DDS: A Next-Generation Approach to Building Distributed Real-Time Systems

Gerardo Pardo-Castellote, Ph.D.
Co-chair OMG DDS SIG
CTO, Real-Time Innovations
gerardo.pardo@rti.com

The Real-Time Middleware Experts
2010 Masterclass
http://www.rti.com
Outline

- Overview of Technology
  - Background
  - Applications
  - Data-Centric Pub-Sub
  - Quality of Service
  - Add-on components

- Application development cycle
- Architecting data-centric systems & modeling the Data
- Protocol, Performance & Scalability
- Integrating external and legacy systems.
- Future directions and Standards:
Challenge: More Data, More Speed, More Sources

TRENDS:
- Growing Information Volume
- Lowering Decision Latency
- Increasing System Availability
- Accelerating technology insertion and deployment

Next-generation systems needs:
- Performance
- Scalability
- Robustness & Availability
- Platform Integration & Evolution
- Safety-Critical Certification
- Security
Solution: Standards-based Integration Infrastructure for Real-Time Applications

- Streaming Data
- Sensors
- Events

Data Distribution Service

- Real-Time Applications
- Enterprise Applications
- Actuators

© 2009 Real-Time Innovations, Inc.
Architecture for the next-generation systems

- Existing technologies are reaching robustness/performance/scalability limits
- RTI DDS brings a fundamental new architecture and approach
  - Fully decentralized, peer-to-peer, “no bottlenecks” architecture
  - Superior Wire Protocol
  - Powerful data-centric model
  - Built-in Robustness and High-Availability
  - Standards-based, multi-platform

Brokers as choke-points

Single-lane traffic
No prioritization

RTI Approach
History: DDS the Standards

- Data Distribution Service for Real-Time Systems
  - API for Data-Centric Publish-Subscribe distributed systems
  - Adopted in June 2003
  - Spec version 1.2: formal/2007-07-01

- Interoperability wire protocol
  - Adopted in July 2006
  - Revised in July 2007
  - Spec version 2.1: formal/2009-01-05

- Related specifications
  - UML Profile for DDS
  - DDS for Light-Weight CCM
  - Extensible Topics for DDS(*)

- Multiple (7+) Implementations
Open Architecture

- **Vendor independent**
  - API for portability
  - Wire protocol for interoperability

- **Multiple implementations**
  - 7 of API
  - 4 support RTPS (+1 non-DDS)

- **Heterogeneous**
  - C, C++, Java, .NET (C#, C++/CLI)
  - Linux, Windows, VxWorks, other embedded & real-time

- **Loosely coupled**

Cross-vendor portability

DDS API

Middleware

Real-Time Publish-Subscribe Wire Protocol (RTPS)

Cross-vendor interoperability
RTI DDS Application Examples

Aegis Weapon System
Lockheed Martin
Radar, weapons, displays, C2

B-1B Bomber
Boeing
C2, communications, weapons

Common Link Integration Processing (CLIP)
Northrop Grumman
Standards-compliant interface to legacy and new tactical data links
Air Force, Navy, B-1B and B-52

ScanEagle UAV
Boeing
Sensors, ground station

Advanced Cockpit Ground Control Station
Predator and SkyWarrior UAS
General Atomics
Telemetry data, multiple workstations

RoboScout
Base10
Internal data bus and link to communications center
RTI DDS Application Examples

Multi-ship simulator
FORCE Technology
Controls, simulation display

Mobile asset tracking
Wi-Tronix
GPS, operational status over wireless links

Highway traffic monitoring
City of Tokyo
Roadway sensors, roadside kiosks, control center

Driver safety
Volkswagen
vision systems, analysis, driver information systems

Medical imaging
NMR and MRI
Sensors, RF generators, user interface, control computers

Automated trading
Automated Trading Desk (ATD, now Citigroup)
Market data feed handlers, pricing engines, algorithmic trading applications
RTI DDS Application Examples

**Full-immersion simulation**
National Highway Transportation Safety Authority
Migrated from CORBA, DCOM for performance

**Signal Processing**
PLATH GMBH
RTI supports modular programming across product line

**Air-Traffic Management**
INDRA.
Deployed in
UK, Germany, Spain
Standards, Performance, Scalability

**Large Telescopes**
European Southern Observatory
Performance & Scalability
1000 mirrors, 1sec loop

**Industrial Control**
Schneider Electric
VxWorks-based PLCs communicate via RTI-DDS

**Radar Systems**
AWACS upgrade
Evolvability, Maintainability, and supportability
Standards Focus

- Object Management Group
  - Board of Directors member
  - Authored DDS and RTPS specs, co-chair SIG
- Open Group
- Network Centric Operations Industry Consortium (NCOIC)
  - Chair Open Standards and Patterns Working Group
- STAC Benchmark Council
- Support and integrate with:
  - DDS, RTPS, JMS, SQL, Web Services, CORBA, UML, HLA, JAUS, Eclipse, IPv6…
Corporate Background

- Founded by Stanford researchers
- Focused on real-time middleware
- Solid financials
  - 16-year track record of growth
- Real-Time Market Leader
  - #1 market share in embedded middleware of all types\(^1\)
  - 70+% worldwide share of DDS market\(^2\)
- 50/50 software and services

\(^1\)Embedded Market Forecasts
\(^2\)VDC Analyst Report
## RTI Supports all Phases of Development

<table>
<thead>
<tr>
<th>Services Capabilities</th>
<th>Engagement Timeline</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Workshop</td>
<td>2 days</td>
<td>Introduction to RTI products and capabilities</td>
</tr>
<tr>
<td>QuickStart</td>
<td>2+ days</td>
<td>In-depth training on RTI DDS API, QoS policies, and common architecture patterns</td>
</tr>
<tr>
<td>Support</td>
<td>On-Demand</td>
<td>Web-portal, phone and email customer lauded support</td>
</tr>
<tr>
<td>Architecture Study</td>
<td>3-4 weeks</td>
<td>Custom design review, risk analysis and architecture recommendations</td>
</tr>
<tr>
<td>Design Support Package</td>
<td>4+ weeks</td>
<td>Support hardware &amp; software integration, architecture design, performance tuning, on-site debugging, implementation support</td>
</tr>
<tr>
<td>Integration &amp; Development</td>
<td>SOW supported</td>
<td>Custom feature, tool and software development support</td>
</tr>
<tr>
<td>Ports</td>
<td>As needed</td>
<td>RTI tools and software on your special, purpose built hardware</td>
</tr>
</tbody>
</table>
Benefits of the DDS approach

Simple & Powerful Data-Centric Pub-Sub Model
- Reduces Risk and Development/Integration Time
- Enhances effective performance by delivering the right data to the right place with the right QoS
- Standards-based: API and Protocol

1. Unsurpassed Performance and Scalability
- Priority-aware no choke-points architecture

2. Builds higher quality systems and lowers TCO
- Built-in high-value capabilities
- Handles Availability & other “hard problems”
- Easy to maintain and Evolve
- Leverage multicore
Data-Centric Pub-Sub Model

Essentially a virtual, decentralized global data space

<table>
<thead>
<tr>
<th>Source (key)</th>
<th>Latitude</th>
<th>Longitude</th>
<th>Altitude</th>
</tr>
</thead>
<tbody>
<tr>
<td>UAV1</td>
<td>37.4</td>
<td>-122.0</td>
<td>500.0</td>
</tr>
<tr>
<td>UAV2</td>
<td>40.7</td>
<td>-74.0</td>
<td>250.0</td>
</tr>
<tr>
<td>UAV3</td>
<td>50.2</td>
<td>-0.7</td>
<td>2000.0</td>
</tr>
</tbody>
</table>

Persistence Service

Recording Service
“Global Data Space” generalizes Subject-Based Addressing
- Data objects addressed by **DomainId**, **Topic** and **Key**
- **Domains** provide a level of isolation
- **Topic** groups homogeneous subjects (same data-type & meaning)
- **Key** is a generalization of **subject**
  - **Key** can be any set of fields, not limited to a “x.y.z …” formatted string

- **Table**

<table>
<thead>
<tr>
<th>Sensorid</th>
<th>Value</th>
<th>Units</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>4535</td>
<td>23</td>
<td>Celsius</td>
<td>Build: 234 Rm 13</td>
</tr>
<tr>
<td>5677</td>
<td>14</td>
<td>Celsius</td>
<td>Build: 221 Fum 23</td>
</tr>
</tbody>
</table>

- **Diagram**

- **Data Writer**
- **Data Reader**
“Global Data Space” generalizes Subject-Based Addressing
- Data objects addressed by **DomainId, Topic** and **Key**
- **Domains** provide a level of isolation
- **Topic** groups homogeneous subjects (same data-type & meaning)
- **Key** is a generalization of **subject**
  - **Key** can be any set of fields, not limited to a “x.y.z …” formatted string
“Global Data Space” generalizes Subject-Based Addressing

- Data objects addressed by **DomainId**, **Topic** and **Key**
- **Domains** provide a level of isolation
- **Topic** groups homogeneous subjects (same data-type & meaning)
- **Key** is a generalization of **subject**
  - **Key** can be any set of fields, not limited to a “x.y.z …” formatted string
Demo: Publish-Subscribe

ShapesDemo
**DDDS communications model**

- **Participants** scope the global data space (domain)
- **Topics** define the data-objects (collections of subjects)
- **Writers** publish data on Topics
- **Readers** subscribe to data on Topics
- **QoS Policies** are used to configure the system
- **Listeners** are used to notify the application of events
Demo: Real-Time Quality of Service

- Content filter
- Time-based filter
- History
- Deadline
### Real-Time Quality of Service (QoS)

<table>
<thead>
<tr>
<th>QoS Policy</th>
<th>QoS Policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>DURABILITY</td>
<td>USER DATA</td>
</tr>
<tr>
<td>HISTORY</td>
<td>TOPIC DATA</td>
</tr>
<tr>
<td>READER DATA LIFECYCLE</td>
<td>GROUP DATA</td>
</tr>
<tr>
<td>WRITER DATA LIFECYCLE</td>
<td>PARTITION</td>
</tr>
<tr>
<td>LIFESPAN</td>
<td>PRESENTATION</td>
</tr>
<tr>
<td>ENTITY FACTORY</td>
<td>DESTINATION ORDER</td>
</tr>
<tr>
<td>RESOURCE LIMITS</td>
<td>OWNERSHIP</td>
</tr>
<tr>
<td>RELIABILITY</td>
<td>OWNERSHIP STRENGTH</td>
</tr>
<tr>
<td>TIME BASED FILTER</td>
<td>LIVELINESS</td>
</tr>
<tr>
<td>DEADLINE</td>
<td>LATENCY BUDGET</td>
</tr>
<tr>
<td>CONTENT FILTERS</td>
<td>TRANSPORT PRIORITY</td>
</tr>
</tbody>
</table>
20X Faster than JMS / Broker-based solutions

RTI DDS is about 20X faster than JMS

Throughput with a single publisher
(2KB messages)

CPU load [%]

RTI DDS
JMS

RTI DDS reliable multicast exhibits near perfect scalability

Platform: Linux 2.6 on AMD Athlon, Dual core, 2.2 GHz
DDS Is Scalable

- Going from 1 to 888 subscribers of the same data has only a 10% impact on throughput

- Ultra-low latency and jitter
  - Deterministic
  - No intermediaries

- DDS operates peer-to-peer, without brokers
- DDS uses RTPS, an Advanced Multi-Session protocol supporting Reliable Multicast
Pre-built components address many challenging use-cases

- Presence
- Discovery
- Historical Cache
- Durable Data
- Availability
- Redundancy & Failover
- Recording
- Database Connectivity
- Web Accessibility
- Transformation
- Event Processing
- WAN Routing
- Security Guard Hooks
Outline

- Overview of Technology
- Application development cycle
  - How to begin. Hello world example.
  - Defining data in XML and XSD
  - Development and Run-time Tools: Ping, Spy, Analyzer, Wireshark, Excel
  - Discovery and Builtin-Topics
  - Configuring QoS via XML files

- Architecting data-centric systems & modeling the Data
- Protocol, Performance & Scalability.
- Integrating external and legacy systems.
- Future directions and Standards:
Hands-on Example (C++)

Three minutes to a running app!!
1. Define your data
2. Create your project
3. Build
4. Run: publisher  subscriber

Aux:
File Browser
Console
Delete Files
rtiddsspy

- MyType.idl
- MyType.h
- MyTypeSupport.c
- MyTypePublisher.cpp
- MyTypeSubscriber.cpp
- MyType.sln
- rtiddsgen
- Publisher.exe
- Subscriber.exe
- compiler
Alternatives beyond IDL and CDR

Type Representation
“Foo” schema
IDL, XSD, ...

DDS-API
DataWriter<Foo>
Foo.h,
Foo.java
Language Bindings

Data Representation
10110011...
CDR, TLV,
JSON, XML

DDS-RTPS
Wire Interoperability

Type Representation
“Foo” schema
IDL, XSD, ...

DDS-API
DataReader<Foo>
Foo.h,
Foo.java
Language Bindings

Data Representation
10110011...
CDR, TLV,
JSON, XML

rtiddsgen
DDDS API
ddsgen
rtiddsgen supports 4 alternative ways to define types:
- All are equivalent
- You can convert between all these formats

- **IDL**
  - + Simple, Compact, Similar to C/C++/Java
  - + Allows type sharing with CORBA
  - - Perceived as “legacy”
  - - Limited tool support

- **XML**
  - + Good tool support and syntax validation
  - + Familiar to a large community. Fashionable
  - - More verbose. Custom Syntax

- **XSD**
  - + Good tool support
  - + Commonly used as a type-description language
  - - Cumbersome syntax for certain types. Not human friendly

- **WSDL**
  - + Same as XSD and allows type sharing with Web-Services
  - - Same as XSD

---

**Exercise:**
- Start with an IDL Type
  - Convert to XML
  - Convert to XSD
- Start with an XML-defined type
  - Convert to IDL
  - Convert to XSD
rtiddsgen Details

rtiddsgen [-d <outdir>] [-language <C|C++|Java|C++/CLI|C#>]
    [-namespace] [-package <packagePrefix>]
[-example <arch>] [-replace] [-debug]
[-corba [client header file]] [-optimization <level of optimization>]
[-stringSize <Unbounded strings size>]
[-sequenceSize <Unbounded sequences size>]
[-notypecode] [-ppDisable] [-ppPath <path to the preprocessor>]
[-ppOption <option>] [-D <name>=<value>]]
[-U <name>] [-I <directory>] [-noCopyable] [-use42eAlignment]
[-help] [-version] [-convertToIdl | -convertToXml | -convertToXsd | -convertToWsdl]

([-inputIdl] <IDLInputFile.idl> | [-inputXml] <XMLInputFile.xml> | [-inputXsd] <XSDInputFile.xsd> | [-inputWsdl] <WSDLInputFile.wsdl>]

- **DefinitionFile** can be IDL, XSD and XML file
- **-example** generates example pub/sub apps and makefiles for compilation.
- **-replace** replaces everything that’s generated. Use if the data type definition has changed. Always use with caution if you’ve made modifications.
struct MemberStruct{
    short sData;
}

typedef MemberStructType; //@top-level false
<?xml version="1.0" encoding="UTF-8"?>
<types xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:noNamespaceSchemaLocation="../rti.dds_topic_types.xsd">
  <struct name="MemberStruct"
          topLevel="false">
    <member name="sData" type="short"/>
  </struct>

  <typedef name="MemberStructType"
           type="nonBasic"
           nonBasicTypeName="MemberStruct"
           topLevel="false"/>
</types>
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"

targetNamespace="http://www.omg.org/IDL-Mapped/">
  <xsd:import namespace="http://www.omg.org/dds"
schemaLocation="rti_dds_topic_types_common.xsd"/>
  <xsd:complexType name="MemberStruct">
    <xsd:sequence>
      <xsd:element name="sData" minOccurs="1" maxOccurs="1"
type="xsd:short"/>
    </xsd:sequence>
  </xsd:complexType>
  <!-- @topLevel false -->

  <xsd:complexType name="MemberStructType">
    <xsd:complexContent>
      <xsd:restriction base="tns:MemberStruct">
        <xsd:sequence>
          <xsd:element name="sData" type="xsd:short" minOccurs="1" maxOccurs="1"/>
        </xsd:sequence>
      </xsd:restriction>
    </xsd:complexContent>
    <!-- @topLevel false -->
  </xsd:complexType>
</xsd:schema>
Data Persistence

A standalone service that persists data outside of the context of a DataWriter.

Can be configured for:
- Redundancy
- Load balancing
- Direct for performance
- Relay/Transactional
- Redundant/ Fault-tolerant
Data Persistence

A standalone service that persists data outside of the context of a DataWriter
Can be configured for:
• Redundancy
• Load balancing

Demo:
1. PersistenceService
2. ShapesDemo
3. Application failure
4. Application re-start
5. Persistence Svc failure
6. Application re-start
Ownership and High Availability

- Owner determined per Topic and Key
- Only writer with highest strength can publish a Key
- Automatic failover when highest strength writer:
  - Loses liveliness
  - Misses a deadline
  - Stops writing the subject

- Shared Ownership allows any writer to update any object
Outline

- Overview of Technology
- Application development cycle
- Architecting data-centric systems & modeling the Data
  - Examples: News example, Data Streaming, Commands, Video
  - Data Persistence with Examples
  - Using DynamicData
- Protocol, Performance & Scalability.
- Integrating external and legacy systems.
- Future directions and Standards:
Designing a Data-Centric System

- Define/Model the Global Data Space
- Configure the Cache Management
- Configure Discovery
- Configure the Protocol

- Configure/Use hooks for
  - Fault detection
  - Controlled access
Global Data Space / Global State

- Identify the number of domains

- Domain Information model
  - Topics
  - Types
  - Keys
  - Ownership
Domain and Domain Participants

- **N1 App 1**
  - Pub/Sub
  - (A,B,C,D)

- **N2 App 2**
  - Subscribe
  - (C)

- **N3 App 3**
  - Pub/Sub
  - (E,F/A,C)

- **N4 App 4**
  - Pub/Sub
  - (D/C,E,F)

- **N4 App 5**
  - Publish
  - (C)

- **N5 App 6**
  - Subscribe
  - (B,C)

**Domain**

**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”
Domain and Domain Participants

Using Multiple domains for Scalability, Modularity & Isolation

- **Node 1 - App 1**
  - Pub/Sub

- **Node 2 - App 1**
  - Subscribe

- **Node 3 - App 1**
  - Pub/Sub

- **Node 4 - App 1**
  - Pub/Sub

- **Node 4 - App 2**
  - Publish

- **Node 5 - App 1**
  - Subscribe

- **Node 5 - App 2**
  - Pub/Sub

- **Node 6 - App 1**
  - Pub/Sub

**Domain A**

**Domain B**

**Domain C**

Added Func.

---

demo_domain_0  Multiple Domain System  demo_domain_1
## Topics & Datatypes, Keys & Subjects

### Topic “MarketData”

<table>
<thead>
<tr>
<th>source</th>
<th>type</th>
<th>symbol</th>
<th>Exchange</th>
<th>volume</th>
<th>bid</th>
<th>ask</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPRA</td>
<td>IBM</td>
<td>NYSE</td>
<td>200000</td>
<td>118.30</td>
<td>118.36</td>
<td></td>
</tr>
<tr>
<td>OPRA</td>
<td>AAPL</td>
<td>NASDAQ</td>
<td>171.20</td>
<td>171.28</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RTFP</td>
<td>EQ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Additional fields (payload)

- demo_filters

### Topic “OrderEntry”

<table>
<thead>
<tr>
<th>Exchange</th>
<th>type</th>
<th>Symbol</th>
<th>Order num</th>
<th>number</th>
<th>limit</th>
<th>stop</th>
<th>expiration</th>
</tr>
</thead>
<tbody>
<tr>
<td>NYSE</td>
<td>BUY</td>
<td>IBM</td>
<td>11956</td>
<td>500</td>
<td>120</td>
<td>-</td>
<td>DAY</td>
</tr>
<tr>
<td>NYSE</td>
<td>BUY</td>
<td>IBM</td>
<td>11957</td>
<td>1000</td>
<td>124.5</td>
<td>124</td>
<td>DAY</td>
</tr>
<tr>
<td>NASDAQ</td>
<td>SELL</td>
<td>AAPL</td>
<td>11958</td>
<td>400</td>
<td>-</td>
<td>160</td>
<td>DAY</td>
</tr>
</tbody>
</table>
QoS: Ownership

Specifies whether more than one DataWriter can update the same instance of a data-object

Ownership = EXCLUSIVE
“Only highest-strength data writer can update each data-instance”

Ownership = SHARED
“All data-writers can each update data-instance”

Provides fast, robust, transparent replacement for fail-over and/or take-over.
QoS: Ownership Strength

Specifies which DataWriter is allowed to update the values of data-objects

OWNERSHIP_STRENGTH
“Integer to specify the strength of an instance”

Strength = 1
Data Writer “LEFT”
Publisher

Strength = 4
Data Writer “RIGHT”
Publisher

Domain Participant
“RIGHT” “LEFT”

Data Reader
Subscriber

Note: Only applies to Topics with Ownership = Exclusive
Configure the Cache Management

- **Cache State Content**
  - History
  - Lifespan
  - Persistence
  - Resources

- **Reader Cache View**
  - Partitions
  - Content-Based Filter
  - Time-Based Filter
  - Order
QoS: History – Last x or All

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

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

demo_history
QoS: Lifespan

User can set lifespan duration
Manages samples in the history queues, attached to each Sample

Data Writer
Publisher

Data Reader
Subscriber

Perm. Storage

lifespan_pub
lifespan_sub
<table>
<thead>
<tr>
<th>Instance</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>249</td>
</tr>
<tr>
<td>2</td>
<td>230</td>
</tr>
<tr>
<td>3</td>
<td>275</td>
</tr>
<tr>
<td>4</td>
<td>262</td>
</tr>
<tr>
<td>5</td>
<td>258</td>
</tr>
<tr>
<td>6</td>
<td>261</td>
</tr>
<tr>
<td>7</td>
<td>259</td>
</tr>
</tbody>
</table>

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

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

Domain Participant

Data Writer
Publisher

Data Reader
Subscriber

Discarded samples

minimum separation

Data Samples

time_filter_example
Cache Management in Action

Topics
- Square, Circle, Triangle
- Attributes

Data types (schemas)
- Shape (color, x, y, size)
  - Color is instance Key
- Key
  - Color field used for key

QoS
- History, Partition
- Time-Based Filter
- Content-Based Filter
Configure the Protocol

- Discovery
- Reliability
- Batching
- Liveliness
- Flow Control
- Asynchronous write
- Network Configuration
  - Enabled Transports + transport properties
  - Multicast addresses
  - Transport Priority
- OS settings
  - Threads
  - Memory
Tunable Reliability Protocol

- Configurable AckNack reply times to eliminate storms
- Fully configurable to bound latency and overhead
  - Heartbeats, delays, buffer sizes
- Performance can be tracked by senders and recipients
  - Configurable high/low watermark, Buffer full
- Flexible handling of slow recipients
  - Dynamically remove slow receivers

Reliable
- Guaranteed Ordered Delivery
- “Best effort” also supported

Publisher

Subscriber

Data Writer

Data Reader

S1
S2
S3
S4
S5
S6
S7
S8
S6
S7
S5
S4
S2
S1
Configure Notifications, Fault Detection & Management

- Listeners
- Deadline Qos
- Liveliness Qos
- Built-in Readers
- Notification of matching
QoS: Deadline

**DEADLINE** “deadline period”

- **Publisher**: Commits to provide data each deadline period.
- **Listener**: Failed to get data
  - **Subscriber**: Expects data every deadline period.

**Deadline Example**

```
S X S S S S S S
```
QoS: Liveliness – Type and Duration

Type: Controls who is responsible for issues of ‘liveliness packets’
AUTOMATIC = Infrastructure Managed
MANUAL = Application Managed

Failed to renew lease

lease_duration

LP
LP
LP
S

Liveliness Message
Exercise: How could “chat rooms” be implemented?

- Different Topics for each Chat room?
- Map to Partitions?
- Add field to the message and use content-filtered Topics?
- Same as before and also make room part of the Key?
- Others?

Discuss pros and cons of each approach
Exercise: How could we implement Ground control stations that monitor UAVs

- Different Topics for each UAV?
  - Or use Keys?
- Different Domains for each Ground Station?
  - Or Partitions?
- How to control multiple UAVs from the same ground station?
- How to switch the ground station that controls the UAV?
- How to do failover between ground stations?
- How to direct a message to one or all UAVs?
- How to detect loss of connection to an UAV?

Discuss pros and cons of each approach
Outline

- Overview of Technology
- Application development cycle
- Architecting data-centric systems & modeling the Data
- Protocol, Performance & Scalability.
  - Details on Reliable Protocol
  - Latency and Throughput
  - Using RTI’s LatencyTest and Perftest
  - Batching
  - Asynchronous writes & FlowController
  - Maximizing latency and Throughput
- Integrating external and legacy systems.
- Future directions and Standards:
RTI DDS is about 20X faster than JMS

RTI DDS reliable multicast exhibits near perfect scalability

Platform: Linux 2.6 on AMD Athlon, Dual core, 2.2 GHz
Extremely low latency and jitter

Reliable, ordered delivery over Gigabit Ethernet between 2.4 GHz Core 2 Quad processors running 32-bit Red Hat Enterprise Linux 5.0
Orders of magnitude more scalable than broker-based solutions

- Going from 1 to 888 subscribers of the same data has only a 10% impact on throughput
- New topics can be added to a system without impacting the latency and throughput on other topics
- Throughput with 8 topics is 8x the throughput with 1 topic

[Graphs showing throughput and latency improvements]

Realizing Performance & Scalability

- **RTI operates peer-to-peer, without brokers**
- **RTI uses RTPS, an Advanced Multi-Session protocol supporting Reliable Multicast**

Others: Broker-based middleware

- AMQP
- unspecified
Advanced Scalability & Performance Techniques

- Latency and Priority Aware message batching
- Content-Aware multi-channel reliable multicast
- Enhanced Reliable Protocol
  - Selective ACKs (SACKs) for Confirmed Reliability
  - NACK-only Reliable Protocol for Massive Scalability
- Smart caching integrated with the message protocol
- Content-Filtering at the source
Message Batching

Without batching each message is separately sent. For small messages protocol headers might be bigger than payload.

With batching messages are held a little and combined into larger batches maximizing throughout and minimizing CPU.

Transparent:
Receiver still sees individual messages.
Reliability with Batching

- Reliability must work even when messages are batched
- ACK or NACK of individual samples would negate some of the benefits of batching…
- => Protocol must be batch aware so that it can ACK/NACK complete batches!

Sender

```
write()
```

Receiver

```
B1
B2
B3
```

```
B1
```

```
B2
```

```
B3
```

```
B3
```

```
B3
```

```
B3
```

```
B2
```

```
B2
```

ACK(B3), NACK(B2)

Repair B2
Batching is hard but it pays!

RTI DDS 4.3b perftest results

Sample size (bytes)

Throughput (Mbps)

- Linux Baseline
- Linux 10Kb Batch

Intel Core2Duo Single-CPU Dual-Core 2.4GHz, 4MB cache
32-bit CentOS 5 (RHEL 5), 2GB memory, Intel E1000 NIC
Classic (TCP Style) Reliable Protocol
No packet loss situation

ShapesDemo
Packets 04 and 05 are received but the protocol drops them because a prior packet 03 is missing. This wastes valuable bandwidth.
RTI DDS Reliability (Reader Cache + SACK) improves performance when packet loss occurs

Packets 04 and 05 are received and cached waiting for the repair of 03.

No bandwidth is wasted.
RTI DDS NACK-only reliability eliminates ACK traffic if there is no packet loss.

No ACK traffic under normal operating conditions.
RTI DDS NACK-only reliability greatly reduces traffic even with packet loss

Negative Acknowledgments sent only when some message is lost

This approach is far more scalable when there are many subscribers
Asynchronous Publishing & Flow Controller

- synchronous send path:

- asynchronous send path:

1. rapid write() return
2. network traffic shaping
3. coalesce data samples
Qos Policies

- **DDS_PublishModeQosPolicy**
  - kind
    - `DDS_SYNCHRONOUS_PUBLISH_MODE_QOS`
    - `DDSASYNCHRONOUS_PUBLISH_MODE_QOS`
  - flow controller name

- **DDS_AsynchronousPublisherQosPolicy**
  - `disable_asynchronous_write <FALSE>`
  - `thread`
  - `disable_asynchronous_batch <FALSE>`
  - `asynchronous_batch_thread`
AP in Participant’s World

Publisher A
- Publishing Thread A

Publisher B
- Publishing Thread B

Publisher C
- Publishing Thread C

Publisher D
- Publishing Thread D

Participant
Flow Controller Token Distribution

- **basic token bucket**
  - steady-state traffic
    - tokens_added_per_period
    - token_period
  - max burst control
    - max_tokens
  - Additional controls
    - tokens_leaked_per_period
    - all values can be DDS_LENGTH_UNLIMITED
  - piggyback discount (Token Exchange Rate)
    - bytes_per_token

- **scheduling policy**
  - round-robin (RR)
  - earliest-deadline-first (EDF)
    - deadline = time of write + DDS_DataWriterQos::latency_budget
Using Asynchronous Publishing

- **DataWriter-side:**

```c
DDS_FlowControllerProperty_t property;

property.sched_policy = DDS_EDF_FLOW_CONTROLLER_SCHED_POLICY;
property.token_bucket.max_tokens = A; // [0, DDS_LENGTH_UNLIMITED]
property.token_bucket.tokens_added_per_period = B; // [0, DDS_LENGTH_UNLIMITED]
property.token_bucket.tokens_leaked_per_period = C; // [0, DDS_LENGTH_UNLIMITED]
property.token_bucket.bytes_per_token = D; // [1024, DDS_LENGTH_UNLIMITED]
property.token_bucket.period = E; // [0, DDS_DURATION_INFINITE]

DDSFlowController *controller =
    participant->create_flowcontroller("superflow", property);
...
writer_qos.publish_mode.kind = DDSASYNCHRONOUS_PUBLISH_MODE_QOS;
writer_qos.publish_mode.flow_controller_name = "superflow";
//
// Set up History queue size to hold deferred issues!!!!!.
writer_qos.history.kind = DDS_KEEP_LAST_HISTORY_QOS;
writer_qos.history.depth = z; // <<<<<<<<<<<<<<<<<<<<<!!!!!!!!!
...
writer->write(data_sample, ...);
... // Optional wait for pipe to empty
writer->wait_for_asynchronous_publishing(timeout);
... // Optional On-Demand trigger
controller->trigger_flow();
... // Optional Modify controller properties
controller->set_property();
```

- **No changes on DataReader-side!!**
Flow Controller Design Challenge

- Requirements;
  - Large 1 mbyte issue.
  - Transmit over period of 10 seconds
  - Low priority transmission
  - Transport buffer size set to 32K

- CONTROLLER:

  ```
  property.sched_policy = ??;
  property.token_bucket.max_tokens = ??
  property.token_bucket.tokens_added_per_period = ??
  property.token_bucket.tokens_leaked_per_period = ??;
  property.token_bucket.bytes_per_token = ??;
  property.token_bucket.period = ?
  ```
Flow Controller Design Challenge

- Requirements;
  - Large 1 mbyte issue.
  - Transmit over period of 10 seconds
  - Low priority transmission
  - Transport buffer size set to 32K
  - Cannot loose any issues

- CONTROLLER:

```plaintext
property.sched_policy = DDS_RR_FLOW_CONTROLLER_POLICY;
property.token_bucket.max_tokens = 1
property.token_bucket.tokens_added_per_period = 1
property.token_bucket.tokens_leaked_per_period = unlimited;
property.token_bucket.bytes_per_token = 32k;
property.token_bucket.period = 200ms
```

- Extra Credit Discussion:
  What about reliable protocol properties?
Outline

- Overview of Technology
- Application development cycle
- Architecting data-centric systems & modeling the Data
- Protocol, Performance & Scalability.
- Integrating external and legacy systems.
  - Routing Service
  - Systems of Systems
  - Cross Domain Solutions
  - Accessing Data over a WAN
  - Database Connectivity
  - Access over the Web

- Future directions and Standards:
Real-Time Recording Service

- Applications:
  - Future analysis and debugging
  - Post-mortem
  - Compliance checking
  - Replay for testing and simulation purposes

- Record high-rate data arriving in real-time
- Non-intrusive – multicast reception

Demo:
1. Start RecordingService
2. Start ShapesDemo
3. See output files
4. Convert to: HTML XML
5. View Data: HTML XML

sqlite stop_all
Relational Database Integration

Publish-Subscribe Action
- Write()
- Read() & Take()
- Dispose()
- Wait() & Listener

Relational Actions
- UPDATE [2,3] & INSERT
- SELECT
- DELETE

ShapesDemo

Event driven – The fastest way to observe database changes!

1. start mysql
2. start RTC

stop all

sql gui
sql shell

Table T1
- I1
- I2
- I3

RTI Real-Time Connect
COTS tools: **Excel** – Interacting with your data

- Display live RTI DDS Data in Excel
- Perform real-time computations and charts
- Publish RTI DDS data from Excel
RTI Routing Service

- Selective, real-time data forwarding and transformation
- Can Change Topic Name and Topic Schema
  - Allows for custom transformations via “plugin”
  - Can filter/guard data
- QoS managed, can cache last-known value for data
- Dynamically configured
- Location independent deployment
Global Scalability: LAN to WAN... ...without sacrificing Performance and Security

Topics:
- Site Status
- Alarms
- Health Logs
- Sensor Data

Site A

Topics:
- Site Status
- Sensor Data

Site B

WAN / Internet TCP/TLS/SSL

Topics:
- Site Status
- Proc Sensor Data
- Result Data
- Alarms

Site C

Topics:
- Site Status
- Result Data

Site D
Direct access to real-time data from Web-Based Applications

1. start replay & router
2. start viewing maps

Web Enabled DDS

Recorded Data
Tactical Real-Time Data

simulated tracks

GUARD

Recorded Track Files

stop all
Outline

- Overview of Technology
- Application development cycle
- Architecting data-centric systems & modeling the Data
- Protocol, Performance & Scalability.
- Integrating external and legacy systems.
- Future directions and Standards:
  - Extensible Topics for DDS
  - Web Enabled DDS
  - Standard C++ PSM for DDS
- Q&A
Extensible Dynamic Types Submission

class Overview

TypeModel::Type

Data

TypeRepresentation::TypeRepresentation

LanguageBinding::TypeLanguageBinding

LanguageBinding::DataLanguageBinding

DataRepresentation::DataRepresentation
Example: Current mechanisms

Type Definition

IDL:
Foo.idl

struct Foo {
    string<> name;
    long ssn;
};

Language Binding

IDL to Language Mapping:
Foo.h
Foo.cpp
FooTypeSupport.cpp

struct Foo {
    char *name;
    int ssn;
};

Foo f = {"hello", 2};

Data Representation

IDL to CDR:
00000006
68656C6C
6F000000
00000002
Type-Definition – Language Representation – Serialized encapsulation ... Each offers options

Type Representation
“Foo” schema IDL, XSD, ...

Language Bindings
Foo.h, Foo.java

Data Representation
10110011...
CDR, TLV, JSON, XML

DDS-API
DataWriter<Foo>

DDSGen

DDS-API
DataReader<Foo>

DDS-RTPS
Wire Interoperability

Type Representation
“Foo” schema IDL, XSD, ...

Language Bindings
Foo.h, Foo.java

Data Representation
10110011...
CDR, TLV, JSON, XML
Web-Enabled Data-Centric Global Data Space

- Stateless access of data via application appropriate technologies and protocols
- Not a bridge, broker, or message router
### Web Enabled DDS

<table>
<thead>
<tr>
<th>App</th>
<th>Std. Web protocols</th>
<th>Web Enabled DDS</th>
<th>App</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web</td>
<td>HTTP</td>
<td>Web</td>
<td>DDS</td>
</tr>
<tr>
<td>HTTP</td>
<td></td>
<td>HTTP</td>
<td>RTPS</td>
</tr>
</tbody>
</table>

A service that exposes DDS Global Data over Web Protocols:
Applications can interact with DDS directly over the Web
No need for bridges or special bindings for scripting languages
Day 2: Exercises

Gerardo Pardo-Castellote, Ph.D.
Co-chair OMG DDS SIG
CTO, Real-Time Innovations
gerardo.pardo@rti.com

The Real-Time Middleware Experts
http://www.rti.com
Preparations

- **Install RTI DDS:**
  - **Windows**
    - Unzip: RTI_Masterclass2GO.zip
      - into directory C:\RTI
    - Execute: install_actions.bat
  - **Linux**
    - Boot your computer from the USB

- **Test you can do the following**
  - rtiddsgen -help
Install VisualStudio from the ISO’s

- Copy the VS2008 and WindowsSDK ISO’s
- Install DaemonTools
- Mount the ISOs as virtual drives
- Proceed with the installation
  - 1st the VS2008
  - 2nd the Platform SDK
- Test the installation by creating a “hello world” project compiling and running it
Exercise #0  - Hello World

Define your data type:

- Create a directory “HelloWorld”
- Create a file called hello.idl and open it in VisualStudio
- Add the following contents:

```cpp
const long MSG_LEN=256;
struct HelloMsg {
    string<MSG_LEN> user; // @key
    string<MSG_LEN> msg;
};
```
Run rtiddsgen (for C++)

- rtiddsgen hello.idl -language C++ -example i86Win32VS2005 \  
  -replace -ppDisable
- rtiddsgen hello.idl -language Java -example i86Win32jdk \  
  -replace -ppDisable

- Look at the directory you should see:
  - hello-vs2005.sln
  - And Several other files…

- Open the Solution File (type hello-vs2005.sln on the console)
  - Look at HelloMsgPublisher.cxx
  - Look at HelloMsgSubscriber.cxx

- Build the Solution
Run rtiddsgen (for Java)

- rtiddsgen hello.idl -language Java -example i86Win32jdk \
  -replace -ppDisable

- Look at the directory you should see:
  - makefile_hello_i86Win32jdk
  - And Several other files…
    • Look at HelloMsgPublisher.java
    • Look at HelloMsgSubscriber.java

- You can use the makefile to build and the Java programs:
  - gmake –f makefile_hello_i86Win32jdk
Execute the program

- **C++:**
  - On one window run:
    - `objs\i86Win32VS2005\HelloMsgPublisher.exe`
  - On another window run:
    - `objs\i86Win32VS2005\HelloMsgSubscriber.exe`

- **Java**
  - On one window run:
    - `gmake –f makefile_hello_i86Win32jdk HelloMsgPublisher`
  - On another window run:
    - `gmake –f makefile_hello_i86Win32jdk HelloMsgSubscriber`

- You should see the subscribers getting an empty string…
Modify the program to produce something

- **C++**: Open HelloMsgPublisher.cxx in VisualStudio
- **Java**: Open HelloMsgPublisher.java in your preferred tool

- Look for the comment:
  ```
  /* Modify the data to be sent here */
  ```

- Add the line:
  ```
  strcpy_s(instance->msg, MSG_LEN,
          "Hello this is gerardo");
  ```

  Use your own name instead of “gerardo”

- Kill the Publisher, Rebuild the publisher and run it again
Playing with rtiddsspy

- Run rtiddsspy while the other applications are running
- Start and stop applications. What do you see in rtiddsspy
Exercise #1 – Shapes Publisher

- Create a new directory Shapes
- In the Directory create a file called ShapeType.idl
- Edit the file to have the following content:

```c++
const long COLOR_LEN=64;
struct ShapeType {
    string<COLOR_LEN>color; //@key
    long x;
    long y;
    long shapesize;
};
```

- Run:

```bash
rtiddsgen ShapeType.idl -language C++ -example
i86Win32VS2005 --replace -ppNotRun
```
Exercise #2 – Using keys

- Create a new directory Chat
- In the Directory create a file called chat.idl
- Edit the file to have the following content:

```cpp
const long NAME_LEN=64;
const long MSG_LEN=256;
struct ChatMsg {
    string<NAME_LEN> name; // @key
    long age;
    string<MSG_LEN> chatRoom;
    string<MSG_LEN> msg;
};
```

- Run:
  rtiddsgen chat.idl -language C++ -example i86Win32VS2005 – replace -ppNotRun
Edit the chat_publisher.cxx

- Go to the line with comment: /* Main loop */
  - Add the line:
    ```
    strcpy_s(instance->name, NAME_LEN, "Gerardo Pardo");
    ```
  (Use your own name)

- Go to the line with comment:
  - /* Modify the data to be sent here */
  - Add the lines:
    ```
    instance->age = count;
    strcpy_s(instance->msg, NAME_LEN, "Como va todo?");
    ```
  (Use your age and personalized message)

- Rebuild and execute
Exercise #3 Use Qos

- Set RELIABILITY
- Set HISTORY to KEEP_LAST or KEEP_ALL
  - Test different depths
- Use Partitions
  - Create several Partitions:
    - E.g. by ChatRoomName
  - Publish in your ChatRoom
  - Subscribe to one or more ChatRooms
Exercise #4 Use content filters

- Edit the chat_subscriber.cxx
- Add the lines:

```cpp
DDSCoerntFilterTopic *cfTopic;
DDS_StringSeq filter_params;
filter_params.maximum(0);
cfTopic = participant->
    create_contentfilteredtopic(
        "Selected Chats", topic,
        "age > 4", filter_params);
```

- Look of the call to create_datareader
  - Replace “topic” with “cfTopic” in the parameter list.
Exercise #5 Use Exclusive Ownership

- Set up in pairs edit the chat_publisher.cxx and use the same “name” for both of you.
- Re-run the publisher application you will see mixed messages.
- Edit the chat_publisher.cxx
  - Before creating the data writer add the lines:
    ```
    publisher->get_default_datawriter_qos(dwq);
    dwq.ownership.kind = DDS_EXCLUSIVE_OWNERSHIP_QOS;
    dwq.ownership_strength.value = 10;
    ```
  - Replace DDS_DATAWRITER_QOS_DEFAULT with dwq in the create_datawriter() call.
- Edit the chat_subscriber.cxx
  - Before creating the data reader add the lines:
    ```
    DDS_DataReaderQos drq;
    subscriber->get_default_datareader_qos(drq);
    drq.ownership = DDS_EXCLUSIVE_OWNERSHIP_QOS;
    ```
  - Replace DDS_DATAWRITER_QOS_DEFAULT with drq in the create_datareader() call.
Summary

- Reduces software lifecycle costs
  - Loose coupling
  - Replaces need for custom middleware in high-performance, real-time applications

- Reduces risk
  - Standards-compliant API and wire protocol
  - Multiple implementations
  - Widely adopted

- Most widely proven and mature implementation
- Highest performance
- Industry-leading expertise and services capability
- Free trial, research and IR&D licenses
- Comprehensive VxWorks support