Analysis of the Advanced Message Queuing Protocol (AMQP) and comparison with the Real-Time Publish Subscribe Protocol (DDS-RTPS Interoperability Protocol)

July 2007

Gerardo Pardo-Castellote, Ph.D.
CTO, Real-Time Innovations, Inc.
gerardo@rti.com
Angelo Corsaro, Ph.D.
angelo.corsaro@gmail.com

www.rti.com
Background: AMQP

- AMQP = Advanced Message Queuing Protocol

A protocol to communicate between clients and messaging middleware servers (brokers)

- A protocol developed by an industry consortia:
  - IONA
  - JP Morgan
  - RedHat
  - 29West
  - Others

- The goals are:
  - Support Messaging semantics of financial industry
  - Provide needed performance to financial industry
  - Be extensible to new queuing and routing
  - Allow service to be programmed by the application

- High-level Requirements:
  - To accommodate existing messaging API standards (for example, Sun’s JMS)
  - To allow complete configuration of server wiring via the protocol

- Low-level Design Requirements:
  - To be compact, using a binary encoding that packs and unpacks rapidly
  - To handle messages of any size without significant limit
  - To carry multiple channels across a single connection
  - To be longlived, with no significant inbuilt limitations
  - To allow asynchronous command pipelining
Background: DDS-RTPS

- DDS-RTPS = Data-Distribution Service Real-Time Publish Subscriber Protocol
  
  *The standard protocol used by the OMG Data-Distribution Service Specification*

- An OMG standard developed and supported by middleware vendors and users:
  - RTI, PrismTech, THALES, SELEX, NSWC-DD, MITRE, OCI, MilSOFT, etc.

- The goals are:
  - Support Real-Time Publish-Subscribe Middleware
  - Deliver high performance and scale to large systems
  - Support packet based unrealizable transports, including multicast
  - Support Quality of Service
  - Provide for forward and backwards compatibility
  - Provide automatic discovery

- High-level Requirements:
  - To accommodate the OMG DDS specification and all its Qos

- Low-level Design Requirements:
  - To be compact, using a binary encoding that packs and unpacks rapidly
  - To handle messages of any size without significant limit
  - To carry multiple channels across a single connection
  - To be longlived, with no significant inbuilt limitations
  - To allow asynchronous command pipelining
Agenda

- Middleware Models and Protocols
  - Service Models
  - Protocol
- Comparison
  - Communications Model
  - Object Model
  - Architecture Model
  - Protocol
- Conclusion
Middleware = Service Model + Protocol

- Service model composed of:
  - Communications model
  - Object Model
  - Architecture model
- All these interact
- The service model and protocol are also coupled

Protocols cannot be compared in isolation! They must be compared in the context of the service model.
Middleware Service Model

Service Model =  Communications Model
    + Object Model
    + Architecture Model

- Communications Model:
  - Abstract model of how applications interact:
    - Queue Based
    - Data-Distribution
    - Replicated Data
    - Distributed Transactions
    - Remote Method Invocation

- Object Model
  - Middleware entities the application uses to interact with the service:
    - Queues, Publishers, Domains, Caches, Federations, Remote Objects…

- Architecture Model
  - Centralized, Brokered, Peer-to-Peer
Centralized Broker Pub-Sub Service

One central server materializes all middleware entities
All traffic flows via server
E.g. “naïve” implementations of JMS, CORBA Notification, etc.
Centralized Multi-Broker Pub-Sub Service

Each Queue/Topic Can be placed on a different Server
E.g. Better implementations of JMS, CORBA Notification, etc.
De-centralized Brokered Pub-Sub Service

App uses messaging or RMI to interact with Service Access points
Pub-Sub Service distributes messages internally between servers

**Internally PS-Service can be peer-to-peer, hub-and-spoke, multicast, etc.**
De-Centralized Unbrokered Pub-Sub Service

App links (binds) directly with the Pub-Sub service
Queuing occurs locally on each client
Clients communicate peer-to-peer
## Service Model Architecture Examples

<table>
<thead>
<tr>
<th>Model</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Centralized Brokered</strong></td>
<td>Typical JMS implementations</td>
</tr>
</tbody>
</table>
| **Centralized Multi-Brokered**| Better JMS implementations  
CORBA event & notification service |
| **De-centralized Brokered**   | TIBCO RendezVous  
TIBCO SmartSockets  
IBM WebSphere MQ (MQSeries) using client connection |
| **De-centralized Un-brokered (Peer-to-peer)** | Most DDS implementations: RTI DDS, OpenSplice, Tao-DDS  
IBM WebSphere MQ (MQSeries) using Binding connection |
Agenda

- Middleware Models and Protocols
  - Service Models
  - Protocols
- Comparison
  - Communications Model
  - Object Model
  - Architecture Model
  - Protocol
- Conclusion
There are interactions between the service model and the Protocol.

A Brokered Service model requires 2 protocols:
- Client Protocol (used by client applications)
- Service Protocol (used between the Brokers)

A fully peer-to-peer middleware requires only one protocol.
Protocol layers: Middleware perspective (proposed)

The existing middleware perspectives are inadequate to model middleware protocols:

The 7-layer OSI model:
- Has layers with very limited functionality: (Session Presentation)
- Considers TCP and UDP both “layer 4 concepts”

The 5-layer TCP/IP model:
- Combines all layers above Transport into single “Application Layer”
- Considers TCP and UDP equivalent

Some protocols may expand multiple layers
Protocol layers (Middleware perspective)

- **Layer 6**: Application operations
  - Invoke Operations with known semantics
  - Send messages to queues
  - Pub/Sub specific topics

- **Layer 5**: Middleware operations
  - Multi-layer protocols
  - HTTP
  - GIOP
  - SOAP
  - FTP
  - TLS
  - A middleware protocol at this Level must operate on top of a Layer 4 protocol like TCP

- **Layer 4**: Reliability, Flow Control, Packet Assembly
  - SCTP
  - TCP
  - RTP
  - PGM
  - DTLS
  - STUN
  - A middleware protocol at this Level can operate on top of UDP or Multicast

- **Layer 3**: Packet delivery & routing
  - UDP
  - IP
  - IP Multicast
Middleware Comparison Dimensions

Service Model:
- Communication Model
- Object Model
- Architecture Model (+ required brokers)

Protocol:
- Layer
- Features
- Transport Assumptions
- Overhead

Impact/Capabilities:
- Features: Reliability, Persistence, QoS, Batching
- Impact:
  - Deployment
  - Performance (protocol overhead, #intermediaries)
  - Robustness (failure conditions)
## AMQP vs. DDS-RTPS

<table>
<thead>
<tr>
<th>Aspect</th>
<th>AMQP</th>
<th>DDS-RTPS</th>
</tr>
</thead>
</table>
| **Communications** (service model) | Message Queues  
Globally addressable queues | Data Distribution  
Global Data Space |
| **Object Model** (service model) | AMQ Broker  
Exchange  
Binding  
Queue  
Routing Key | DomainParticipant  
Topic  
Publisher Subscriber  
DataWriter DataReader |
| **Architecture** (service model) | Clients and Brokers  
Queues Centralized in Brokers | Peer-to-Peer  
No Intermediaries |
| **Protocol** | AMQP – Layer 5 only  
(only middleware actions) | RTPS – Layer 4 and 5  
Middleware actions  
Reliability, Flow control, Fragmentation |
Agenda

- Middleware Models and Protocols
  - Service Models
  - Protocol
- Comparison
  - Communications Model
  - Object Model
  - Architecture Model
  - Protocol
- Conclusion
Provides a “Shared Queue Space” that is accessible to all interested applications.

- **Message** are sent to an **Exchange**
- Each message has an associated **Routing Key**
- **Brokers** forward messages to one or more **Queues** based on the **Routing Key**
- Subscriber get messages from **named Queues**
- Only **one subscriber** can get a given message from each **Queue**
DDS Service Model: Communication Model

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

- Data objects addressed by **Domain**, **Topic** and **Key**
- Subscriptions are **decoupled** from Publications
- Contracts established by means of **QoS**
- Automatic **discovery** and **configuration**
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
Agenda

- Middleware Models and Protocols
  - Service Models
  - Protocol
- Comparison
  - Communications Model
  - Object Model
  - Architecture Model
  - Protocol
- Conclusion
**AMQ Service Model : Object Model**

- **Exchange** – Receives messages and routes to a set of message queues
- **Queue** – Stores messages until they can be processed by the application(s)
- **Binding** – Routes messages between Exchange and Queue. Configured externally to the application
  - Default binding maps routing-key to Queue name
- **Routing Key** – label used by the exchange to route Content to the queues
- **Content** – Encapsulates application data and provides the methods to send receive, acknowledge, etc.
AMQ Queues

- Stores and distributes messages
- Each message delivered to a single client consumer

Properties (on creation):
- **Name**
  - Client Named or Server Named
- **Durable**
  - Durable remains present after re-start
    - But may lose non-persistent messages
- **Auto-delete**
  - Will auto-delete when all clients have finished using it
- **Private (Exclusive)/Shared**
  - Private (Exclusive) ⇔ read by a single consumer
AMQ Binding & Routing Key

- **Binding**
  - Tells exchange how to route messages:
    ```
    Queue.Bind <queue> TO <exchange> WHERE <condition>
    ```
- **The `<condition>` can involve:**
  - Message Properties
  - Header fields
  - Content
  - In most cases uses a single field: the “routing key”
- **Routing key = virtual address used in the binding**
  - For Point2point routing-key = name of msgQ
  - For Topic PubSub routing-key = topic hierarchy value
  - In other cases routing-key may be combined with msg header and content
AMQ Content

- Client producer creates message
- Producer fill content, properties and routing information
- Producer sends msg to Exchange
- Exchange route msg to set of Queues. Each is treated as a separate copy (no common identifier)
- Queue passes message to a single consumer if present or else buffers it.
  - Upon ‘delivery’ msg removed from queue.
- 2 kind of acks: Automatic or Explicit
  - Explicit requires app to indicate so for each message
DDS Service Model: Object Model

- **DomainParticipant** – Allows application to join a DDS Domain (Global Data Space)
- **Topic** – A string that addresses a group of objects in the Global Data Space
  - Each Object is identified by a Key (some fields within the object data)
- **Publisher, Subscriber** – Pools resources for DataWriters and DataReaders
- **DataWriter** – Declares intent to publish a Topic and provides type-safe operations to write/send data
- **DataReader** – Declares intent to subscribe to a Topic and provides type-safe operations to read/receive data
Agenda

- Middleware Models and Protocols
  - Service Models
  - Protocol
- Comparison
  - Communications Model
  - Object Model
  - Architecture Model
  - Protocol
- Conclusion
Centralized Multi Broker

Each Queue is materialized in a Broker
DDS-RTPS: Unbrokered Pub-Sub Service

Unbrokered Peer-to-Peer
Each application links (binds) directly with the Pub-Sub service
“Queuing/Cacheing” occurs locally on each client
# Model tradeoffs for RT systems

<table>
<thead>
<tr>
<th>Model</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Centralized Brokered</strong></td>
<td>Simplest to implement&lt;br&gt;Fewest network connections per client</td>
<td>Single point of failure&lt;br&gt;Server is bottleneck&lt;br&gt;Poor performance&lt;br&gt;Poor predictability&lt;br&gt;Does not scale</td>
</tr>
<tr>
<td><strong>Centralized Multi-Brokered</strong></td>
<td>Simple to implement&lt;br&gt;Scales to many Queues</td>
<td>Hard to administer&lt;br&gt;Potentially many connections per client&lt;br&gt;Servers can be bottlenecks&lt;br&gt;Single point of failure for any queue&lt;br&gt;Not scalable to many readers/writers per queue</td>
</tr>
<tr>
<td><strong>De-centralized Brokered</strong></td>
<td>Scales to large # queues&lt;br&gt;Scales to large # readers/queue&lt;br&gt;Low number of connections&lt;br&gt;No single points of failure</td>
<td>Worst latency&lt;br&gt;Worst predictability</td>
</tr>
<tr>
<td><strong>De-centralized Un-brokered (Peer-to-peer)</strong></td>
<td>Scales to large # queues&lt;br&gt;Scales to large # readers/queue&lt;br&gt;Best latency&lt;br&gt;Best predictability&lt;br&gt;No single points of failure</td>
<td>Most difficult to implement&lt;br&gt;Requires use of UDP to avoid large # connections</td>
</tr>
</tbody>
</table>
Agenda

● Middleware Models and Protocols
  – Service Models
  – Protocol

● Comparison
  – Communications Model
  – Object Model
  – Architecture Model
  – Protocol

● Conclusion
AMQP vs DDS-RTPS Protocols: Scope

AMQP:
- Queues
- Exchanges
- RoutingKeys

DDS:
- Domain
- Topic
- Key

Client Protocol:
- Service Protocol

AMQP
- Not Specified!

DDS
- Not Needed!
AMQP vs DDS-RTPS Protocol: Layers

- **Layer 6**: Application operations
  - Invoke Operations with known semantics
  - Send messages to queues
  - Pub/Sub specific topics

- **Layer 5**: Middleware operations
  - HTTP
  - GIOP
  - SOAP
  - FTP
  - TLS
  - SCTP
  - TCP
  - RTP
  - PGM
  - DTLS
  - STUN

- **Layer 4**: Reliability, Flow Control, Packet Assembly
  - UDP
  - IP
  - IP Multicast

- **Layer 3**: Packet delivery & routing

- **Layer 2**: Protocol layers
  - DDS-RTPS
  - AMQP
## AMQP vs. DDS-RTPS Protocols

<table>
<thead>
<tr>
<th>Aspect</th>
<th>AMQP</th>
<th>DDS-RTPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reliability</td>
<td>NO: Must be Provided by Transport</td>
<td>YES: Ack and Nack based Configurable Heartbeats</td>
</tr>
<tr>
<td>Fragmentation</td>
<td>NO: Must be Provided by Transport</td>
<td>YES: Fragmentation fully supported Reliability supported at the fragment level</td>
</tr>
<tr>
<td>Discovery</td>
<td>Not defined by protocol</td>
<td>Defined by protocol</td>
</tr>
<tr>
<td>Multiplexed connections, Batching</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Expected deployment</td>
<td>Up to 500 msgs/sec per broker</td>
<td>Up to 100000 msgs/sec per peer</td>
</tr>
<tr>
<td>Transport</td>
<td>Connection-Oriented (Assumes TCP)</td>
<td>RTPS – Layer 4 and 5 Middleware actions Reliability, Flow control, Fragmentation</td>
</tr>
<tr>
<td></td>
<td>No UDP, No Multicast</td>
<td></td>
</tr>
</tbody>
</table>
Conclusion

- Protocols must be compared in the context of the Middleware Service model they support

- AMQP developed to support a Brokered model.
  - Layer 5 protocol that defines only Client 2 Broker communications
  - Inter-Broker communications undefined
  - Assumers TCP-like transport

- DDS-RTPS developed to support un-brokered fully peer-to-peer model
  - Layer 4 + 5 protocol defines peer-to-peer interactions
  - All communication aspects defined
  - Assumes datagrams only. Supports UDP and Multicast

- DDS-RTPS far more suitable for high-performance and real-time systems
  - No intermediate brokers => better latency and predictability
  - No bottlenecks or single points of failure
  - Supports Multicast and unreliable channels (e.g. wireless links with dropouts)