Programming mobile robots with ROS2 and the RCLAda Ada Client Library

Alejandro R. Mosteo

2021-jun-07

Ada-Europe International Conference on Reliable Software Technologies

CONTENTS

- About
- ROS2 introduction
- ROS2 build process
- Ada package creation
- RCLAda big picture
- RCLAda API highlights
- Robotics specifics

First of all…

Install ROS2 + Webots

(https://github.com/ada-ros/tutorial-aeic21/blob/master/Exercises.md#0-setup-of-the-working-environment)
or download the Virtual Machine
(https://github.com/ada-ros/tutorial-aeic21/releases)

Robotics, Perception and Real-Time group - RoPeRT

University of Zaragoza

Engineering Research Institute of Aragon

RoPeRT

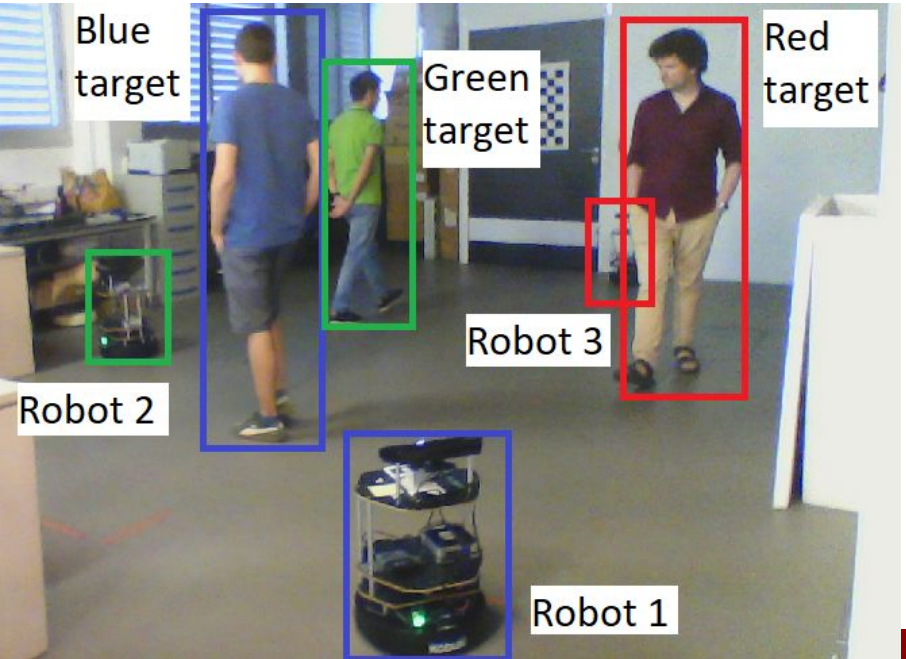
Optimal distributed coordination

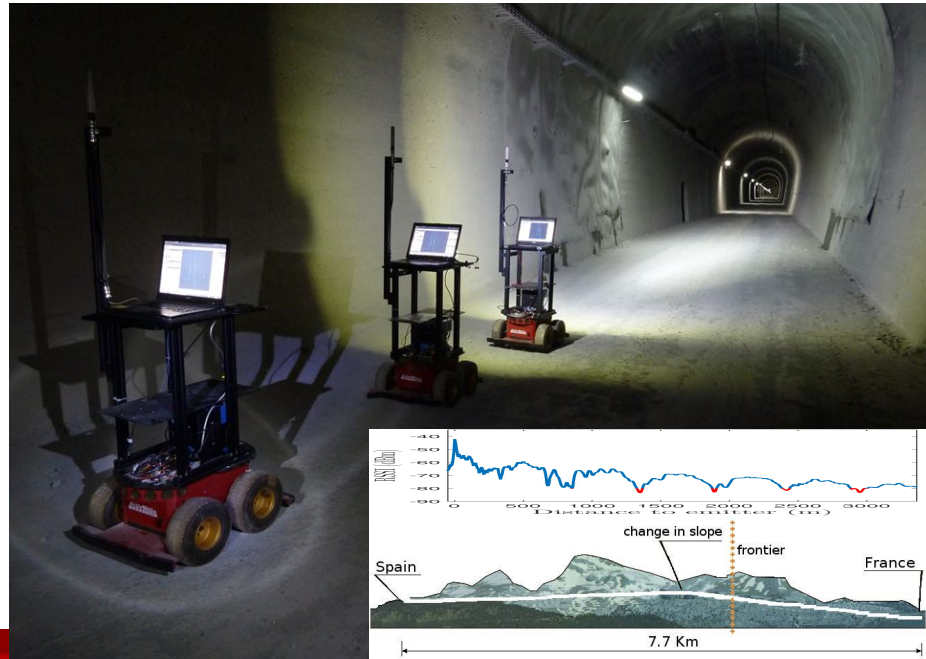
Real-time multi-hop communication

Underground drone reconnaissance

ROBOTICS
UNIVERSIDAD DE ZARAGOZA

http://robots.unizar.es/





ROBOTICS
UNIVERSIDAD DE ZARAGOZA

http://robots.unizar.es/

In a nutshell

What do we want: to “move” robots

How we will do it: with ROS2

Supported languages: C++, Python

Unsupported favorite language: Ada

Robotics needs

Robotics

Academia ⟷ Industry

Fast Prototyping

Long runs

Correctness

Certifiability

The Seattle Times

FAA orders Boeing 787 safety fix: Reboot power once in a while

FAA orders Boeing 787 safety fix: Reboot power once in a while. Originally published December 1, 2016 at 11:05 am Updated December 2, 2016 at 12:35 am.

Dec 1, 2016

High-integrity subsets

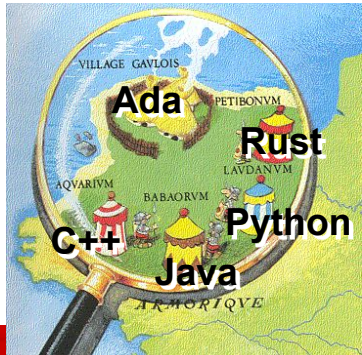
Clamp down on general-purpose (C/C++)

Language for High-Integrity Systems

Design objective (Ada)

Ada history

- 1975: Working group DoD / UK MoD
  - STRAWMAN first discussions
- 1978: STEELMAN requirements
  - **Embedded, reliability, maintainability, efficiency requirements**
  - No suitable existing candidate
- 1979: Green proposal by Jean Ichbiah of Honeywell Bull
  - Renamed to Ada
- 1983: standard ANSI /MIL-STD-1815A (Ada 83)
- 1991-1997: DoD mandate years
  - From 450 to 37 languages by 1998
- Today: niche in many critical industries
  - **Aerospace, railways, automotive, …**



Evolution

- Ada Rapporteur Group
  - Receives suggestions, requests, comments
  - Prioritizes “not” doable in current Ada
- Ada Reference Manual (ARM)
  - AARM: Annotated ARM for experts, compiler writers
  - All are ISO standards
- **Ada Conformity Assessment Test Suite (ACATS)**

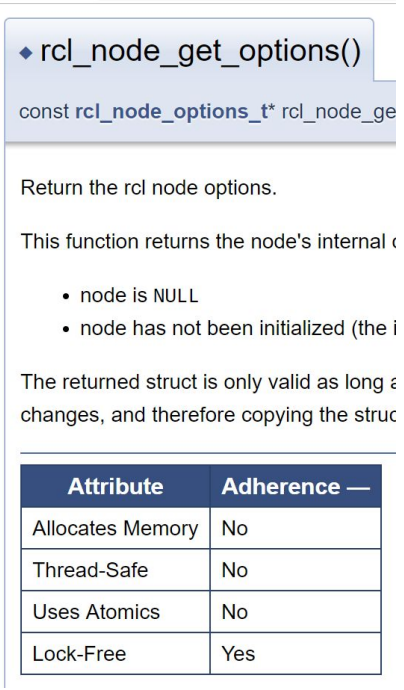
Source: https://www.adacore.com/about-ada

Feature	Ada 83	Ada 95	Ada 2005	Ada 2012	Ada 202X
Packages	✓	✓	✓	✓	✓
Generics	✓	✓	✓	✓	✓
Derived ADTs	✓	✓	✓	✓	✓
Object orientation (tagged types)		✓	✓	✓	✓
Multiple inheritance (abstract interfaces)			✓	✓	✓
Design by Contract				✓	✓
Numeric types (fixed, floating, decimal, custom)	✓	✓	✓	✓	✓
Tasks	✓	✓	✓	✓	✓
Monitors		✓	✓	✓	✓
Real-time systems annex		✓	✓	✓	✓
Ravenscar profile			✓	✓	✓
Multiprocessor affinities, Multiprocessor Ravenscar				✓	✓
Parallel constructs (blocks, loops)					✓

Why ROS2 + Ada

ROS2

- More focus on
  - Embedded, real-time, industrial use
- Traditional strong points of Ada
  - Portable & precise memory layouts, at bit level
  - Annex C: systems programming
    - Interrupts, atomics, volatiles
  - Annex D: (hard) real time
    - Priorities, schedulers, monotonic clock, tasking profiles
  - Portability (ISO/IEC 8652:2012)
  - Certification
    - DO-178B/C (aero), EN 50128 (railway), ECSS-E-ST-40C/ECSS-Q-ST-80C (space), IEC 61508 (industrial automation), ISO 26262 (automotive)



◆ rcl_node_get_options()

const **rcl_node_options_t*** rcl_node_ge

Return the rcl node options.

This function returns the node's internal

- node is `NULL`
- node has not been initialized (the i

The returned struct is only valid as long a changes, and therefore copying the struc

Attribute	Adherence —
Allocates Memory	No
Thread-Safe	No
Uses Atomics	No
Lock-Free	Yes

ROS2 introduction

What is ROS

- Robot Operating System
  - But not really

ROS

Open Source Robotics Foundation

*"The Robot Operating System (ROS) is a set of **software libraries** and **tools** that help you build robot applications. From **drivers** to state-of-the-art **algorithms**, and with powerful developer tools, ROS has what you need for your next robotics project. And it's all **open source**."*

Main OSRF project (10 years now)

ROS2 Main Parts

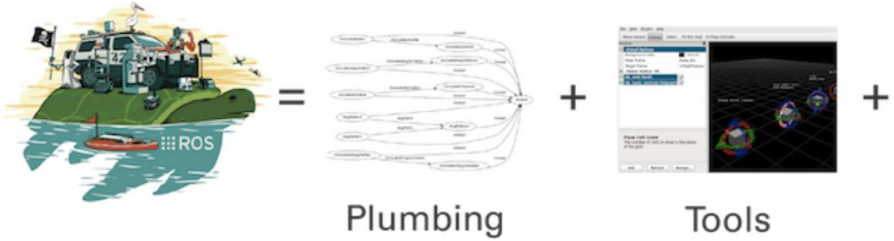
ROS

- Collection of Debian/Ubuntu packages ready to use
  - Sensor/platform/actuator drivers
  - High-level algorithms
- Build system (∈ “tools”)
  - Heterogeneous language environment
  - Nodes isolated as processes/threads
- Intercommunication facilities (“plumbing”)
  - Message publishing
  - Remote Procedure Calls
  - Actions

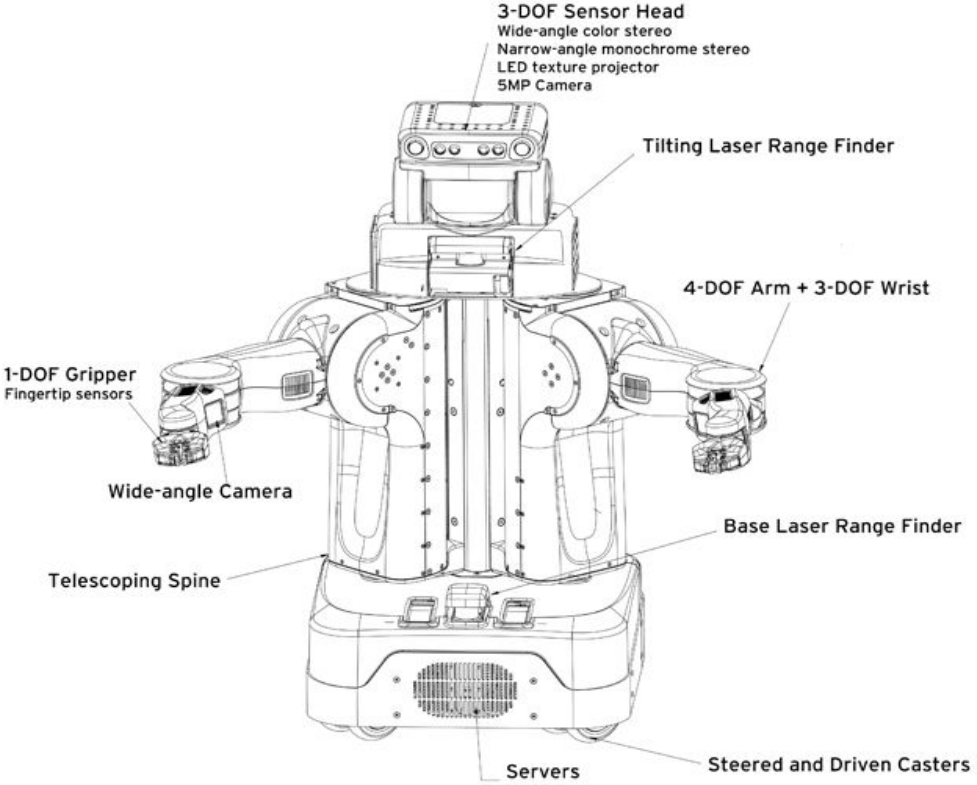
Colcon

CMake
Cross-platform Make

python



PR2: the kick-off robot



- ROS support widespread
  - Expected in research/academia contexts
  - Either by 1st or 3rd parties

Summit XL 4WD Autonomous Robot
Product Code : RB-Rtk-04
1 Review(s)
USD $15,000.00

Svenzva Robotics Revel 6 DoF Robotic Arm
Product Code : RB-Svz-01
USD $6,495.00

ROSbot 2.0 w/ LIDAR & RGBD Robotic Platform
Product Code : RB-Rco-07
USD $1,899.00

TeraRanger Duo ToF Rangerfinder with Sonar Sensor
Product Code : RB-Ter-07
Excl. Tax: €185.00
Incl. Tax: €223.85

2.3 3D Sensors (range finders & RGB-D cameras)

- Argos3D P100 ToF camera
- Basler ToF ES camera
- DUO3D™ stereo camera
- Ensenso stereo cameras
- Forecast 3D Laser with SICK LIDAR
- SceneScan and SP1 by Nerian Vision Technologies
- OpenNI driver for Kinect and PrimeSense 3D sensors
- Trifo Ironsides
- PMD Camcube 3.0
- IFM O3M250 ToF camera
- Intel® RealSense™ F200/VF0800
- Intel® RealSense™
- Roboception rc_visard stereo camera
- Terabee 3D ToF camera
- Orbbec Astra
- SICK MRS1xxx lasers
- SICK MRS6xxx lasers
- SICK LD-MRS laser (identical to IBEO LUX) or csiro-asl/sick_ldmrs
- Sentis ToF M100 camera
- Mesa Imaging SwissRanger devices (3000/4000/4500)
- Velodyne HDL-64E 3D LIDAR
- Livox 3D LiDAR

2.2 2D range finders

- NaviRadar
- HLS-LFCD LDS
- Hokuyo Scanning range finder
- PepperI+Fuchs R2000 laser
- Leuze rotoScan laser rangefinder driver (ROD-4, RS4)
- RPLIDAR 360 laser scanner Driver(python)
- RPLIDAR A1/2 laser(c++)
- SICK LMS1xx lasers or LMS1xx
- SICK LMS2xx lasers or sicktoolbox_wrapper
- SICK S3000 laser
- SICK S300 Professional
- SICK TiMxxx lasers or sick_tim
- SICK Safety Scanners (microScan3)
- TeraRanger Multiflex
- TeraRanger Hub & Tower
- TeraRanger Hub Evo & Tower Evo
- TeraRanger Evo 64px ToF range finder
- Neato XV-11 Laser Driver

Why ROS


GAZEBO


pointcloudlibrary


ROS


ROS
industrial

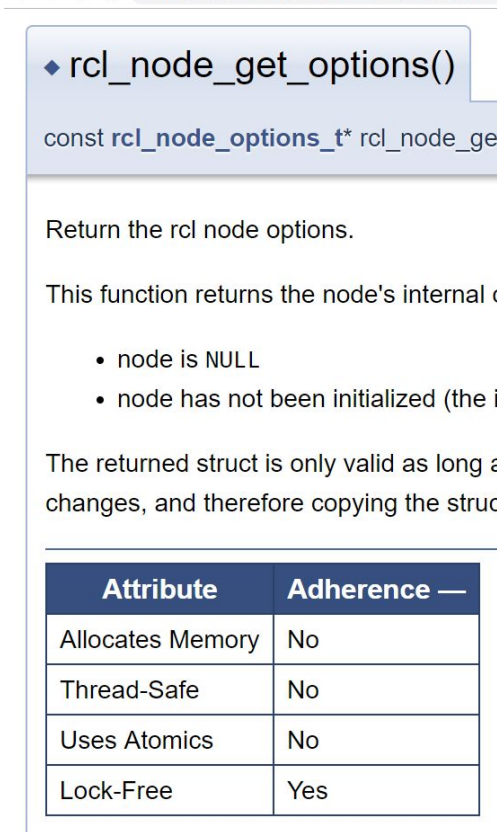

MoveIt!


OpenCV


ROSin

ROS2 vs ROS

- More emphasis on
  - Embedded (microcontrollers)
    - ROS has been mostly a linux affair
  - Real-time
    - Coming from firm/soft real-time
  - Actual readiness for industrial settings
    - Long lived processes vs short experiments
    - Standard DDS for data transport
      - Swappable implementation
- Traditional strongholds of Ada



◆ rcl_node_get_options()

const **rcl_node_options_t*** rcl_node_ge

Return the rcl node options.

This function returns the node's internal

- node is `NULL`
- node has not been initialized (the i

The returned struct is only valid as long a changes, and therefore copying the struc

Attribute	Adherence —
Allocates Memory	No
Thread-Safe	No
Uses Atomics	No
Lock-Free	Yes

ROS2 hot topics

- Working Groups on
  - Real-time
  - Safety-critical
    - Already seen interest in SPARK

ROS 2 and Real-time

Next Generation ROS

Dejan_Pangercic 9d

In one of the previous ROS 2 TSC meeting it was suggested that we form a Working Group in which we will try to analyse the current state of ROS 2 and make it real-time.

To this date we have the following articles about real-time in ROS 2:

1. Original article by Jackie: https://design.ros2.org/articles/realtime_background.html 11
2. ROS 2 ported on some RTOS (https://www.esol.com/embedded/ros.html 9, http://blackberry.qnx.com/en/articles/what-adas-market-needs-now 2)
3. Apex.AI article about porting ROS 1 applications to ROS 2 applications: https://www.apex.ai/blog/porting-algorithms-from-ros-1-to-ros-2 12
4. Bosch proposing how to make Callback-group-level Executor real-time https://vimeo.com/292707644 7

Since real-time is not something that can start and stop within the ROS 2 “borders”, we would like to propose to analyse an entire stack, from the hardware platform to the applications written with ROS 2.

Safety-critical WG

Next Generation ROS

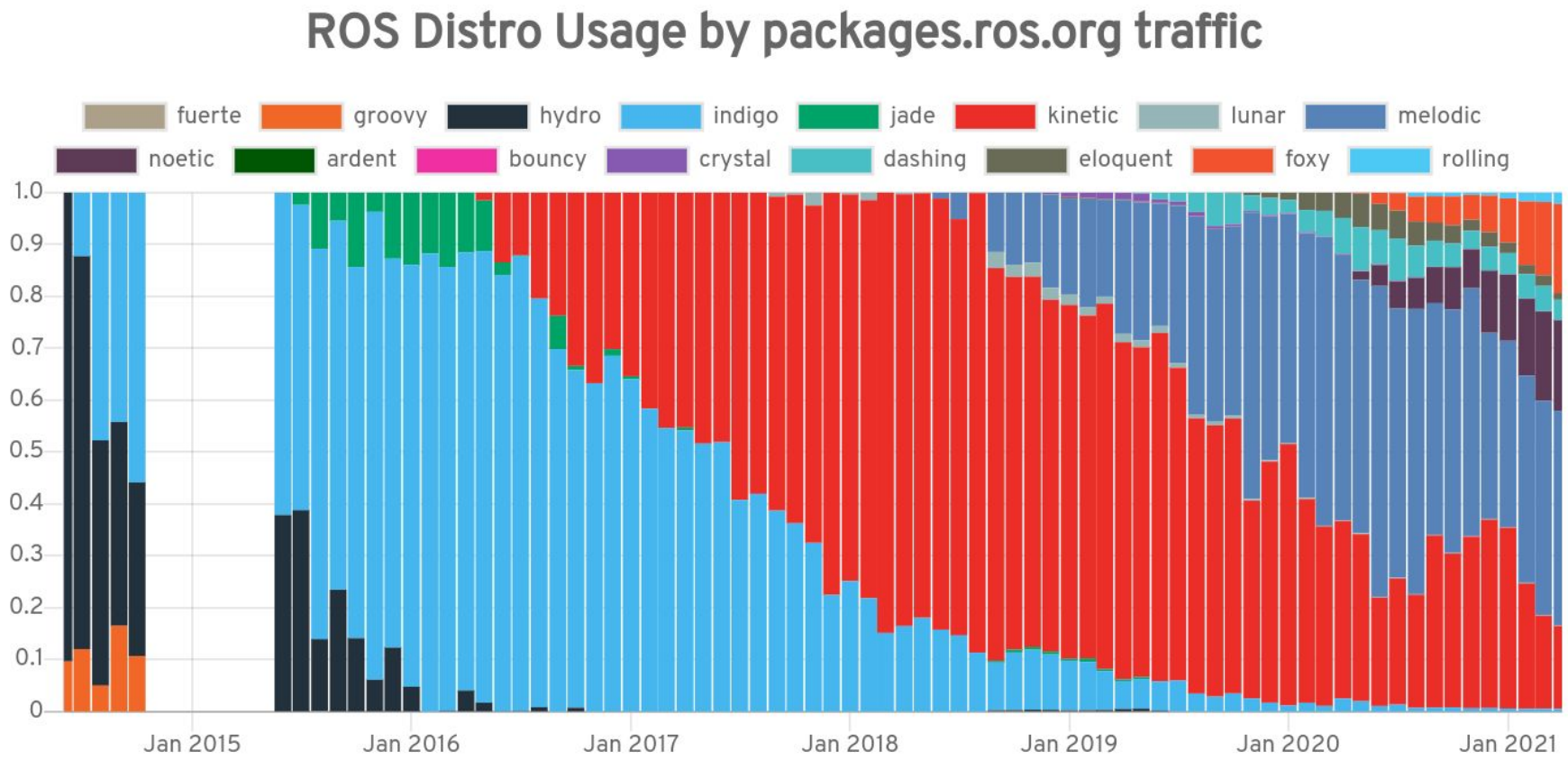
gbiggs 4d

Some time ago I was asked to lead a working group looking at the use of ROS 2 in safety-critical systems. These are systems that may potentially cause harm to people or the environment, and I think that most of us agree that a large number of robot applications fall into this category.

The working group will look at topics including:

- Documenting how to use ROS 2 in a safety-critical application
- Use of tools to support the above
- Additional processes, tools and methods needed for building a safety-critical robot that are not currently covered by something in ROS but could be
- How to make the client libraries usable in a safety-critical system, and work on safety-focused client libraries (for example, a SPARK client library)
- Cross-over issues with the QA and real-time working groups for infrastructure, tooling and methods
- Cross-over issues with the navigation and manipulation working groups for sample applications
- Anything else safety-related someone brings along

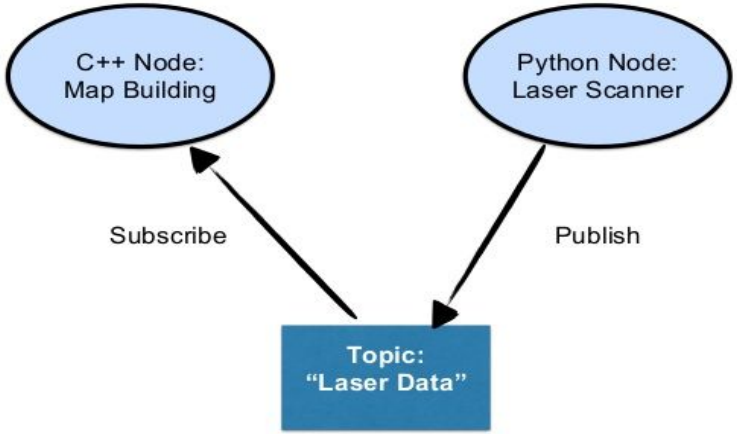
ROS version adoption

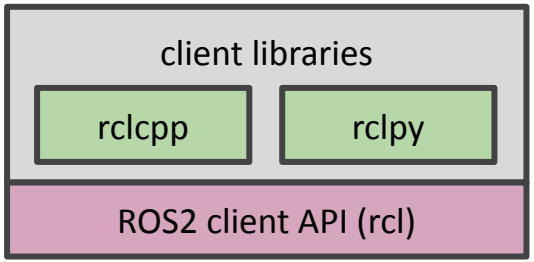


https://metrics.ros.org/packages_rosdistro.html

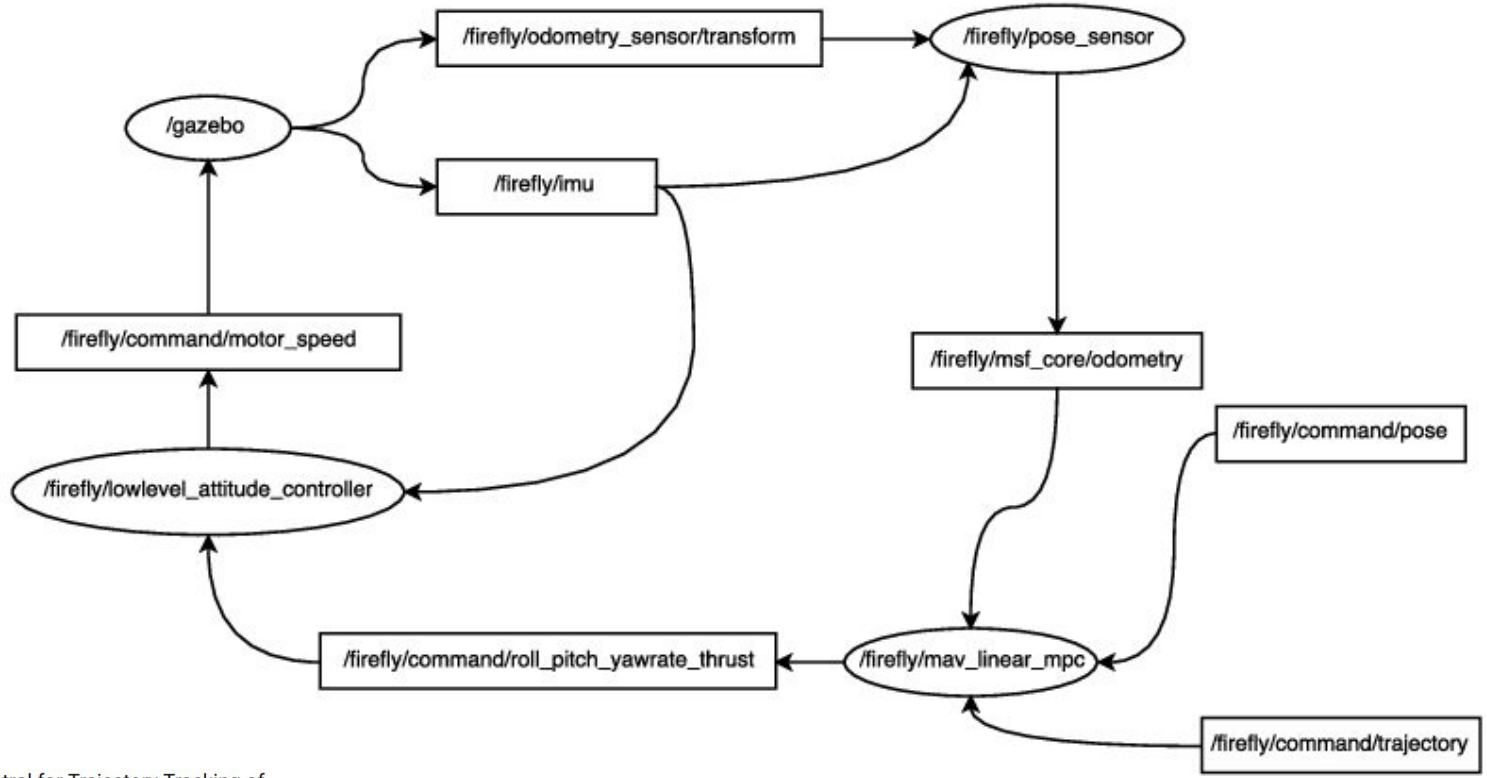
Down to business

- ROS2 “program”
  - Set of nodes (processes)
    - Found in ROS packages
  - Interconnected (DDS) by
    - Topics
      - Publish, Subscribe
    - Services
      - Request + Response
  - Supported languages
    - C++, Python (Client APIs)
    - C (low-level API)





Nodes + Topics



Model Predictive Control for Trajectory Tracking of Unmanned Aerial Vehicles Using Robot Operating System

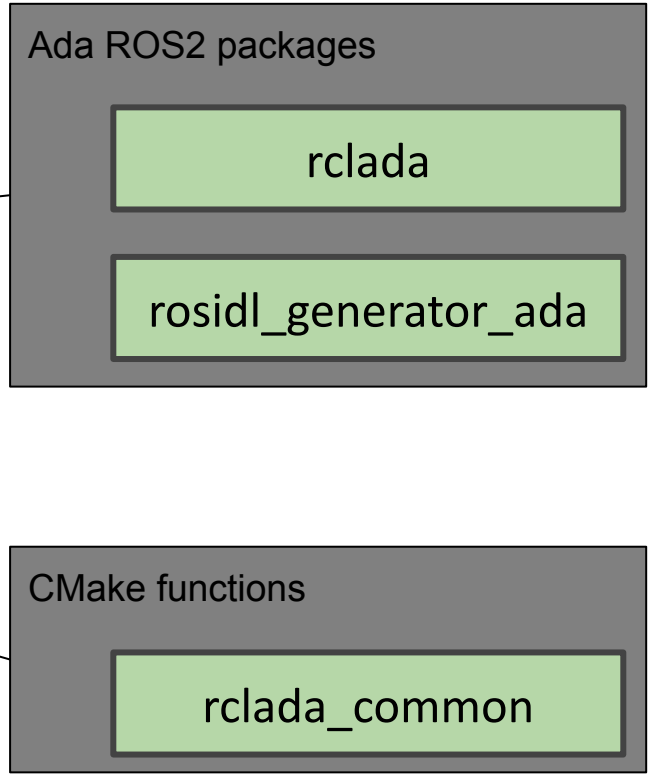
May 2017 · Studies in Computational Intelligence
DOI: 10.1007/978-3-319-54927-9_1
In book: Robot Operating System (ROS) The Complete Reference, Volume 2 · Publisher: Springer
Editors: Anis Koubaa

Mina Samir Kamel · Thomas Stastny · Kostas Alexis · Roland Siegwart

ROS2 build process

Workflow (developer)

1. `$ ros2 pkg create`
   write code
2. `$ colcon build`
   compile code
3. `$ ros2 launch`
   execute code



ENVIRONMENT SETUP

- AFTER a successful `colcon build`
  - Build incrementally in case of problems

    `--packages-select`

    `--packages-up-to`

- **setup.bash** recursively loads environment layers
  - ROS2 ← RCLAda ← Ada projects

- **local_setup.bash** loads only one layer

Ada [ROS2] package creation

Structure

RCLAda

colcon build

ament_cmake

Ada-specific
CMake functions

Support for ament_cmake

rclada_common

- **ada_begin_package**()
- **ada_end_package**()

  Needed to propagate Ada information through ROS2 packages

- **ada_add_executables**(TARGET SRCDIR DSTDIR EXECUTABLES...)

  Declares an Ada executable to be built and exported (tab completion)
- **ada_add_library**(TARGET SRCDIR GPRFILE)

  Declares an Ada library project to be built and exported to other Ada packages

- **ada_import_interfaces**(PKG_NAME...)

  Generates bindings to the typesupport handle functions

  Generates Ada types for messages

- **ada_generate_binding**(TARGET SRCDIR GPRFILE INCLUDE HEADERS...)

  Invokes the binding generator in the context of an Ada project

```
// CMakeLists.txt
cmake_minimum_required(VERSION 3.5)

project(my_ada_ros2_project VERSION 0.1.0)

find_package(rclada_common REQUIRED)

ada_begin_package()

find_package(rclada REQUIRED)
find_package(rosidl_generator_ada REQUIRED)

ada_import_interfaces(PKG_NAME)

ada_add_executables(
       my_ada_project          # CMake target
       ${PROJECT_SOURCE_DIR}   # Path to *.gpr
       bin                     # Path to binaries
       my_ada_main)            # Binaries (nodes)

ada_end_package()
```

Standard CMake project declaration

Import Ada-specific CMake functions

Import Ada environment

Import RCLAda GPR projects

- RCLAda: Nodes, Topics, etc
- ROSIDL_Ada: Messages

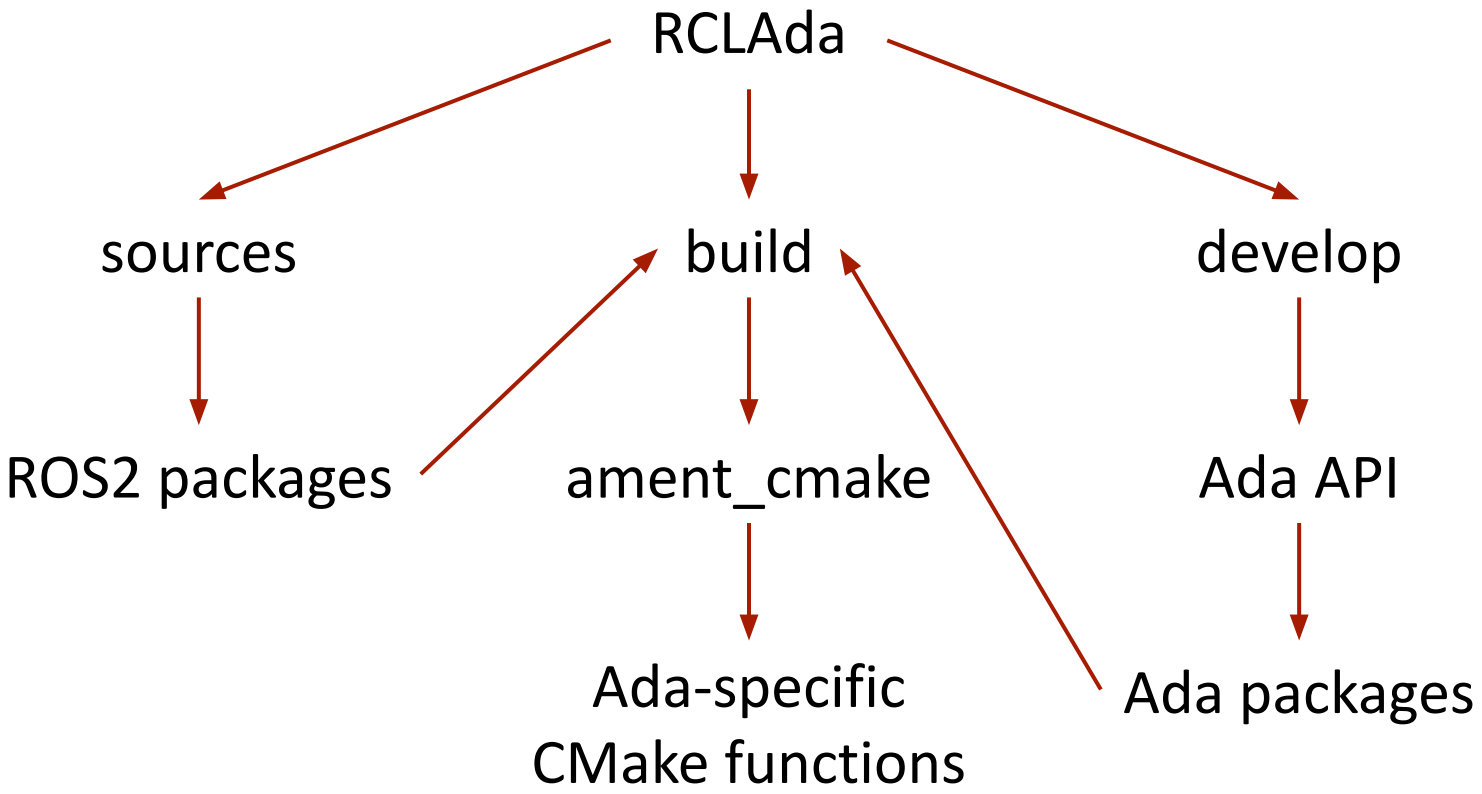
Import message definitions

Declare our Ada binary/library project

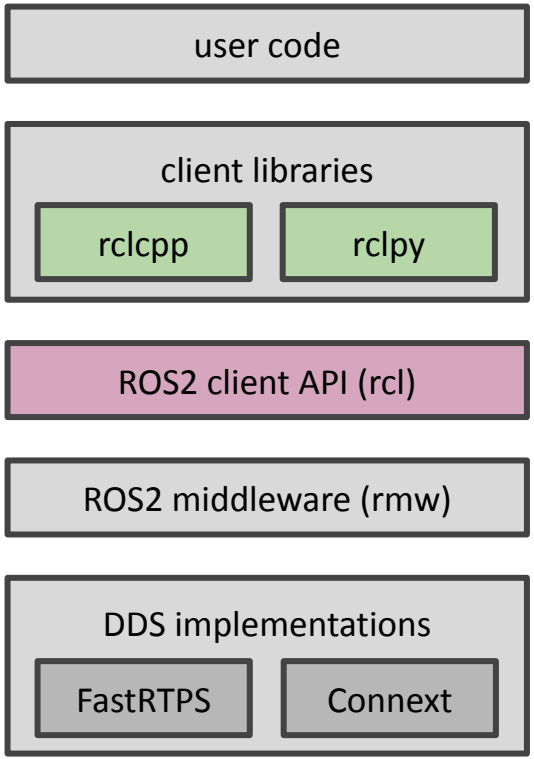
Export our additions to downstream

RCLAda big picture

STRUCTURE

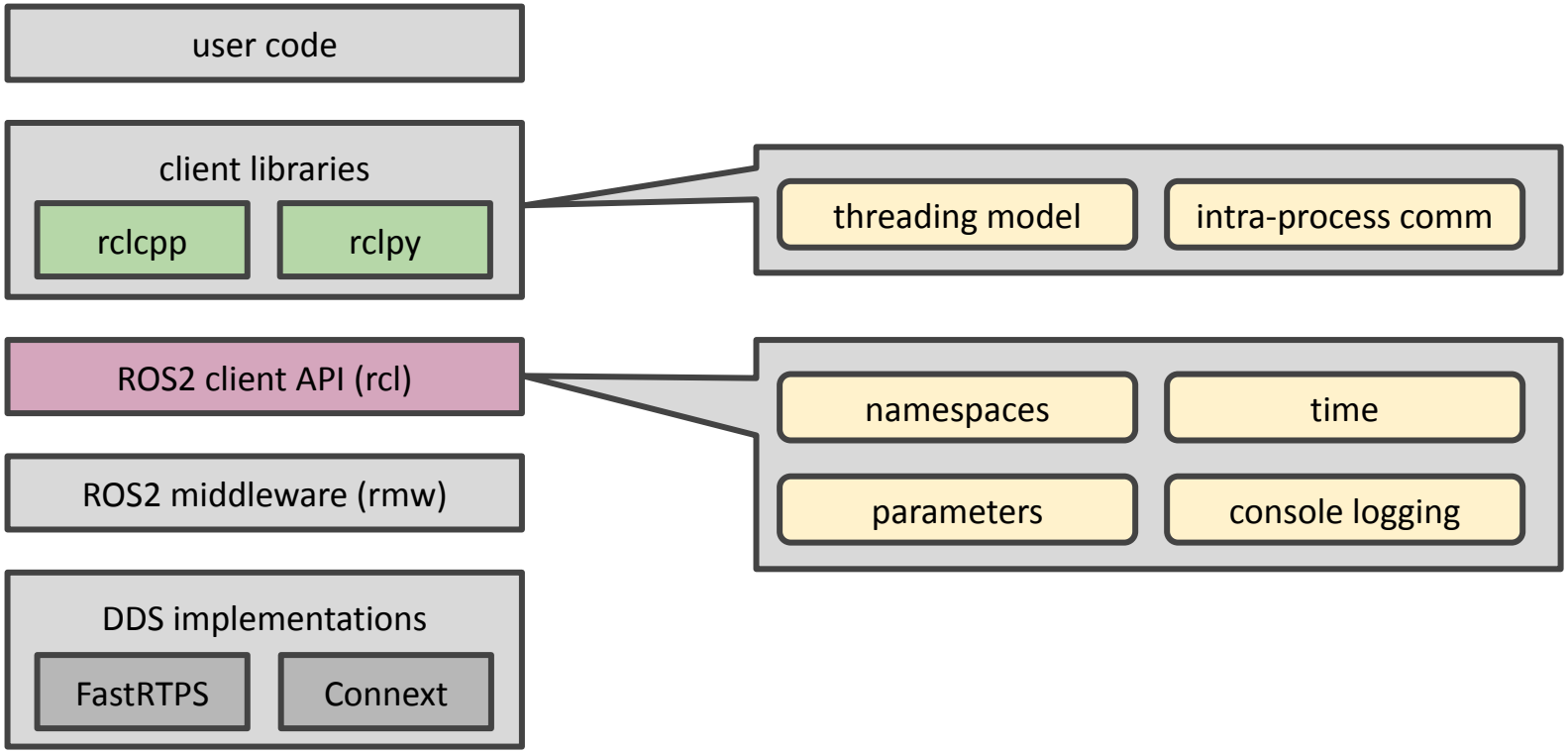

RCLAda
sources
build
develop
ROS2 packages
ament_cmake
Ada API
Ada-specific
CMake functions
Ada packages

ROS2 client support

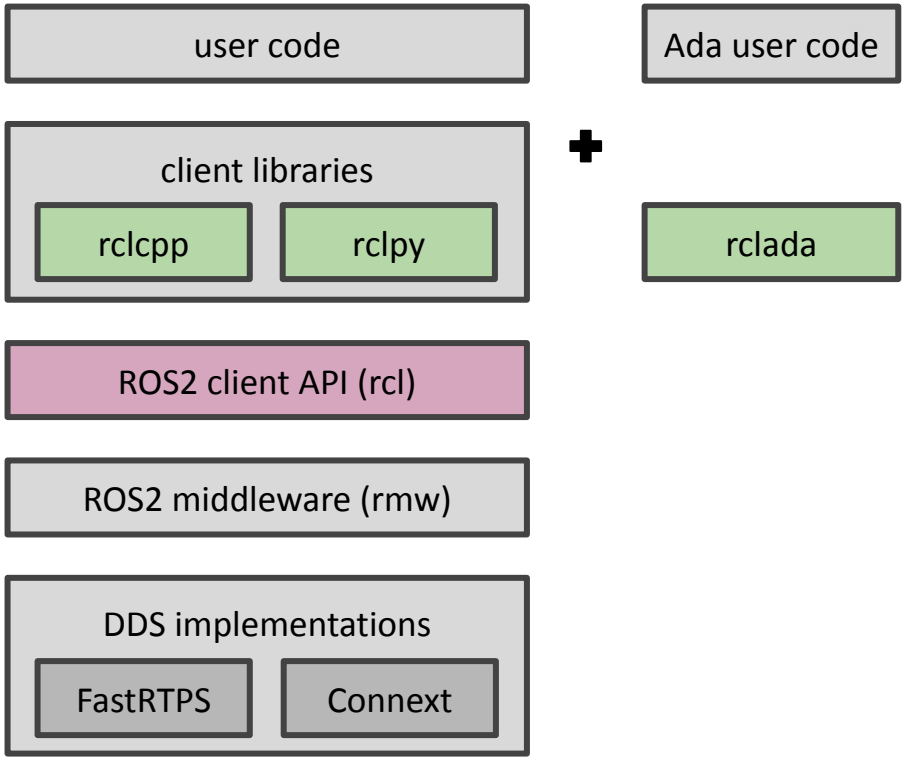


Original diagram by Deanna Hood, William Woodall: https://github.com/ros2/ros2/wiki/ROS-2-Client-Libraries

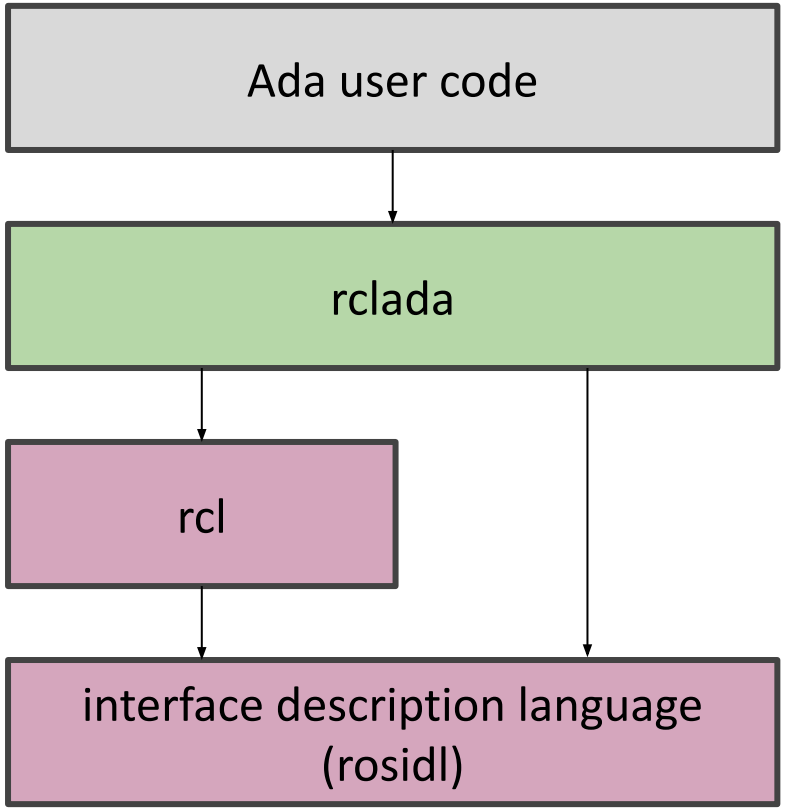
ROS2 client support



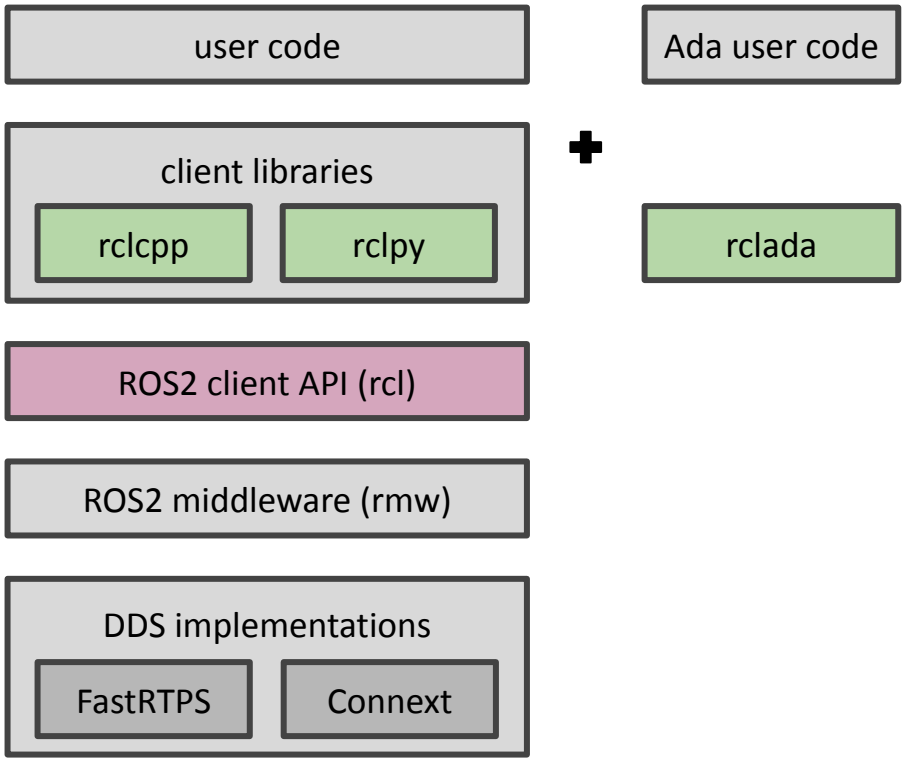
ROS2 client support

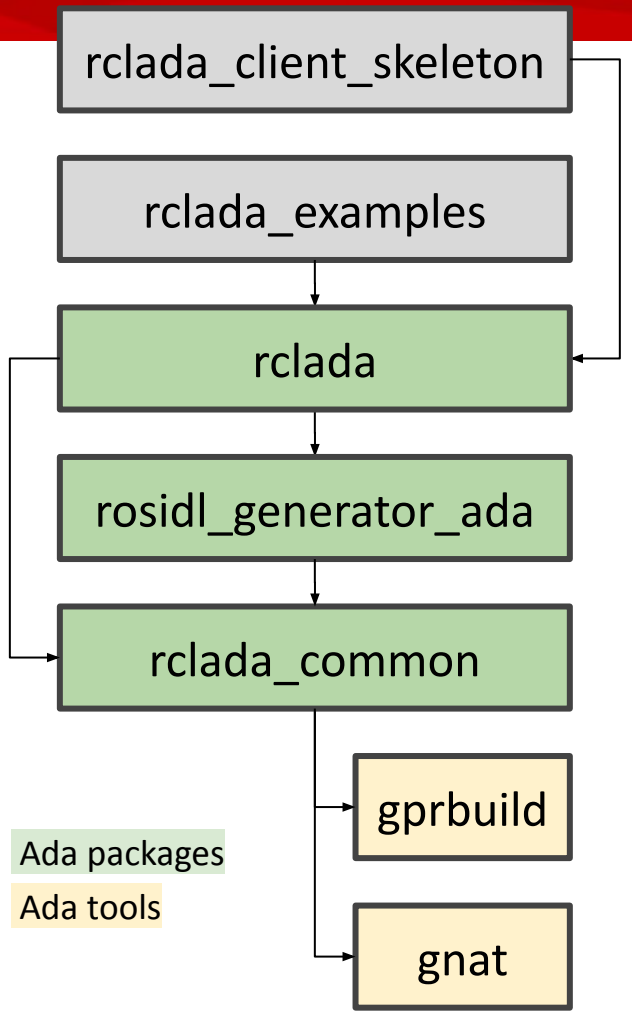


ROS2 client support II

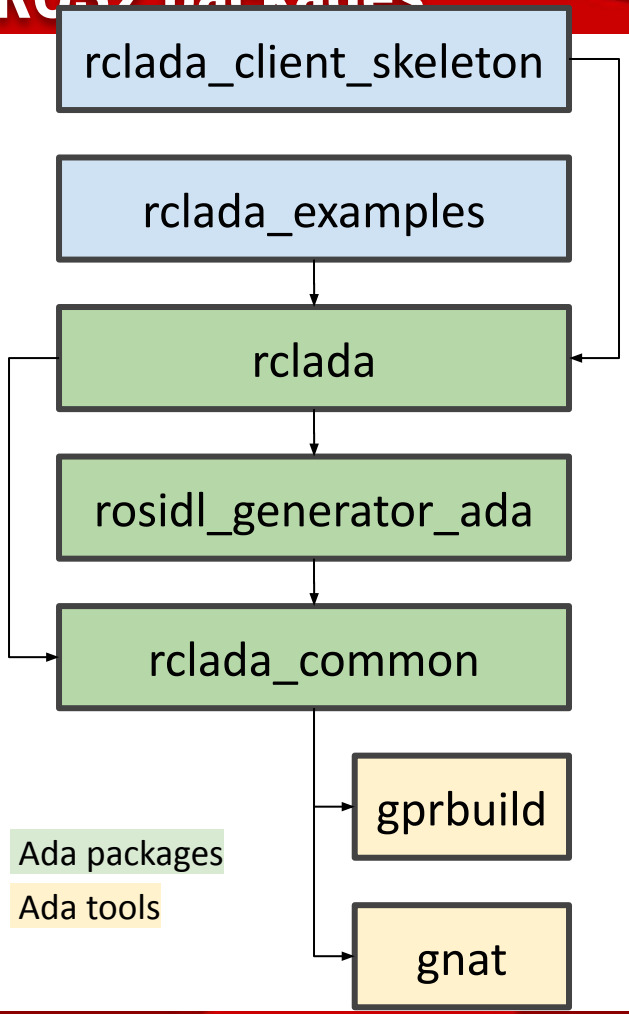


ROS2 client support





Ada ROS2 packages



Empty quickstart package

talker, listener, add_two_ints, …

Client API (nodes, topics, services, …) & self-tests

Message support

CMake functions

GNAT build system

Ada compiler (gnat-gcc) and tools

Writing Ada bindings

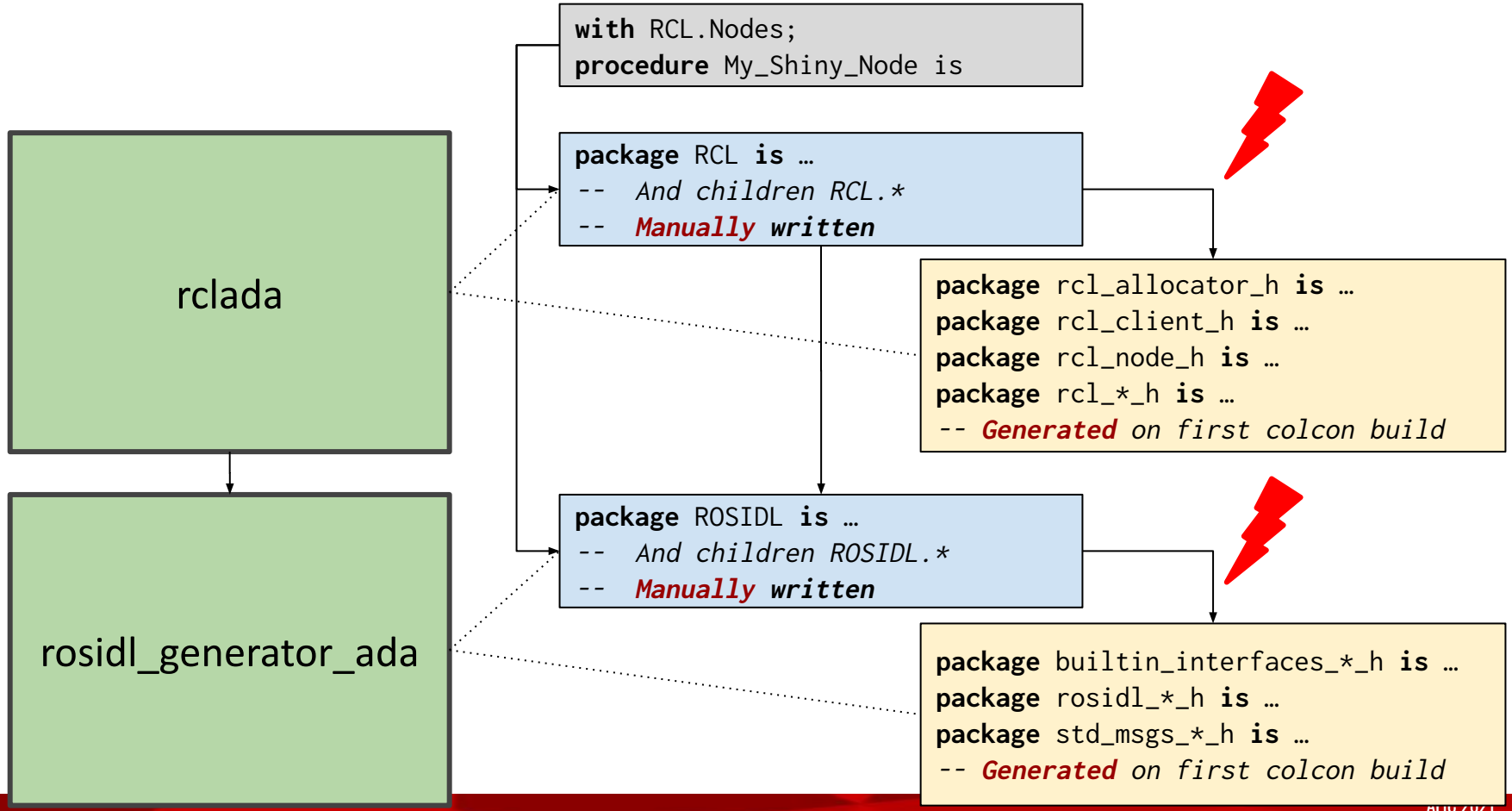
- Writing bindings:
  - Manual writing
    - ☺ No need to be exhaustive
    - ☺ High quality (thick binding)
    - ☹ More effort
    - ☹ May become de-sync'd
  - Automated generation
    - ☺ "Less" work
    - ☺ Completeness
    - ☺ Assured consistency
    - ☹ Lower quality (thin binding)
    - ☹ Might not compile
- Ada/GNAT support:
  - Annex B: interface to other languages
    - C/C++, Fortran, Cobol
  - **`gcc -fdump-ada-spec file.h`**

```
/* C prototype */
int initialize(options_t *opts,
               char *argv[]);
```

```
-- Ada automatic binding
function Initialize
   (opts : access Options_T;
    argv : System.Address)
    return Interfaces.C.int
    with Import, Convention => C;
```

```
-- Ada manual binding
type Arg_Array is
  array (Natural range <>) of aliased
  Interfaces.C.Strings.Chars_Ptr
  with Convention => C;

function Initialize
   (opts : in out Options_T;
    argv :        Arg_Array)
    return Interfaces.C.int
    with Import, Convention => C;
```

RCLAda: leverage *colcon* for best of both worlds

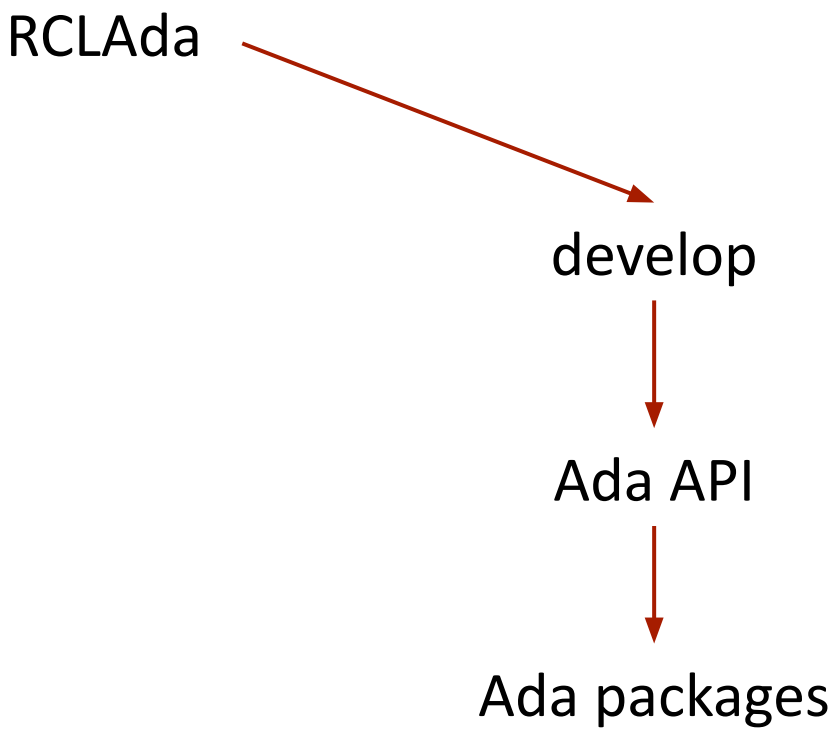


RCLAda Distinguishing Features

- No heap allocations (in RCL Ada client layer)
  - Guaranteed by language restrictions & libraries
- Relies on automatic low-level binding
  - Immediate detection of mismatches on ROS2 API changes
- Language ingrained in safety/high-integrity culture
  - Enforced safe program initialization / task completion
  - Strong static type system (incl. numerics) (plus predicates)
  - A convenient path to formal verification with SPARK (same toolchain)
  - Available toolchains for certification
- Strong backwards & cross-platform compatibility
  - Simple interoperability with C/C++

RCLAda API highlights

Structure



Completi

rclada

- Main features:
  - `RCL.Node` : Complete
  - `RCL.Publisher` : Complete
  - `RCL.Subscription` : Complete
  - `RCL.Client` : Complete
  - `RCL.Service` : Complete
- Support:
  - `RCL.Allocators` : Complete
  - `RCL.Calendar` : Complete
  - `RCL.Executors` : Complete
  - `RCL.Graph` : Complete
  - `RCL.Options` : Partial
  - `RCL.Timer` : Complete
  - `RCL.Wait` : Complete

rosidl_generator_ada

- Messages:
  - `ROSIDL.Dynamic`: Complete
  - `ROSIDL.Typesupport`: Complete
- Dynamic access (through introspection):
  - Typesupport: Complete
  - Simple types: Complete
  - Nested types: Complete
  - Array types: Complete
  - Matrix types: Complete
- Static access (through generated types):
  - Typesupport: Pending
  - Simple types: Pending
  - Nested types: Pending
  - Array types: Pending
  - Matrix types: Pending

ROS2 IDL messages

```
# LaserScan.msg: Single scan from a planar laser range-finder

std_msgs/Header header  # timestamp in header is the acquisition time-axis

float32 angle_min       # start angle of the scan [rad]
float32 angle_max       # end angle of the scan [rad]
float32 angle_increment # angular distance between measurements [rad]

float32 time_increment  # time between measurements [seconds]

float32 scan_time       # time between scans [seconds]

float32 range_min       # minimum range value [m]
float32 range_max       # maximum range value [m]

float32[] ranges        # range data [m]
float32[] intensities   # intensity data [device-specific units].  If your
                        # device does not provide intensities, please leave
                        # the array empty.
```

```
declare
   Support : ROSIDL.Typesupport.Message_Support :=
             ROSIDL.Typesupport.Get_Message_Support
               (Pkg_Name, Msg_Type);
   Msg : ROSIDL.Dynamic.Message := Init (Support);
begin
   Msg ("valid").As_Bool := True;
   Msg ("X").As_Float32  := 1.0;
   --  Individual values

   Msg ("Values").As_Array (42).As_Int8 := 0;
   --  Array indexing

   Msg ("Image").As_Matrix ((100, 50, 1)).As_Int8 := 0;
   --  Matrix indexing
end;
```

rosidl_generator_ada

Obtain message type

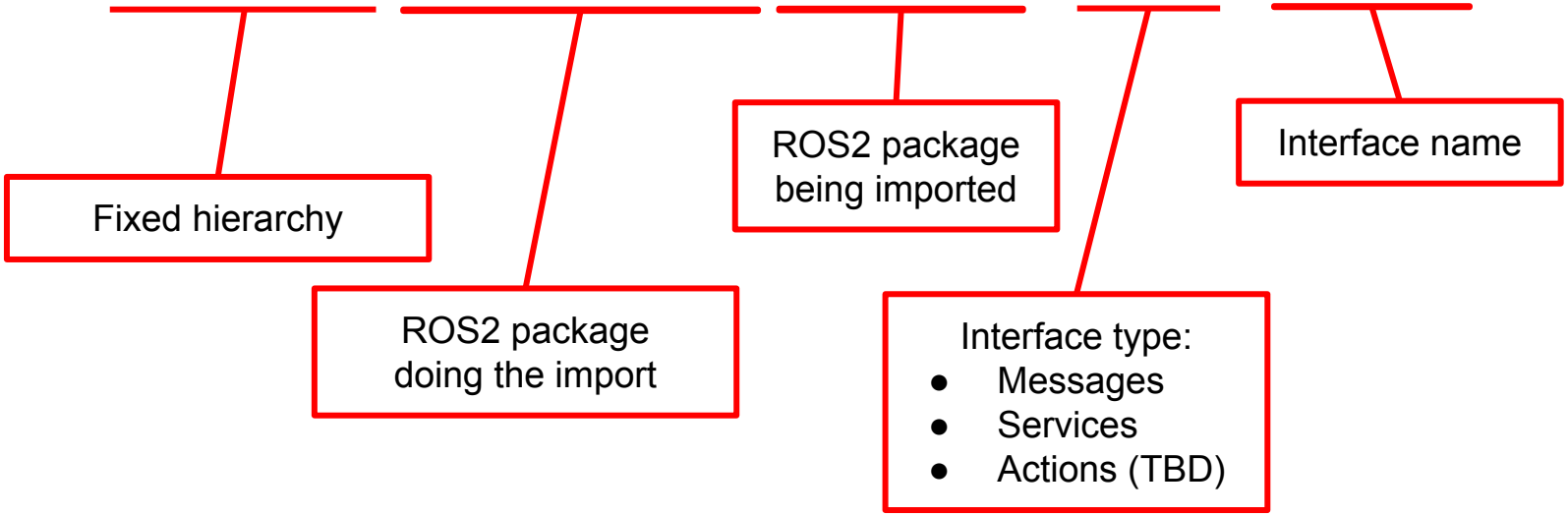
Reference to fields
- No data copy
- Type-checked

1D vector indexing
- Bounds checked

Matrix indexing
- Tuple of indices
- Dimensions checked

Raw C messages / Managed Ada messages

```
package ROSIDL.Static.Tutorial_Solutions.Geometry_Msgs.Messages.Posestamped is
```



Generated Ada types corresponding to C structs

```
package ROSIDL.Static.Tutorial_Solutions.Geometry_Msgs.Messages.Posestamped is

   type Message is limited record
      Header : Std_Msgs.Messages.Header.Message;
      Pose   : Geometry_Msgs.Messages.Pose.Message;
   end record
     with Convention ⇒ C;

   package Handling is new
     ROSIDL.Static.Message
       (Pkg       ⇒ "geometry_msgs",
        Name      ⇒ "PoseStamped",
        Part      ⇒ ROSIDL.Message,
        C_Message ⇒ Message);
```

“Handling” package

```
package ROSIDL.Static.Tutorial_Solutions.Geometry_Msgs.Messages.Posestamped.Handling is

   Support : constant Typesupport.Message_Support;
   --  Support to create messages of this type


   type Message is new Wrapped_Message with private;
   --  Use this type for RAII managed memory

   function Data (This : Message) return C_Message_Access;
   --  Access the underlying fields in the raw C C_Message type

   function Dynamic (This : Message) return ROSIDL.Dynamic.Message;
   --  Access the same message via introspection
```

package RCL.Logging is

```
procedure Set_Level (Severity : Levels;
                     Name     : String := "");
--  Sets the minimum level that will be output.

procedure Info (Message  : String;
                Locate   : Boolean      := False;
                Location : Log_Location := Logging.Location;
                Name     : String       := "");

procedure Warn (Message  : String;
                Locate   : Boolean      := False;
                Location : Log_Location := Logging.Location;
                Name     : String       := "");
```

CLIENT CALL TIMEOUTS

Client timeouts guarantee

- Exception on timeout
- Callback not used after timeout

Connect timeout vs regular

- Connect: time until service is available
- Regular: time until response is received

Everything on the stack: Ada indefinite types

```
declare
   type Int_Array is array (Positive range <>)
                          of Integer;

   Arr   : Int_Array (1 .. 100);
   Hello : constant String := "Hello";

   Other_Arr : Int_Array (1 .. Get_Elsewhere);
begin
   -- Variable stack use so measure it or limit it!
end;

declare
   type Unconstrained (Length : Natural) is record
      Name : String (1 .. Length);
   end record;

   U1 : constant Unconstrained := Get_Unconstrained;
   U2 :          Unconstrained (10);
begin
```

Indefinite type *(unknown size at compile time)*
- **but definite values! *(known size at runtime)***

Constrained by declaration with static size

Constrained by initialization with static size

Constrained by declaration with unknown size at compile time

RCLAda does not use heap allocations

Constrained by initialization with unknown size

Constrained by declaration with static size

ROS2 allocators ⇔ Ada storage pools

- Ada defines Storage_Pool type for different:
  - memory areas (typical in some small boards)
  - allocation policies (including user-defined)
- ROS2 allocators mapped into Ada storage pools
  - Idiomatic use in Ada
  - Reuse of existing pools (e.g., GNAT.Debug_Pools)

```
$ rclada_test_allocators 1
Total allocated bytes:              2335
Total logically deallocated bytes:  2335
Total physically deallocated bytes: 0
Current Water Mark:                 0
High Water Mark:                    415

$ rclada_test_allocators 4
Total allocated bytes:              8095
Total logically deallocated bytes:  8095
Total physically deallocated bytes: 0
Current Water Mark:                 0
High Water Mark:                    415
```

```
type Int_Ptr is access Integer -- named pointer type
  with Storage_Pool => Deterministic_Pool;

type Node_Access is access all RCL.Nodes.Node'Class
  with Storage_Size => 0;       -- No heap allocations

pragma No_Allocators;
pragma No_Implicit_Heap_Allocations;
pragma No_Standard_Allocators_After_Elaboration;
pragma No_Standard_Storage_Pools;
--  See Restrictions in GNAT manual for many more
```

Allocator details

```
typedef struct rcutils_allocator_t
{
  void * (*allocate)(size_t size,
                     void * state);

  void (* deallocate)(void * pointer,
                      void * state);

  void * (*reallocate)(void * pointer,
                       size_t size,
                       void * state);

  void * (*zero_allocate)(size_t number_of_elements,
                          size_t size_of_element,
                          void * state);
                          void * state;
} rcutils_allocator_t;
```

```
package System.Storage_Pools is

   type Root_Storage_Pool is tagged private;

   procedure Allocate
     (Pool                     : in out Root_Storage_Pool;
      Storage_Address          :    out Address;
      Size_In_Storage_Elements : in     Storage_Count;
      Alignment                : in     Storage_Count)
   is abstract;

   procedure Deallocate
     (Pool                     : in out Root_Storage_Pool;
      Storage_Address          : in     Address;
      Size_In_Storage_Elements : in     Storage_Count;
      Alignment                : in     Storage_Count)
   is abstract;
```

```
Pool  : aliased GNAT.Debug_Pools.Debug_Pool;                    -- Ada pool, compiler provided
Alloc : aliased RCL.Allocators.Allocator (Pool'Access);         -- ROS2 allocator, wrapping Ada pool
Node  :         RCL.Node := Node.Init
                            (Options => (Allocator => Alloc'Access)); -- Set node allocator
```

Concurrent executors

```
package RCL.Executors.Concurrent is

   type Runner_Pool is array (Positive range <>) of Runner;
   -- Runner task type declaration omitted

   type Executor (Max_Nodes  : Count_Type :=
                     Default_Nodes_Per_Executor;
                  Queue_Size : Count_Type :=
                     Count_Type (System.Multiprocessors.Number_Of_CPUs) * 32;
                  Threads    : Positive :=
                     Positive (System.Multiprocessors.Number_Of_CPUs);
                  Priority   : System.Priority :=
                     System.Max_Priority) is
   new Executors.Executor (Max_Nodes) with
      record
         Pool  : Runner_Pool  (1 .. Threads);
         Queue : Queues.Queue (Capacity => Queue_Size,
                               Ceiling  => Priority);
         Started : Boolean := False;
      end record;

end RCL.Executors.Concurrent;
```

Parent abstract type in RCL.Executors

Task pool type

Executor type with discriminants
- **# of nodes supported**
- **Queue size**
- **Threads in the pool**
- **Priority**

System.* defined in ARM

OO derivation syntax

Members constrained by discriminants
- Standard Ada bounded queues
- All Ada bounded containers are stack based

See `rclada_test_multicore.adb`
- One producer
- Pooled consumers

rclada vs rclcpp

```
class Talker : public rclcpp::Node
{
public:
 explicit Talker : Node("talker")
 {
   msg_ = std::make_shared<std_msgs::msg::String>();

   auto publish_message = [this]() -> void
     {
       msg_->data =
         "Hello World: " + std::to_string(count_++);
       pub_->publish(msg_);
     };

   pub_   = this->create_publisher
              <std_msgs::msg::String>(topic_name);
   timer_ = this->create_wall_timer(1s, publish_message);
 }
private:
  size_t count_ = 1;
  std::shared_ptr<std_msgs::msg::String> msg_;
  rclcpp::Publisher<std_msgs::msg::String>::SharedPtr pub_;
  rclcpp::TimerBase::SharedPtr timer_;
};

int main(int argc, char * argv[])
{
 auto topic = std::string("chatter");
 auto node = std::make_shared<Talker>(topic);
 rclcpp::spin(node);
}
```

```
procedure Talker is
  Support : constant ROSIDL.Typesupport.Message_Support :=
    ROSIDL.Typesupport.Get ("std_msgs", "String");

  Node : Nodes.Node             := Nodes.Init
                                     (Utils.Command_Name);
  Pub : Publishers.Publisher    := Node.Publish
                                     (Support, "/chatter");
  Msg : ROSIDL.Dynamic.Message := ROSIDL.Dynamic.Init
                                     (Support);

  Counter : Positive := 1;

  procedure Callback (Node    : in out Nodes.Node'Class;
                      Timer   : in out Timers.Timer;
                      Elapsed :        Duration)
  is
     Txt : constant String := "Hello World:" & Counter'Img;
  begin
     Msg ("data").Set_String (Txt);
     Pub.Publish (Msg);
     Counter := Counter + 1;
  end Callback;

begin
  Node.Timer_Add (1.0, Callback'Access);
  Node.Spin;
end Talker;
```

C++

```cpp
class Talker : public rclcpp::Node
{
public:
 explicit Talker : Node("talker")
 {
   msg_ = std::make_shared<std_msgs::msg::String>();

   auto publish_message = [this]() -> void
     {
       msg_->data =
         "Hello World: " + std::to_string(count_++);
       pub_->publish(msg_);
     };

   pub_   = this->create_publisher
              <std_msgs::msg::String>(topic_name);
   timer_ = this->create_wall_timer(1s, publish_message);
 }
private:
  size_t count_ = 1;
  std::shared_ptr<std_msgs::msg::String> msg_;
  rclcpp::Publisher<std_msgs::msg::String>::SharedPtr pub_;
  rclcpp::TimerBase::SharedPtr timer_;
};

int main(int argc, char * argv[])
{
 auto topic = std::string("chatter");
 auto node = std::make_shared<Talker>(topic);
 rclcpp::spin(node);
}
```

```
procedure Talker is
  Support : constant ROSIDL.Typesupport.Message_Support :=
    ROSIDL.Typesupport.Get ("std_msgs", "String");

  Node : Nodes.Node             := Nodes.Init
                                     (Utils.Command_Name);
  Pub : Publishers.Publisher    := Node.Publish
                                     (Support, "/chatter");
  Msg : ROSIDL.Dynamic.Message := ROSIDL.Dynamic.Init
                                     (Support);

  Counter : Positive := 1;

  procedure Callback (Node    : in out Nodes.Node'Class;
                      Timer   : in out Timers.Timer;
                      Elapsed :        Duration)
  is
     Txt : constant String := "Hello World:" & Counter'Img;
  begin
     Msg ("data").Set_String (Txt);
     Pub.Publish (Msg);
     Counter := Counter + 1;
  end Callback;

begin
  Node.Timer_Add (1.0, Callback'Access);
  Node.Spin;
end Talker;
```

C++

```
class Talker : public rclcpp::Node
{
public:
 explicit Talker : Node("talker")
 {
   msg_ = std::make_shared<std_msgs::msg::String>();

   auto publish_message = [this]() -> void
     {
       msg_->data =
         "Hello World: " + std::to_string(count_++);
       pub_->publish(msg_);
     };

   pub_   = this->create_publisher
              <std_msgs::msg::String>(topic_name);
   timer_ = this->create_wall_timer(1s, publish_message);
 }
private:
  size_t count_ = 1;
  std::shared_ptr<std_msgs::msg::String> msg_;
  rclcpp::Publisher<std_msgs::msg::String>::SharedPtr pub_;
  rclcpp::TimerBase::SharedPtr timer_;
};

int main(int argc, char * argv[])
{
 auto topic = std::string("chatter");
 auto node = std::make_shared<Talker>(topic);
 rclcpp::spin(node);
}
```

```
procedure Talker is
  Support : constant ROSIDL.Typesupport.Message_Support :=
    ROSIDL.Typesupport.Get ("std_msgs", "String");

  Node : Nodes.Node             := Nodes.Init
                                     (Utils.Command_Name);
  Pub : Publishers.Publisher    := Node.Publish
                                     (Support, "/chatter");
  Msg : ROSIDL.Dynamic.Message := ROSIDL.Dynamic.Init
                                     (Support);

  Counter : Positive := 1;

  procedure Callback (Node    : in out Nodes.Node'Class;
                      Timer   : in out Timers.Timer;
                      Elapsed :        Duration)
  is
     Txt : constant String := "Hello World:" & Counter'Img;
  begin
     Msg ("data").Set_String (Txt);
     Pub.Publish (Msg);
     Counter := Counter + 1;
  end Callback;

begin
  Node.Timer_Add (1.0, Callback'Access);
  Node.Spin;
end Talker;
```

C++

```
class Talker : public rclcpp::Node
{
public:
 explicit Talker : Node("talker")
 {
   msg_ = std::make_shared<std_msgs::msg::String>();

   auto publish_message = [this]() -> void
     {
       msg_->data =
         "Hello World: " + std::to_string(count_++);
       pub_->publish(msg_);
     };

   pub_   = this->create_publisher
              <std_msgs::msg::String>(topic_name);
   timer_ = this->create_wall_timer(1s, publish_message);
 }
private:
  size_t count_ = 1;
  std::shared_ptr<std_msgs::msg::String> msg_;
  rclcpp::Publisher<std_msgs::msg::String>::SharedPtr pub_;
  rclcpp::TimerBase::SharedPtr timer_;
};

int main(int argc, char * argv[])
{
 auto topic = std::string("chatter");
 auto node = std::make_shared<Talker>(topic);
 rclcpp::spin(node);
}
```

```
procedure Talker is
  Support : constant ROSIDL.Typesupport.Message_Support :=
    ROSIDL.Typesupport.Get ("std_msgs", "String");

  Node : Nodes.Node             := Nodes.Init
                                     (Utils.Command_Name);
  Pub : Publishers.Publisher    := Node.Publish
                                     (Support, "/chatter");
  Msg : ROSIDL.Dynamic.Message := ROSIDL.Dynamic.Init
                                     (Support);

  Counter : Positive := 1;

  procedure Callback (Node    : in out Nodes.Node'Class;
                      Timer   : in out Timers.Timer;
                      Elapsed :        Duration)
  is
     Txt : constant String := "Hello World:" & Counter'Img;
  begin
     Msg ("data").Set_String (Txt);
     Pub.Publish (Msg);
     Counter := Counter + 1;
  end Callback;

begin
  Node.Timer_Add (1.0, Callback'Access);
  Node.Spin;
end Talker;
```

C++

```
class Talker : public rclcpp::Node
{
public:
 explicit Talker : Node("talker")
 {
   msg_ = std::make_shared<std_msgs::msg::String>();

   auto publish_message = [this]() -> void
     {
       msg_->data =
         "Hello World: " + std::to_string(count_++);
       pub_->publish(msg_);
     };

   pub_   = this->create_publisher
              <std_msgs::msg::String>(topic_name);
   timer_ = this->create_wall_timer(1s, publish_message);
 }
private:
  size_t count_ = 1;
  std::shared_ptr<std_msgs::msg::String> msg_;
  rclcpp::Publisher<std_msgs::msg::String>::SharedPtr pub_;
  rclcpp::TimerBase::SharedPtr timer_;
};

int main(int argc, char * argv[])
{
 auto topic = std::string("chatter");
 auto node = std::make_shared<Talker>(topic);
 rclcpp::spin(node);
}
```

```
procedure Talker is
  Support : constant ROSIDL.Typesupport.Message_Support :=
    ROSIDL.Typesupport.Get ("std_msgs", "String");

  Node : Nodes.Node             := Nodes.Init
                                     (Utils.Command_Name);
  Pub : Publishers.Publisher    := Node.Publish
                                     (Support, "/chatter");
  Msg : ROSIDL.Dynamic.Message := ROSIDL.Dynamic.Init
                                     (Support);

  Counter : Positive := 1;

  procedure Callback (Node    : in out Nodes.Node'Class;
                      Timer   : in out Timers.Timer;
                      Elapsed :        Duration)
  is
     Txt : constant String := "Hello World:" & Counter'Img;
  begin
     Msg ("data").Set_String (Txt);
     Pub.Publish (Msg);
     Counter := Counter + 1;
  end Callback;

begin
  Node.Timer_Add (1.0, Callback'Access);
  Node.Spin;
end Talker;
```

C++

```cpp
class Talker : public rclcpp::Node
{
public:
 explicit Talker : Node("talker")
 {
   msg_ = std::make_shared<std_msgs::msg::String>();

   auto publish_message = [this]() -> void
     {
       msg_->data =
         "Hello World: " + std::to_string(count_++);
       pub_->publish(msg_);
     };

   pub_   = this->create_publisher
              <std_msgs::msg::String>(topic_name);
   timer_ = this->create_wall_timer(1s, publish_message);
 }
private:
  size_t count_ = 1;
  std::shared_ptr<std_msgs::msg::String> msg_;
  rclcpp::Publisher<std_msgs::msg::String>::SharedPtr pub_;
  rclcpp::TimerBase::SharedPtr timer_;
};

int main(int argc, char * argv[])
{
 auto topic = std::string("chatter");
 auto node = std::make_shared<Talker>(topic);
 rclcpp::spin(node);
}
```

```
procedure Talker is
  Support : constant ROSIDL.Typesupport.Message_Support :=
    ROSIDL.Typesupport.Get ("std_msgs", "String");

  Node : Nodes.Node             := Nodes.Init
                                     (Utils.Command_Name);
  Pub : Publishers.Publisher    := Node.Publish
                                     (Support, "/chatter");
  Msg : ROSIDL.Dynamic.Message := ROSIDL.Dynamic.Init
                                     (Support);

  Counter : Positive := 1;

  procedure Callback (Node    : in out Nodes.Node'Class;
                      Timer   : in out Timers.Timer;
                      Elapsed :        Duration)
  is
     Txt : constant String := "Hello World:" & Counter'Img;
  begin
     Msg ("data").Set_String (Txt);
     Pub.Publish (Msg);
     Counter := Counter + 1;
  end Callback;

begin
  Node.Timer_Add (1.0, Callback'Access);
  Node.Spin;
end Talker;
```

C++

```cpp
class Talker : public rclcpp::Node
{
public:
 explicit Talker : Node("talker")
 {
   msg_ = std::make_shared<std_msgs::msg::String>();

   auto publish_message = [this]() -> void
     {
       msg_->data =
         "Hello World: " + std::to_string(count_++);
       pub_->publish(msg_);
     };

   pub_   = this->create_publisher
              <std_msgs::msg::String>(topic_name);
   timer_ = this->create_wall_timer(1s, publish_message);
 }
private:
  size_t count_ = 1;
  std::shared_ptr<std_msgs::msg::String> msg_;
  rclcpp::Publisher<std_msgs::msg::String>::SharedPtr pub_;
  rclcpp::TimerBase::SharedPtr timer_;
};

int main(int argc, char * argv[])
{
 auto topic = std::string("chatter");
 auto node = std::make_shared<Talker>(topic);
 rclcpp::spin(node);
}
```

```
procedure Talker is
  Support : constant ROSIDL.Typesupport.Message_Support :=
    ROSIDL.Typesupport.Get ("std_msgs", "String");

  Node : Nodes.Node             := Nodes.Init
                                     (Utils.Command_Name);
  Pub : Publishers.Publisher    := Node.Publish
                                     (Support, "/chatter");
  Msg : ROSIDL.Dynamic.Message := ROSIDL.Dynamic.Init
                                     (Support);

  Counter : Positive := 1;

  procedure Callback (Node    : in out Nodes.Node'Class;
                      Timer   : in out Timers.Timer;
                      Elapsed :        Duration)
  is
     Txt : constant String := "Hello World:" & Counter'Img;
  begin
     Msg ("data").Set_String (Txt);
     Pub.Publish (Msg);
     Counter := Counter + 1;
  end Callback;

begin
  Node.Timer_Add (1.0, Callback'Access);
  Node.Spin;
end Talker;
```

Node canonical examples

Talker

```
procedure Talker is
   Support : constant ROSIDL.Typesupport.Message_Support :=
               ROSIDL.Typesupport.Get_Message_Support ("std_msgs", "String");
   Node    : Nodes.Node           := Nodes.Init   (Utils.Command_Name);
   Pub     : Publishers.Publisher := Node.Publish (Support, "/chatter");

   task Publisher;
   task body Publisher is
      Count  : Positive               := 1;
      Period : constant Duration      := 1.0;
      Next   : Calendar.Time          := Calendar.Clock;
      Msg    : ROSIDL.Dynamic.Message := ROSIDL.Dynamic.Init (Support);
   begin
      loop
         Msg ("data").Set_String ("Hello World:" & Count'Img);
         delay until Next;
         Pub.Publish (Msg);
         Counter := Count + 1;
         Next    := Next  + Period; -- Next := @ + Period; -- in Ada 202x
       end loop;
   end Publisher;

begin
   Node.Spin (Until => Forever);
end Talker;
```

Dynamic handle retrieval

Node initialization in the stack

Topic creation

An Ada task without sync entries

Duration is a built-in Ada type

Message allocation

Message fields are
- indexed by name
- type checked
- bounds checked

Delay without drift

Spin forever (named parameter)

Listener

```
procedure Listener is

   procedure Callback (Node : in out Nodes.Node'Class;
                       Msg  : in out ROSIDL.Dynamic.Message;
                       Info :        ROSIDL.Message_Info) is
   begin
      Logging.Info ("Got chatter: '" & Msg ("data").Get_String & "'");
   end Callback;

   Node : Nodes.Node := Nodes.Init ("listener");

begin
   Node.Subscribe
     (ROSIDL.Typesupport.Get_Message_Support ("std_msgs", "String"),
      "/chatter",
      Callback'Access);

   Node.Spin (Until => Forever);
end Listener;
```

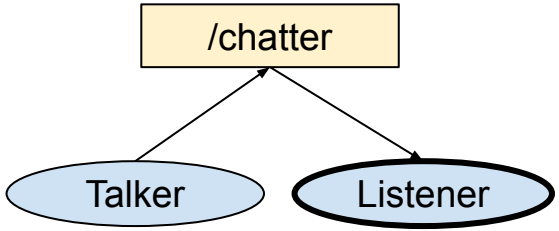
LISTENER

Callback definition

Standard ROS2 Logging

Ada String (not null-terminated)

Register callback
- Using procedure pointer

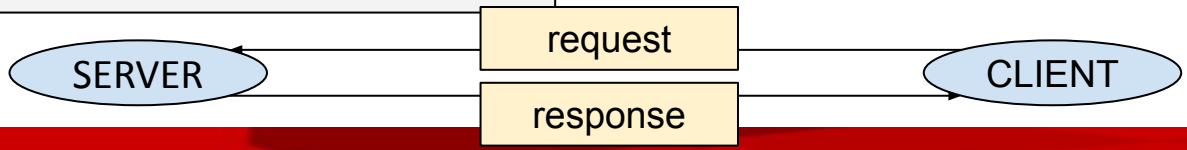


Services

```
procedure Server is
   -- Omitted declarations

   procedure Adder
     (Node : in out Nodes.Node'Class;
      Req  :        ROSIDL.Dynamic.Message;
      Resp : in out ROSIDL.Dynamic.Message)
   is
      A : constant ROSIDL.Int64 := Req ("a").As_Int64;
      B : constant ROSIDL.Int64 := Req ("b").As_Int64;
   begin
      Resp ("sum").As_Int64 := A + B;
   end Adder;

begin
   Node.Serve
     (ROSIDL.Typesupport.Get_Service_Support
        ("example_interfaces", "AddTwoInts"),
      "add_two_ints",
      Adder'Access);
end Server;
```

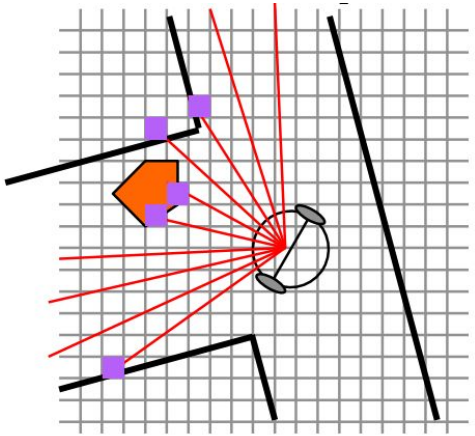
```
procedure Client is -- Synchronous version
   -- Omitted declarations

   Request : ROSIDL.Dynamic.Message := … ;

begin
   Request ("a").As_Int64 := 2;
   Request ("b").As_Int64 := 3;
   declare
      Response : constant ROSIDL.Dynamic.Message :=
                   Node.Client_Call (Support,
                                     "add_two_ints",
                                     Request);
   begin
       Logging.Info ("Got answer:" &
                     Response ("sum").As_Int64.Image);
   end;
end Client;
```

Blocking call (if desired)

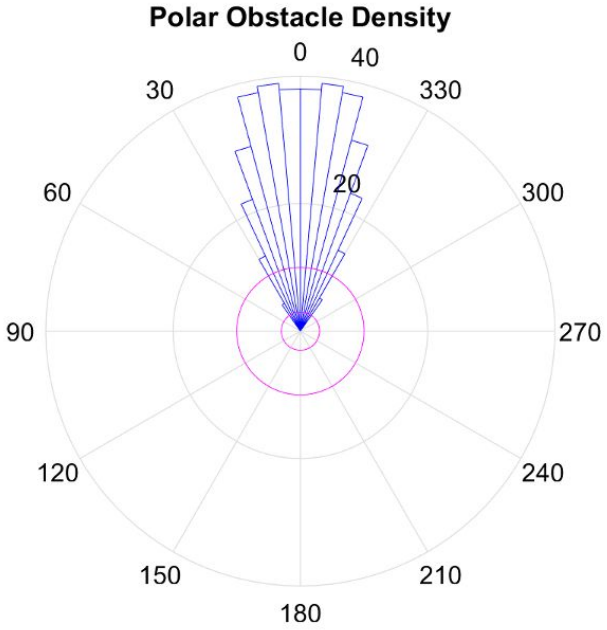


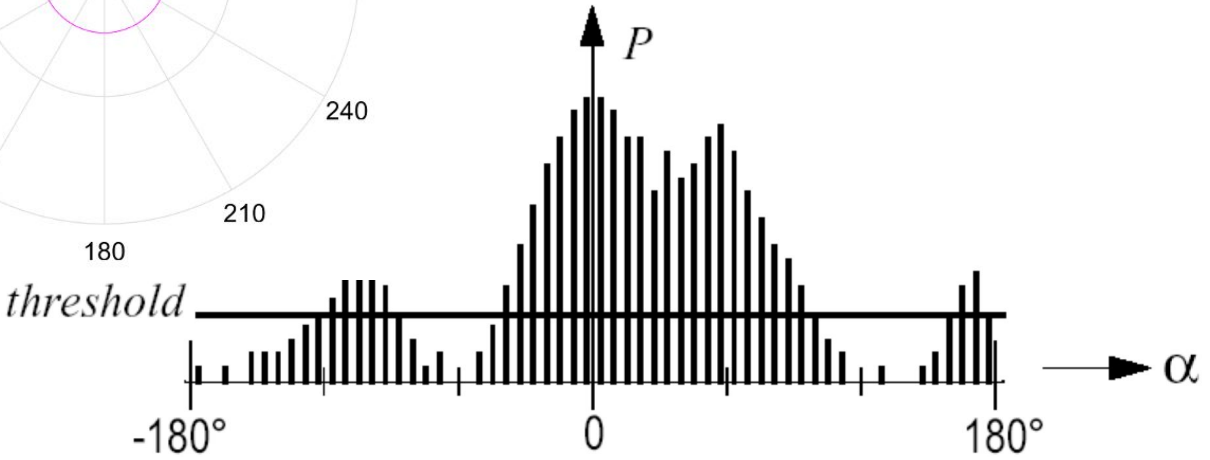
Robotics specifics

VECTOR FIELD HISTOGRAM

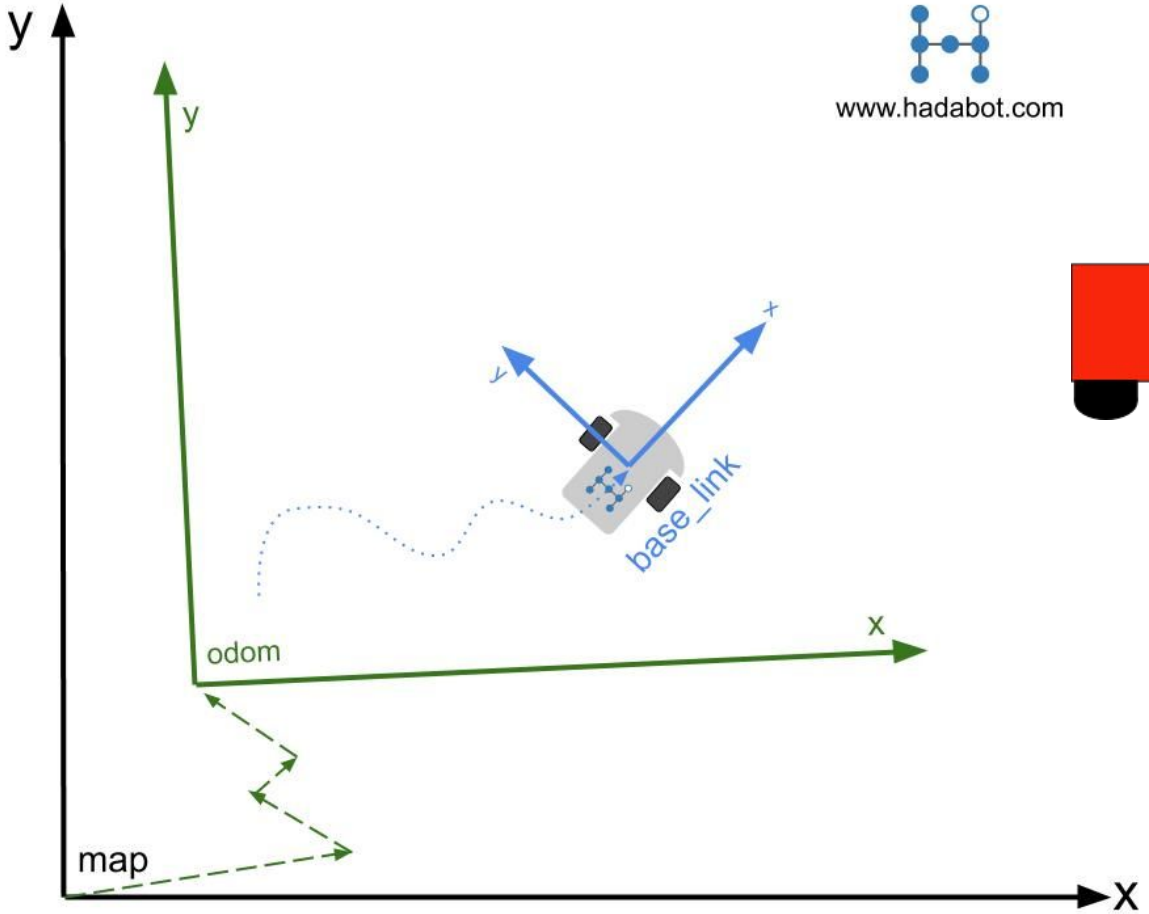


Adapted from © R. Siegwart, I. Nourbakhsh.
Source: https://es.mathworks.com/help/nav/ug/vector-field-histograms.html
Source: Lynn E. Parker,
http://web.eecs.utk.edu/~leparker/Courses/CS594-fall08/Lectures/Oct-23-ObstAvoid-II+Architectures.pdf



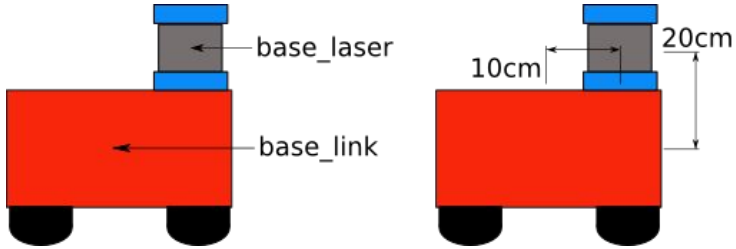


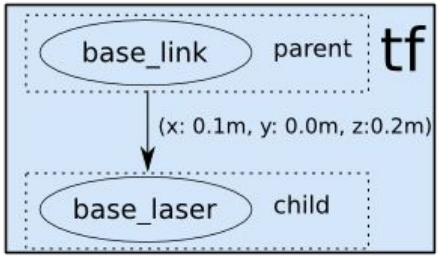
REFERENCE FRAMES



www.hadabot.com

https://blog.hadabot.com/ros2-navigation-tf2-tutorial-using-turtlesim.html

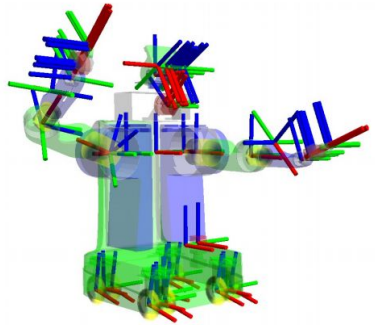




https://navigation.ros.org/setup_guides/transformation/setup_transforms.html

REFERENCE FRAMES

```
function Can_Transform (From, Into : String) return Boolean;

function Transform (Point : Point3D;
                    From,
                    Into  : String)
                    return Point3D;
--  May raise if either the transformation is unknown or else the node is
--  shutting down.

procedure Publish_Transform
  (From, Into  : String;
   Translation : TF2.Translation;
   Rotation    : TF2.Euler;
   Static      : Boolean := False);
--  A static transform is published via static broadcast, and is not
--  expected to change (fixed parts of robots).
```



CONCLUSION

- CMake functions for build integration
  - Ada nodes have same standing as other nodes
- Ada API for pure Ada node writing
  - Ada nodes can interact with other language nodes
- RCLAda distinguishing features (vs rclcpp, rclpy, others)
  - No heap allocations
    - Guaranteed by language restrictions & libraries
  - Relies on automatic low-level binding
    - Early detection of mismatches on ROS2 API changes
  - Language ingrained in safety/HRT culture
    - All message data accesses are type and bounds checked (at runtime)
    - Static message generation is forthcoming

THANKS FOR YOUR INTEREST

https://github.com/ada-ros/ada4ros2/

amosteo@unizar.es

@mosteobotic

Centro Universitario de la Defensa Zaragoza
cud.unizar.es

Academia General Militar · Ctra. Huesca s/n · 50090 Zaragoza · 976 739 500