right time

Towards an ‘information-centric’ world

- Loosely coupled components
- Dynamic systems
- Traditional architectures don’t suffice
ARCHITECTURE TRENDS: DDS, A Major Specification
_\textit{in Line with the Network Centric Warfare Paradigm}_

**DDS mandated for US - OACE**
- DDS is key for success in NAVY OA
- Evaluated since 2004

="NDDS/SPLICE provide good performance and scalability as a publish-subscribe middleware for combat system applications"

(NSWC-DD, OA Technical assessment 07/2004)

**DARPA “ARMS” study**

- **Key Trend**
  - DoD system requirements are increasingly more dynamic, diverse, & demanding

- **Problems**
  - Existing architectures
  - Existing COTS
  - Existing multiple technology bases
  - brittle & configured statically
  - too big, slow, buggy, incapable, & inflexible
  - proprietary & limited effectiveness by impeding
  - Assurability of QoS,
  - Adaptability
  - Affordability

- **Today, each system brings its own: networks computers displays software people**

- **Consequences**
  - Hard to meet required performance levels
  - Hard to control distributed resources
  - High software lifecycle costs
  - e.g., many “accidental complexities” & low-level platform dependencies

"The finalization and availability of the DDS specification really is a tremendous achievement that addresses a significant need in both government and civilian sectors"

(Dr. Richard Soley OMG Chairman, Consumer Electronics, September 13, 2004)

**Recognized potential**
- DARPA recognizes DDS importance
- Dynamic Resource Management potential
Of the major distributed technologies, CORBA and DDS offer the widest range of competitive vendor alternatives for a multi language, multi platform, real-time processing environment.

Support for Real-time QoS is essential.
Data-Distribution and Real-Time

- DDS APPLICABILITY (from US OACE documentation)
Traditional Architectures: fault propagation
Traditional Architectures: fault propagation

Traditional Architectures: fault propagation

Diagram showing the relationships between different components labeled L, G, H, I, J, K, Z, Y, M, N, P, Q, T, W, and X.
Traditional Architectures: fault propagation
Autonomous components
Interacting only with the information-bus
Spontaneous:  Z  , Self-healing:  D’
Redundant & Replicated:  L’  ,  Y’
QOS-driven Data Distribution Service (reliability, urgency, importance):  DDS
Pub/Sub: ‘Keep it simple stupid’
Many different types of requirements
Design principles:

- Minimize dependencies between components
- Share stable properties

Focus on:

- Autonomous component behavior
- Common information model

Middleware delivers:

- “The right information at the right place at the right time”
- **System design**
  - Provide a **stable basis** to operate upon by applications
  - Enhance component **autonomy**
  - Allow transparent and global **QoS** assurance

- **System development**
  - Reduce **complexity** and enhance **re-usability**
  - Provide shared/guaranteed properties
  - **Small** learning effort and flat learning curve

- **System integration**
  - Support effortless component **integration**
  - Provide easy **monitor & control**
  - **Shift ratio** between design and integration effort

- **System deployment**
  - Guarantee **QoS** for reliability, latency and persistency
  - Allow runtime **migration** of applications
  - Allow applications to **join** the system at any time

- **System maintenance & evolution**
  - Allow runtime replacement and **evolutionary upgrading**
  - Support for **logging & replay** of information
  - Provide **future-proof, re-usable, robust and scalable** system
"DDS", a distributed real-time software architecture

Built-in capabilities:
- P/S data distribution
- relational data access
- data persistence
- dynamic (re-) configuration
- quality of service
- fault-tolerance
- information partitioning

Decoupling in space and time
“DDS”, a distributed real-time software architecture

Decoupling in space and time

Built-in capabilities:
- P/S data distribution
- relational data access
- data persistence
- dynamic (re-) configuration
- quality of service
- fault-tolerance
- information partitioning
“DDS”, a distributed real-time software architecture

Decoupling in space and time

Built-in capabilities:
- P/S data distribution
- relational data access
- data persistence
- dynamic (re-) configuration
- quality of service
- fault-tolerance
- information partitioning
“DDS”, a distributed real-time software architecture

Data is dynamically forwarded to all subscribed components

Built-in capabilities:
- **P/S data distribution**
- relational data access
- data persistence
- dynamic (re-) configuration
- quality of service
- fault-tolerance
- information partitioning
Applications subscribe to **TOPICS**

Each **TOPIC** has an associated **name** and data **type**:
- Data (type) definition in IDL
- ‘**key**’ fields for unique identification
- more recent instances overwrite existing instances with same key value 
  (keeping into account the ‘history-depth’ QoS setting of a subscriber)

**Example (IDL types):**

```idl
Struct PointTrack {
    long source; // key
    long trackId; // key
    Position pos;
}
```

```idl
Struct TrackState {
    long source; // key
    long trackId; // key
    long environment;
    long identity;
}
```
“DDS”, a distributed real-time software architecture

- Data is dynamically forwarded to all subscribed processes

Built-in capabilities:
- P/S data distribution
- Relational data access
- Data persistence
- Dynamic (re-) configuration
- Quality of service
- Fault-tolerance
- Information partitioning
Realization

application → cache → real-time information broker → cache → application

application → cache → real-time information broker → cache → application

application → cache → real-time information broker → cache → application

Networking (UDP/IP)
“DDS”, a distributed real-time software architecture

Built-in capabilities:
- P/S data distribution
- relational data access
- data persistence
- dynamic (re-) configuration
- quality of service
- fault-tolerance
- information partitioning
"DDS", a distributed real-time software architecture

Built-in capabilities:
- P/S data distribution
- relational data access
- data persistence
- dynamic (re-) configuration
- quality of service
- fault-tolerance
- information partitioning

Content-based subscription through relational views
Relational Object Models

Track
- trackId
- source

PointTrack
- position

TrackState
- environment
- identity
Data filtering in DDS, e.g. using SQL:

```sql
select * from TrackPosition
where position.range < 10000
```

Aggregation and projection in DDS, e.g. using SQL:

```sql
select position, environment
from PointTrack NATURAL JOIN TrackState
where position.range < 10000
and identity = hostile
```
Typical Process Behavior

Register with DDS

Declare publications

Declare subscriptions

Define queries (read_conditions)

Set query parameters

Select input data

Wait for input data

Process data

Publish output data
User Application:

- Creates related DDS entities
  - Publisher
  - Topic
  - DataWriter
- Configures entities' QoS then
- Provides data to DataWriter
**User Application:**
- Creates related DDS entities
  - Subscriber
  - Topic
  - DataReader
- Configures entities' QoS and attach listeners
- Receives Data from DataReader through attached listeners
**User Application:**
- Creates related DDS entities
  - Subscriber
  - Topic
  - DataReader
- Configures entities' QoS
- Creates a Condition and attaches it to a WaitSet
- Waits on the WaitSet until data arrive, then picks it on the DataReader
“DDS”, a distributed real-time software architecture

Built-in capabilities:
- P/S data distribution
- relational data access
- data persistence
- dynamic (re-) configuration
- quality of service
- fault-tolerance
- information partitioning
“DDS”, a distributed real-time software architecture

Built-in capabilities:
- P/S data distribution
- relational data access
- data persistence
- dynamic (re-) configuration
- quality of service
- fault-tolerance
- information partitioning

Persistent data remains available for later access
“DDS”, a distributed real-time software architecture

Persistent data remains available for later access

Built-in capabilities:
- P/S data distribution
- relational data access
- **data persistence**
- dynamic (re-) configuration
- quality of service
- fault-tolerance
- information partitioning
“DDS”, a distributed real-time software architecture

Built-in capabilities:
- P/S data distribution
- relational data access
- **data persistence**
- dynamic (re-) configuration
- quality of service
- fault-tolerance
- information partitioning

Persistent data remains available for later access
Data Persistency

- **Volatile** data:
  - no copies outside process space
  - typically *measurement* related data

- **Transient** data:
  - copies are kept on more than one host
  - outlives process
  - typically *state* related data

- **Persistent** data:
  - same as transient data, but additionally stored on disk
  - outlives system
  - typically *configuration* data
Self-Healing: Fault-tolerant persistence
“DDS”, a distributed real-time software architecture

Built-in capabilities:
- P/S data distribution
- relational data access
- data persistence
- **dynamic (re-)configuration**
- quality of service
- fault-tolerance
- information partitioning

Self forming architecture
“DDS”, a distributed real-time software architecture

Built-in capabilities:
- P/S data distribution
- relational data access
- data persistence
- **dynamic (re-) configuration**
- quality of service
- fault-tolerance
- information partitioning

Self forming architecture
“DDS”, a distributed real-time software architecture

Built-in capabilities:
- P/S data distribution
- relational data access
- data persistence
- dynamic (re-) configuration
- quality of service
- fault-tolerance
- information partitioning

QoS is attributed to data (statically or dynamically)
Data Delivery

- **Reliability:**
  - “Best effort”
    - No Guaranteed delivery i.e. no retransmissions when data ‘gets lost’
    - Typical for **periodic measurement** related data
  - “Reliable”
    - Guaranteed delivery by automatic re-transmissions of lost data
    - Typical for **non-periodic and important** published information

- **Latency:**
  - “Latency Budget”
    - Specifies ‘how fast’ data should be delivered i.e. its ‘urgency’
    - Allows middleware to balance between high-volume & low-latency

- **Priority:**
  - “Transport_Priority”
    - Specifies the priority of the published information i.e. its ‘importance’
    - Allows middleware to pre-empt low-priority data with high-priority data
Overview: DDS QoS policies

**QosPolicy**
- name : string

**DurabilityQosPolicy**
- kind : DurabilityQosKind

**OwnershipQosPolicy**
- kind : OwnershipQosKind

**OwnershipStrengthQosPolicy**
- value : long

**LivelinessQosPolicy**
- lease_duration : Duration_t
- kind : LivelinessQosKind

**PresentationQosPolicy**
- access_scope : PresentationQosAccessScopeKind
- coherent_access : boolean
- ordered_access : boolean

**LatencyBudgetQosPolicy**
- duration : Duration_t

**DeadlineQosPolicy**
- period : Duration_t

**TimeBasedFilterQosPolicy**
- minimum_separation : Duration_t

**UserDataQosPolicy**
- data [*] : char

**PartitionQosPolicy**
- name [*] : string

**ReliabilityQosPolicy**
- kind : ReliabilityQosKind

**DestinationOrderQosPolicy**
- kind : DestinationOrderQosKind

**LatencyBudgetQosPolicy**
- duration : Duration_t

**DurabilityQosPolicy**
- max_samples : long
- max_instances : long
- max_samples_per_instance : long

**HistoryQosPolicy**
- kind : HistoryQosKind
- depth : long
QoS Policies – Conceptual View

- PresentationQoS
- QueryCondition
- ReadCondition
- OwnerShipQoS
- DestinationOrderQoS
- DurabilityQoS
- HistoryQoS
- ResourceLimitsQoS
- TimeBasedFilterQoS
- PartitionQoS
- ContentBasedFilter

DataReader Caches

Resolve updates

Storage Area

Receive

Computer
QoS Contract “Request / Offered”

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

Communication not established if offered QoS < requested QoS

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

QoS not compatible

Publisher

Subscriber

Data Writer

Data Reader

Offered QoS

Requested QoS

Topic
“DDS”, a distributed real-time software architecture

Built-in capabilities:
- P/S data distribution
- relational data access
- data persistence
- dynamic (re-) configuration
- quality of service
- fault-tolerance
- information partitioning

Self-healing software architecture
“DDS”, a distributed real-time software architecture

Biult-in capabilities:
- P/S data distribution
- relational data access
- data persistence
- dynamic (re-) configuration
- quality of service
- fault-tolerance
- information partitioning

Passive process replication
“DDS”, a distributed real-time software architecture

Built-in capabilities:
- P/S data distribution
- relational data access
- data persistence
- dynamic (re-) configuration
- quality of service
- fault-tolerance
- information partitioning

Passive process replication
“DDS”, a distributed real-time software architecture

Built-in capabilities:
- P/S data distribution
- relational data access
- data persistence
- dynamic (re-) configuration
- quality of service
- fault-tolerance
- information partitioning

Semi-active process replication
“DDS”, a distributed real-time software architecture

Built-in capabilities:
- P/S data distribution
- relational data access
- data persistence
- dynamic (re-) configuration
- quality of service
- **fault-tolerance**
- information partitioning

Semi-active process replication
Built-in capabilities:

- P/S data distribution
- relational data access
- data persistence
- dynamic (re-) configuration
- quality of service
- fault-tolerance
- information partitioning

Semi-active process replication
“DDS”, a distributed real-time software architecture

Built-in capabilities:
- P/S data distribution
- relational data access
- data persistence
- dynamic (re-) configuration
- quality of service
- fault-tolerance
- information partitioning
“DDS”, a distributed real-time software architecture

Built-in capabilities:
- P/S data distribution
- relational data access
- data persistence
- dynamic (re-) configuration
- quality of service
- fault-tolerance
- information partitioning

Copyright © PrismTech 2007 Proprietary information subject to non-disclosure
“DDS”, a distributed real-time software architecture

Built-in capabilities:
- P/S data distribution
- relational data access
- data persistence
- dynamic (re-) configuration
- quality of service
- fault-tolerance
- **information partitioning**
Summary: The Problem

> Problem: *engineering (-cost) of distributed systems*
  > too complex
  > not reactive
  > not future-proof
  > not fault tolerant

> Because ‘*multi-dimensional engineering’ is needed:*

> What about *the current ‘state-of-the-art’?*
  > architectures: client/server, message-passing, DBMS
  > most efforts fall short in a number of dimensions:
  > typically:
    > *limited RT performance* (high-volume & low-latency balance)
    > *exploding complexity* (dependencies in many dimensions)
    > *costly evolution* (impact of changes & extensions)
Summary: An Information-Centric Approach

Towards a solution:
- make development effort more simple
- develop less
- develop solutions only once

How:
- minimize component dependencies (‘simple’)
- maximize component autonomy (‘re-use’)
- normalize component interactions (‘only once’)

The clue:
- *share* the stable properties, *localize* the unstable ones
- *information* is what matters most, not how it is processed
- properly modeled data is *stable*, processing often is not
- so *focus on data* first and then on the processing of it
Summary: the DDS concept

Data-centric architecture:

- **autonomous** components with **minimal dependencies**
- **separation** of function and interaction
- architecture that **focuses** on **data**

DDS realization:

- middleware that offers a **normalized environment**
- designed once, **guarantees** system properties
- delivers ‘**the right data at the right place at the right time**’

What about the processing?

- **standard** operating platform (HW, OS)
- **standard** communication facilities
- **standard** programming languages
- **standard** development tools
### Requirement:

<table>
<thead>
<tr>
<th>System design</th>
<th>Realized by:</th>
</tr>
</thead>
<tbody>
<tr>
<td>provide a <strong>stable basis</strong> to operate upon by applications</td>
<td><strong>shared Information Model</strong></td>
</tr>
<tr>
<td>enhance component <strong>autonomy</strong></td>
<td><strong>state-based information-centric system</strong></td>
</tr>
<tr>
<td>allow transparent and global <strong>QoS assurance</strong></td>
<td><strong>Information classification (QoS topic-defaults)</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>System development</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>reduce <strong>complexity</strong> and enhance <strong>re-usability</strong></td>
<td><strong>minimized component dependencies</strong></td>
</tr>
<tr>
<td>provide <strong>shared/guaranteed properties</strong></td>
<td><strong>standardized (DDS-) interaction-environment</strong></td>
</tr>
<tr>
<td><strong>small</strong> learning effort and flat learning curve</td>
<td><strong>intuitive concept, simple/powerful features</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>System integration</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>support effortless component <strong>integration</strong></td>
<td><strong>maximized component autonomy</strong></td>
</tr>
<tr>
<td>provide <strong>easy</strong> monitor &amp; control</td>
<td><strong>globally accessible information (data+metadata)</strong></td>
</tr>
<tr>
<td><strong>shift</strong> ratio between design and integration effort</td>
<td><strong>focus on info-model &amp; decoupled applications</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>System deployment</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>guaranty <strong>QoS</strong> for reliability, latency and persistency</td>
<td><strong>realtime “DDS” information backbone</strong></td>
</tr>
<tr>
<td>allow <strong>runtime migration</strong> of applications</td>
<td><strong>global &amp; FT availability of transient state data</strong></td>
</tr>
<tr>
<td>allow applications to join the system at <strong>any time</strong></td>
<td><strong>dynamic discovery and data persistence</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>System maintenance &amp; evolution</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>allow runtime replacement and <strong>evolutionary upgrading</strong></td>
<td><strong>de-coupled &amp; autonomous components</strong></td>
</tr>
<tr>
<td>support for <strong>logging &amp; replay</strong> of information</td>
<td><strong>global availability of all (time-stamped) data</strong></td>
</tr>
<tr>
<td>provide <strong>future-proof, re-usable, robust and scalable system</strong></td>
<td><strong>highly adaptive associative data-model</strong></td>
</tr>
</tbody>
</table>
.. 15 minutes break ..
Combat Systems Example...
... A requirements-driving domain
CHARACTERISTICS

Many different customers: >15 Navies world-wide use DDS pub/sub
Many different ships/missions: >22 Ships classes (from FPB’s up to Destroyers)
Large-scale & mission-critical: >150 CPU’s, >2200 applications, >4.000 tracks/sec
Real-time and Fault-tolerant: Battle-damage resistant, deterministic, reliable
Data-traffic: >4,000 publications per second over the system-data bus
Programs: 2,200 programs allocated over 150 processors
Data flows: urgent & non-urgent data (latency), important & less-important data (priority)
DDS: ‘Self-forming 3-tier data-planes’

Near-Realtime HCI
- Dialogue & presentation
- Network tracks
- Display data
- consoles

Realtime Command & Control
- Business logic
- System tracks
- Fully-Distributed processing
- Engagements
- sensors

Hard-realtime sensor/weapon IO
- actuators
- Data-sources
- Missile uplinks

Reference data (time / platform-data)
the OMG DDS Specification
The DDS Specification

• Data Distribution Service for Real-Time Systems
  – Adopted in June 2003, Finalized in April 2004
  – Joint submission
    www.omg.org/technology/documents/formal/data_distribution.htm
  – Specifies the API required for a Data-Centric Publish-Subscribe communication environment for real-time distributed systems

Dr. Richard Soley, OMG Chairman and CEO:
“The DCPS publish/subscribe model stands as a natural complement to the object-centric client-server model provided by CORBA.”
“The finalization and availability of the DDS specification really is a tremendous achievement that addresses a significant need in both government and civilian sectors.”
  -- in Consumer Electronics, September 13, 2004
DDS Structure

- DCPS: Data Centric Publish/Subscribe
  - Purpose: QoS-Driven Distributed Data Management
- DLRL: Data Local Reconstruction Layer (optional)
  - Purpose: An OO Model to access data as local objects
- Related Specification: DDS-RTPS
  - Purpose: provide (network-)interoperability between multiple DDS vendors
DDS Features

**Object Oriented information view**
- Local object model extending the distributed DCPS data model
- Manages relationships and supports native language constructs

**Distributed QoS-driven information management**
- Fault tolerance and global persistence of selected data
- Guaranteed data availability supports application fault-tolerance
- Content-aware filtering and dynamic queries:
  - reduces application complexity
  - improves system performance

**Real-time publish/subscribe messaging:**
- Asynchronous ‘one-to-many’ real-time data communication
- Dynamic data flow based on ‘current interest’ (pub/sub)
- Platform independent data model (IDL)
- Strongly typed interfaces for multiple languages
- Information Ownership management for replicated publishers
Contributions to the OMG-DDS specification

**DDS ‘drivers’**
- OMG/CORBA
  - IDL for data-definition
  - Object orientation
- SPLICE
  - Content awareness
  - Information Management
- NDDS/SPLICE
  - pub/sub messaging
  - real-time networking

**DDS Compliance-profiles**
- Object-Model
- Persistence
- Content-Subscription
- Ownership
- Minimum-Profile

**DLRL Module**
**Persistence Module**
**Cont. Sub. Module**
**Core Modules**
Overall DCPS Model

- **QosPolicy** relates to **DCPSEntity** with a multiplicity of 0..1, indicating that a QoS policy may be associated with one or more DCPSEntities.
- **DCPSEntity** has a multiplicity of 1 on its relationship with **Listener** and **Condition**, indicating an association with exactly one listener and condition.
- **DomainEntity** relates to **DomainParticipant** with a multiplicity of 1, suggesting that a domain entity is associated with exactly one domain participant.
- **DomainParticipant** associates with **Publisher**, **Topic**, and **Subscriber**.
- **Publisher** and **Subscriber** each have a multiplicity of 1, indicating a single association with **DomainParticipant**.
- **Topic** has a multiplicity of 1 on its association with **DataWriter** and **DataReader**, implying a single topic instance.
- **DataWriter** and **DataReader** have a multiplicity of 1, indicating a single association with **Topic**.
- **Data** is connected to **DataWriter** and **DataReader**, indicating data transmission between them.

**Summary**:
A DomainParticipant is the entry-point for the service and isolates a set of applications that share a physical network.
## Overall DCPS Model

<table>
<thead>
<tr>
<th>DCPS Entity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Publisher</td>
<td>Responsible for data distribution taking into account the applicable QoS-policies</td>
</tr>
<tr>
<td>DataWriter</td>
<td>Holds the data and enables modifications</td>
</tr>
<tr>
<td>Subscriber</td>
<td>Responsible for receiving data taking into account the applicable QoS policies</td>
</tr>
<tr>
<td>DataReader</td>
<td>Holds the data and provides access to the data</td>
</tr>
<tr>
<td>Topic</td>
<td>Associates a name with a data type</td>
</tr>
<tr>
<td>Content filtered topic</td>
<td>Expresses interest in content-filtered data</td>
</tr>
<tr>
<td>MultiTopic</td>
<td>Expresses interest in aggregated (&amp; filtered) data</td>
</tr>
</tbody>
</table>
A DomainParticipant is the entry-point for the service and isolates a set on applications that share a physical network.
DDS™ by Example: “building a mini CMS, …”

**SENSOR PROCESS**
- Optical sensor
- Scans the environment
- Produces ‘Tracks’
- Position of ‘objects’
- Reports ‘pointTrack’

**CLASSIFICATION PROCESS**
- Classifies tracks
- Determines their identity
- Analyses the trajectories
- Determines hostility
- Reports ‘trackState’

**DISPLAY PROCESS**
- Displays track info
- Both position & identity
- Raises alerts
- Requires ‘pointTrack’
- Requires ‘trackState’
Information modeled as “TOPICS”

Each **TOPIC** has an associated name and data type
- Data-definition in IDL
- ‘Key’ fields for unique identification
- Relational Data Model (keys)

Our example:

*Topic “PointTrack”*

```cpp
struct PointTrackType {
    long trackId;    // key
    Position pos;
}
```

*Topic “TrackState”*

```cpp
struct TrackStateType {
    long trackId;    // key
    Id identity;
}
```
DDS™ by Example: “the Minimum Profile …”

Sensor

PointTrack Publisher

Topic
PointTrack {
long trackId;
Position pos;
}
Key: trackId
QoS: best-effort, volatile

PointTrack Topics

Display

PointTrack Subscriber

OMG-DDS Real-time Information Backbone

Characteristics
- Basic publish/subscribe data distribution
- Topics (types) specified in IDL
- QoS regarding: reliability, urgency, priority, etc.

Features / Advantages
- Autonomous & loosely coupled applications
- Pub/Sub & QoS driven communication
- Strong-typed interfaces
- Smart networking based on priority & latency budget
DDS™ by Example: “The Ownership Profile …”

**Sensor-1**
- Publisher-1
  - Strength=2

**Sensor-2**
- Publisher-2
  - Strength=1

**Display**
- PointTrack Subscriber

OMG-DDS Real-time Information Backbone

**Characteristics**
- Replicated publishers of data (with own ‘strength’)
- Only highest-strength will be received
- On failure, next highest-strength will ‘take-over’

**Features / Advantages**
- Fault-tolerance by replication
- Notes:
  - Requires a lot of resources
  - Quality must be expressible as an ‘integer’
DDS™ by Example: “The Persistence Profile …”

Built-in Durability-Service

- Built-in persistence for non-volatile data
- State preservation for transient publishers
- Settings persistence surviving system downtime
- Replicated durability service for maximal fault-tolerance

OMG-DDS Real-time Information Backbone
DDS™ by Example: “The Persistence Profile…”

**Built-in Durability-Service**

- **PERSISTENT (on Disk)**
- **TRANSIENT (in Memory)**

**Characteristics**
- Built-in persistence for non-volatile data
- State preservation for transient publishers
- Settings persistence surviving system downtime
- Replicated durability service for maximal fault-tolerance

**Features / Advantages**
- **Case-1**: late-joining of Display process
  - Previously produced TrackStates readily available
- **Case-2**: restart of failed Classification process
  - Internal state (already classified tracks) regained

**DDS™ by Example**

```
TrackState {
  long trackId;
  Id identity;
}
```

**QoS**
- **Key**: trackId
- **QoS**: Reliable, Transient

**Display**

- PointTrack Subscriber
- TrackState Subscriber

**Case-1**
**Case-2**

**Characteristics**
- Built-in persistence for non-volatile data
- State preservation for transient publishers
- Settings persistence surviving system downtime
- Replicated durability service for maximal fault-tolerance

**Features / Advantages**
- **Case-1**: late-joining of Display process
  - Previously produced TrackStates readily available
- **Case-2**: restart of failed Classification process
  - Internal state (already classified tracks) regained

**OMG-DDS Real-time Information Backbone**

**Built-in Durability-Service**

**Classification**

- PointTrack Subscriber
- TrackState Publisher & Subscriber

**Settings persistence surviving system downtime**

**State preservation for transient publishers**

**Built-in persistence for non-volatile data**

**Transmit**

**Key**:
- `trackId`

**QoS**:
- Reliable, Transient
**DDS™ by Example: “The Content Subscription Profile …”**

**Characteristics**
- Adds ‘content awareness’
  - Content-filtered Topics & query-conditions
  - Supports ‘compound interest’
  - Multi-topics (combine/filter/re-arrange topics)

**Features / Advantages**
- Reduced application complexity
- SQL-like querying and reconstitution of related data
- Improved system performance
- Only receive/process what is of interest

**OMG-DDS Real-time Information Backbone**

**Display**
- Content-filtered Multi-topic Subscriber
Uses DCPS to distribute

Characteristics
- **Local** Object Oriented Data-Access Layer
- Supports ‘OO’ features:
  - Inheritance, aggregation, composition
- Uses DCPS to distribute **state** by ‘mapped topics’

Features / Advantages
- Ease of Management of (related) data
  - Object oriented ‘graphs of objects’ (value-types)
- Supports ‘native language constructs’ (I.e. navigation)
  - Automatic ‘change-management’ of objects
CORBA & DDS integration
CORBA & DDS

Server Behavior

- Distributed objects
  - Model: Client/Server
  - Remote Method Invocation
  - Reliable Communication
- Purpose:
  - Distributed Processing
  - Synchronous Transactions

CORBA

Data Availability

- Distributed data
  - Model: Publish/Subscribe
  - Distributed Information Access
  - Fine-grained QoS
- Purpose:
  - Real-time data distribution
  - Fault tolerant Info Management

DDS
**Characteristics & Benefits**

- **Corba/DDS Common Definition language: IDL**
  - Type definition for CORBA-interfaces & DDS topics
  - Code generation: Type-generation as well as generated (typed-)interfaces
- **Potential seamless Runtime Cooperation**
  - Shared types allow direct passing-on of RPC-obtained information into DDS-topics
  - Autonomous runtime-systems (ORB and DDS libraries)
CHATROOM EXAMPLE
Exercise – Traditional Chat architecture

Chat-Server

Chat-Client 1
Chat-Client 2
Chat-Client 3
Chat-Client 4
Exercise – DDS based Chat architecture

DDS-Chat Domain

Domain Participant

Chatter 1

Domain Participant

Chatter 3

Domain Participant

Chatter 2

Domain Participant

Chatter 4
Typical sequence of events on a **traditional** Chat-application:

- **Connect** to the Chat-Server.
- **Transmit** your identity.
- **Download** the identities of the other chatters.
- **Receive** chat messages from other users.
  - These messages are **forwarded** to you by the server.
- **Write** your own chat-messages.
  - These messages are **forwarded** by the server to all the other users.

Typical sequence of events on a **DDS-based** Chat-application:

- **Participate** in the Chat-domain.
- **Publish** your identity.
- **Subscribe** yourself to the identities of all other chatters
- **Subscribe** yourself to all chat-messages in the Chat-domain.
  - All messages are delivered to you **directly** by their respective writers.
- **Publish** your own chat-messages.
  - Your messages are **directly** delivered to all the other interested users.
**DEMO: DDS-based Chatroom**

**DDS-Chat Domain**

**Domain Participant**
- Chatter Hans: userID = 1
- Chatter Robert: userID = 2

**Message-Board**
- SHOWS ALL MESSAGES

**User-Load**
- TRACKS ALL CHATTERS

**NameService**
- int userID;
  - string Name
  - Key: userID
  - QoS: Reliable, Transient

**ChatMessage**
- int userID;
  - string Message
  - Key: userID
  - QoS: Reliable, Volatile

**DDS**
- TUNER
Demo DDS architecture: “OpenSplice” example

Choices: **Scalability & Efficiency**
- Single shared library for applications & services
- Ring-fenced shared memory segment
- Data urgency driven network-packing

Choices: **Determinism & Safety**
- Pre-emptive network-scheduler (priority pre-emptive network-threads per priority-band with traffic-shaping)
- Data importance based network-channel selection (Transport_Priority QoS of actual data)
- Partition based multicast-group selection (dynamic mapping of logical DDS partitions)
- Managed critical network-resource (limited impact/damage of faulty-applications)
MDE and DLRL: “Messageboard example”, Reducing Complexity!!

Managed info (DLRL) & MDE: a factor of 10!
CHATROOM LIVE DEMO