ROS COMMUNICATIONS

&

Daniel Stonier / d.stonier@gmail.com

WHAT THIS IS NOT ABOUT

Ros is quite extensive...we will not cover:

- Build System (Catkin & Rosdep)
- Software Packaging (Bloom)
- Build Farm (Open/Closed)
- Capabilities (Drivers, Algorithms, ...)
- Tools (Logging, Introspecting, Viz, ...)
- Community (Documentation, Workshops, ...)

Today we focus on the Plumbing

WHY THE PLUMBING?

- It's core
- Undergoing significant redevelopment
- Other areas are relatively static

CONTENTS

1.0

- History
- State
- Technicals
- Extensions
- Shortcomings

2.0

- Motivation
- State
- Roadmap
- Building on DDS
- Legacy Support

1.0 - HISTORY

- 2007 : Stanford (STAIR, PR1)
- 2007 : Topology : Master/Node/Param Server concepts fixed
- 2007 : Transports : reliable (TCP)/unreliable (UDP) types defined
- 2007 : Messaging Patterns : publishers, subscribers & services
- 2007 : Implementation : Posix C++
- 2008 : Moved to Willow Garage
- 2008 : Reference Robot : PR2
- 2008 : Implementation : Lisp
- 2009 : Implementation : Python
- 2009 : Implementation : ARM support for c++
- 2009 : Messaging Patterns : actions (c++/python)
- 2010 : Nodelets (IntraProcess, c++ only)
- 2010 : 1.0 Release
- 2011 : Reference Robot : Turtlebot
- 2011 : Implementation : Windows support for c++
- 2011 : Implementation : Java/Android
- 2012 : ROS 2.0 movement started
- 2014 : Transferred to OSRF

CURRENT STATE

- Stable
- No new development (since 2011)
- No new development planned
- Various community workarounds introduced (since 2011)

Mostly everyone awaiting ROS 2.0.

1.0 - TOPOLOGY

Centralised, static management of a distributed system

Master - name registration and lookup

Node - processes that perform computation.

Parameter Server - centralised data storage typically used for configuration of nodes.

Topics - where publishers and subscribers interact.

Services - where service servers and clients interact.

Names - filesystem-like naming structure for all ROS components (e.g. /stanford/robot/name)

The glue for making this work is via xmlrpc servers at each component.

1.0 - TOPOLOGY

TODO : image showing 1) master/param, 2) nodes, 3) registrations, 4) connection http://wiki.ros.org/Master

1.0 - TRANSPORTS

Conceptually organised as reliable or unreliable.

TCPROS - implements reliable behaviour, implicit shortcuts for local machine and intraprocess (c++ only) connections.

UDPROS - implements unreliable behaviour, few configuration options, incomplete support (c++ only).

Other options (e.g. shared memory) were explored, but discarded. Consensus was: 1) fast -> use c++ intraprocess connections, 2) convenience -> use python tcpros connections.

1.0 - MESSAGING

Binary wire format, but not self-describing.

Headers, modules, artifacts automatically generated at build-time from external yaml-like text files.

Support for composition of messages, but not inheritance.

Message standards derived from common usage patterns in the community (~C++/Boost). Formalised via REP (~ python PEP's).

A limited set of messaging patterns exist : pubsub, services (blocking), actions (non-blocking services with a feedback channel).

1.0 - PARAMETER SERVER

Centralised key-value data storage.

Primarily used as for static, pre-launch configuration of nodes.

1.0 - LANGUAGE IMPLEMENTATIONS

C++ - full support for transports, topics, services, actions and parameter serving. Type-safe api. Runs on posix, arm and windows. Building on windows is difficult.

Python - only missing UDPROS transport. Python 2.x and 3.x.

Java/Android - missing actions, builds with gradle/android studio.

Lisp - unsure...

Java Script - popular, but rapidly changing & unofficial support over websockets.

1.0 - HOW IT GOT HERE

Designed to be flexible, but guided by the PR2 use case.

- Single robot
- Massive computational resources on board
- Real time requirements met in a custom manner
- Excellent network connectivity
- Applications in research
- Maximum flexibility, nothing prescribed or proscribed
  - e.g., “we don’t wrap your main()”.

1.0 - SHORTCOMINGS

Fell short for multirobot, small embedded, real-time, unreliable networks & computational management.

- Centralised master
  - Autonomous robots each need their own master
- Maintenance
  - Not a large enough developer or user base
- Impractical wireless communications
  - Missing unreliable implementations
  - Unreliable transport configuration (hints) minimal.
  - No multicast support - tf trees and point clouds are expensive
- Node API only exposed at runtime
  - Requires expert knowledge to organise a system launch
  - Cannot validate/manipulate components on launching
- Cannot manage the computational graph
  - No lifecycle management
  - Cannot direct the computation
  - Cannot dynamically realign connections
- Heavy
  - too heavy for embedded (< armv6)
- Static parameter serving
  - awkward workarounds to provide dynamic parameterisations
- Not realtime

2.0 - MOTIVATION

Support emerging use cases officially:

- Teams of robots
- Small Embedded
- Non-ideal networks
- Connect to relevant communities
- Support patterns for building and structuring systems

Offload to new technologies. e.g. zeroconf, google flat buffers, DDS, Redis, WebSockets ...

2.0 - CURRENT STATE

- Decisions
  - No central master - discovery node by node
  - Open Licensing - no closed dependencies
  - ROS Messaging Types
    - Continue usage of external ROS message type formats (it worked)
    - Check cost/apply converters to satisfy any middleware requirements
  - Implement on DDS
    - Offload expensive middleware development
    - Connect with major players in similar industries
  - Backwards compatibility - support ways of talking with ROS 1.0
- Development
  - Design - messaging/library/stack layout
  - Simple examples - validated working pubsub/services, ...
  - Unit testing - across linux, arm, windows.

2.0 - ROADMAP

It'll be ready when it's ready.

Nearest Target

Working prototype with fundamental components available for human consumption ~end of summer 2015.

Quite likely a couple of years before fully featured.

2.0 - DDS

It’s a standard

Several implementations - and these talk to each other, low risk.

Big users - connect to NASA, US Military.

Wireless support - very high level, driven by space applications.

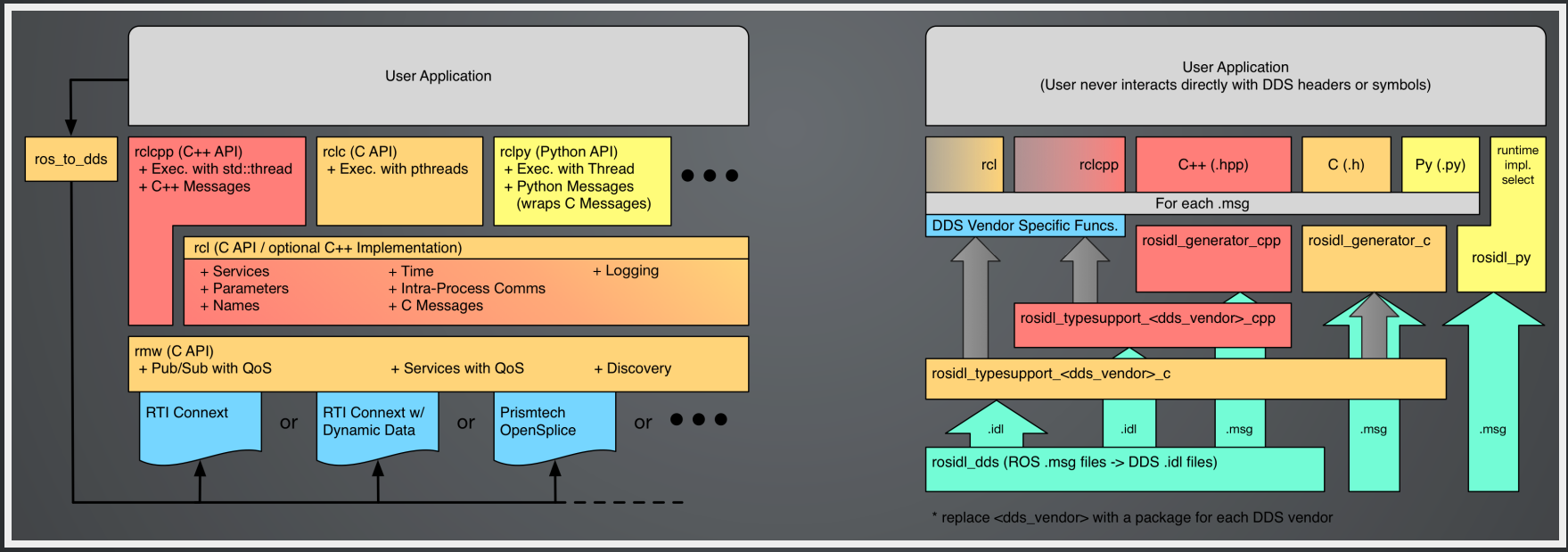
Very Flexible/Detailed - can meet all of ROS' MW needs.

Current Vendor Choice - OpenSplice.

Current Design Goal - hide complexity, but leave it accessible for power users.

2.0 - ROS CLIENT LIBRARY STACK

User interacts with ros client libraries, which in turn interacts with DDS implementations.



2.0 - DISCOVERY & SEGMENTATION

- Discovery
  - Nodes use Multicast/Broadcast UDP to find each other
  - Implemented with the SDP/SPDP/SEDP protocols from DDS.
- Domains
  - DDS Concept
  - Connect to each other on separate multicast domains.
  - Replaces the old concept of a ROS Master
- Partitions
  - DDS Concept
  - Abstract partitioning for nodes on a single domain.
  - Like partitions on a hard drive, but more flexible.

2.0 - TRANSPORTS

Harness QoS capabilities from DDS.
Establish common profiles for robotics.

- Focus on reliable and unreliable
- Build up profiles for connection configurations and apply it to configure all unreliable connections on a system.
  - This is important for settings that can be configured by a system integrator
  - e.g. Standard Wireless Lab Profile
  - e.g. Outdoors Wireless.
- Differentiate for transport hints that must be applied programmatically
  - This is important for settings that effect how you write code
- Save backlogs to send when a wireless connection gets reestablished.
- Make use of multicast when available (when DDS layer supports it).

2.0 - NAMES

Dynamic Remappings

- Runtime Renaming
- Aliasing
  - e.g. alias /foo to /foo_compressed with lz compression

Useful for managing and building up complex systems.

2.0 - LANGUAGES

C Api - use as a basis for client library bindings (N/A in ROS 1.0).

C++/Python - worked well for ROS 1.0, no reason to change it.

2.0 - MANAGING A RUNTIME SYSTEM

Managed Nodes - component style nodes with lifecycle patterns (~OPROS, OROCOS). Supports new use cases, but doesn't replace freeform ROS node patterns.

Graph Management - explore ways to manage the computational and topological graphs. Interesting target, but still in discussion and a long ways off. Design thoughts going into development of the node API.

LEGACY SUPPORT

Goal is to have a smooth transition - the fundamental ROS API (ROS messages) will remain the same while the client libraries will have means of connecting/talking to each other.

Q&A

DANIEL STONIER / D.STONIER@GMAIL.COM