From SPLICE and DDS to ROS2 and Eclipse Cyclone DDS

Erik Boasson
Sr. Technologist
The briefest of recaps: ROS1 & ROS2

• ROS1 was created in 2007 by Willow Garage as a platform for robotics research
  • Moved into stewardship by the Open Source Robotics Foundation
  • Vibrant ecosystem of tools, algorithms and drivers for constructing robots
  • Much used for research, education, prototypes

• ROS2 is a rewrite of ROS1 driven by OSRF, first released 2017
  • Addresses the weaknesses in the architecture of ROS1
  • Maturing nicely, ecosystem is migrating to it
  • Gaining ground in commercial and industrial robotics
ROS1 & ROS2 Architecture

- **Application Layer**:
  - ROS1: Master, Node, Node
  - ROS2: Node, Node, Node

- **Middleware Layer**:
  - ROS1: Client Library, TCPROS/UDPROM, Nodelet API
  - ROS2: Client Library, Abstract DDS Layer, DDS, Intra-process API

- **OS Layer**:
  - ROS1: Linux
  - ROS2: Linux/Windows/Mac/RTOS
Areas where ROS2 improves over ROS1

limited to communications component

• Graceful degradation under load
• Efficient fan-out
• Zero configuration
• Resilience to failures
• Allowing ROS contributors to focus on the robotics side
Why DDS? Why not ...?

So many options to choose from … some obviously unsuited, like:

• Point-to-point message passing
• Shared Data Space / DBMS
• Software Distributed Shared memory
Why DDS? Why not ...?

Some rather attractive ones

- Brokered Publish-Subscribe
- Peer-to-peer Publish-Subscribe
- Eventually consistent Shared Data Space

ROS1 used ~brokered P/S messaging
P2P is better all-around at this scale
DDS appears twice!
SPLICE, NDDS & DDS
DDS’ Two Parents
or: where the two “personalities” come from

• SPLICE — eventually-consistent shared data space
  • Hollandse Signaalapparaten, now Thales Netherlands; Maarten Boasson, ca. 1984
  • Designed to address the complexity of building naval combat management systems
  • Publish-subscribe used as an implementation technique

• NDDS — publish-subscribe messaging (*)
  • Real-Time Innovations, Inc.; Gerardo Pardo-Castellote and Stan Schneider, ca. 1995
  • Designed to address the problems faced in the Robotics Lab at Stanford, early ‘90s
  • (*) based on my understanding of RTI’s history

• ROS2 today is very much like NDDS, albeit with a much nicer API
Similarities and differences

• **Similarities**
  - Low latency
  - High throughput
  - Data flow matters more than data storage
  - Dynamic discovery

• **Differences: SPLICE also directly targeted**
  - Fault-tolerance
  - Independent development of components
  - Extensibility of the system
The oldest SPLICE/DDS-based system is still very much alive

- **TACTICOS**: a combat management system for navy ships
  - first released ~1991
  - 100 machines, 1k processes, 10k topics
  - safety critical — survives component failures
  - complex and dynamically extensible functionality
  - all state is stored in a shared data space
  - microservices push data into SDS, convert/interpret it, display it, act upon it
Many complex subsystems on a common data bus
Enabling TACTICOS

• Thinking of data as describing the system state
  • never send any changes or events
  • a desired/actual state pattern instead of commands

• Allowing the middleware to manage that data
  • the “transient” setting of DDS

• Splitting complex objects into multiple topics
  • extending functionality then means adding topics
  • applications combine multiple topics
Tomorrow
Where is ROS2 going?

a strictly personal view

- ROS2 was really simple in the beginning, DDS rather complex
  - ROS2 simply started as a better ROS
  - DDS started as the underpinning of large-scale naval command-and-control systems

- The improvements in ROS2 allow use in more complex (robot) systems
  - these ask for addition of features to ROS2
  - features already present in DDS
  - this process is already vaguely visible
What key developments to expect?

a strictly personal view

• Designated key fields in data types
  • independent data flows within the same topic
  • important for real-time behaviour
What key developments to expect?

a strictly personal view

• Designated key fields in data types

• Data as a first-class concept
  • ROS2 today is simple publish-subscribe messaging
  • being “data centric” goes further: data has a life of its own
  • processes come and go but (some of) the data remains for later use
  • DDS calls this “transient data”
What key developments to expect?

a strictly personal view

- Designated key fields in data types
- Data as a first-class concept
- Evolving data models
  - Industrial systems have long lifespans
  - Different generations systems will have to interoperate
  - Therefore the data models need to be evolvable
The Challenge

• **ROS2 switched to DDS** to
  • solve the shortcomings of ROS1
  • leverage existing technologies

• **ROS2 is growing its feature set, including more of DDS**
  • some of those features are notoriously hard to get right
  • these features very much go beyond a mere RTPS implementation

• **ROS2 requires an open-source DDS implementation**
  • the more implementations the merrier
  • the only ones implementing these features today are commercial
Eclipse Cyclone DDS
Eclipse Cyclone DDS

- **Eclipse Cyclone DDS is a new DDS implementation**
  - ADLINK contributed initial implementation
  - reuses the proven RTPS implementation of OpenSplice
  - targets full DDS implementation for high-performance, embedded use
  - actively being developed

- **Eclipse Cyclone DDS is an active Eclipse Project**
  - Eclipse Foundation rules ensure high standards and unencumbered IP
  - rapidly growing community
  - dual licensed, EPLv2 and EDL, safe to use in any project
Cyclone DDS Performance: throughput
and open-source DDS implementations supported by ROS2

UDPv4 over loopback interface on a 2017 15” MacBook Pro (2.9GHz; macOS 10.14.4; Xcode 10.2.1)
Versions used (Git commits): Cyclone DDS: a7c7ac5; OpenSplice CE: 8a1bc07a; Fast-RTPS: a8691a40
Results are using the included performance test programs, configurations adapted to avoid fragmentation
Throughput: write as fast as possible, relying only on reader/writer resource limits and DDS internal flow-control to maximise throughput
Latency: one-way latency obtained by measuring round-trip time and dividing by two
Underwhelming Fast-RTPS reliable throughput is in combination with significant sample loss
• pausing publications for a few milliseconds after each batch of 500 samples improves rate and reduces loss
• best-effort included as it has less loss and better throughput than reliable — perhaps reliable measurement is not representative
Cyclone DDS Performance: latency and open-source DDS implementations supported by ROS2

UDPv4 over loopback interface on a 2017 15" MacBook Pro (2.9GHz; macOS 10.14.4; Xcode 10.2.1)
Versions used (Git commits): Cyclone DDS: a7c7ac5; OpenSplice CE: 8a1bc07a; Fast-RTPS: a8691a40
Results are using the included performance test programs, configurations adapted to avoid fragmentation
Throughput: write as fast as possible, relying only on reader/writer resource limits and DDS internal flow-control to maximise throughput
Latency: one-way latency obtained by measuring round-trip time and dividing by two
Underwhelming Fast-RTPS reliable throughput is in combination with significant sample loss
• pausing publications for a few milliseconds after each batch of 500 samples improves rate and reduces loss.
• best-effort included as it has less loss and better throughput than reliable — perhaps reliable measurement is not representative
Cyclone DDS Footprint

- Pure C code
- Run-time library has minimal dependencies
- Compact
  - code size can go down to ~0.5MB
  - latency test needs ~0.5MB of heap memory with some tweaking (macOS)
  - reduced footprint does reduce throughput (~0.5x) & increase latency (~5µs) a bit
  - even so, still the fastest
- Platform support
  - Supported: Linux, Windows, macOS
  - Known to work: OpenIndiana, FreeRTOS
Cyclone DDS goals for 2019

• **Functionality**
  • Transient/persistent data support
  • Improved coverage of DDS QoS
  • DDS Security
  • DDS XTypes

• **Performance, predictability**
  • Time-Sensitive Networking support
  • Reduced reliance on dynamic memory allocation
  • Zero-copy support (e.g., allow direct DMA from device into network buffer)

• **Integration**
  • Language bindings: C++, Java, Haskell, Rust, Python, Ada, ...
    (perhaps not all of them in 2019!)
  • ROS2 RMW implementation
Thank you!

eb@ilities.com / erik.boasson@adlinktech.com

Cyclone DDS: https://github.com/eclipse-cyclonedds/cyclonedds
Cyclone RMW layer PoC: https://github.com/atolab/rmw_cyclonedds