RCLAda, or bringing Ada to the Robotic Operating System
• Motivation & Context
  – About us
  – What is ROS2
  – Why Ada
  – ROS2 example

• RCLAda Architecture
  – Methodology
  – Integration

• RCLAda API
  – Examples
Robotics, Perception and Real-Time group - RoPeRT
University of Zaragoza
Engineering Research Institute of Aragon
Optimal distributed coordination

Real-time multi-hop communication

Underground drone reconnaissance

http://robots.unizar.es/
It is now possible to write ROS2 nodes in Ada 2012

(with minimal CMake knowledge)
About ROS
• Robot Operating System
  - But not really

“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)
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

ROS2 Main Parts

Build system

- Heterogeneous language environment
- Nodes isolated as processes/threads

Intercommunication facilities

- Message publishing
- Remote Procedure Calls
- Actions

Colcon

ROS2 Main Parts

Build system

- Heterogeneous language environment
- Nodes isolated as processes/threads

Intercommunication facilities

- Message publishing
- Remote Procedure Calls
- Actions

Python

CMake

Plumbing

Tools
PR2: the kick-off robot

https://www.youtube.com/watch?v=c3Cq0sy4TBs
• ROS support widespread
  – Expected in research/academia contexts
  – Either by 1st or 3rd parties

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

- Argos3D P100 ToF camera
- Basler ToF ES camera
- DUO3D™ stereo camera
- Ensenso stereo cameras
- Foreca 3D Laser with SICK LiDAR
- SceneScan and SPL by Neriian Vision Technologies
- OpenNI driver for Kinect and PrimeSense 3D sensors
- Trifo Ironsides
- PMD Camcube 3.0
- IFM O3M250 ToF camera
- Intel® RealSense™ F200/VOF800
- Intel® RealSense™
- Roboception rc_visard stereo camera
- Terabee 3D ToF camera
- Orbec Astra
- SICK MR51xxx lasers
- SICK MR56xxx lasers
- SICK LD-MRS laser (identical to IEBO LUX) or cvideo:asl/sick_ldmrs
- Sents 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
- Pepperl+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
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
At the time of this writing (may’19)

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

ROS 2 and Real-time

Dejan_Pangericic 9d

ROS 2 and Real-time

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
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
4. Bosch proposing how to make Callback-group-level Executor real-time: https://vimeo.com/292707644

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

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
**• 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)
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: Ants Koubaa

Nodes + Topics
1. `$ ros2 pkg create`
   write code

2. `$ colcon build`
   compile code

3. `$ ros2 launch`
   execute code

Ada ROS2 packages
- rclada
- rosidl_generator_ada

CMake functions
- rclada_common
// 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_add_executables(
    my_ada_project  # CMake target name
    ${PROJECT_SOURCE_DIR}  # Path to *.gpr
    bin  # Path to binaries
    my_ada_main)  # Binaries (nodes)

ada_end_package()
- `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_msgs(PKG_NAME)`
  
  Generates bindings to the typesupport handle functions
  
  Could disappear once RCLAda is integrated in build farm

- `ada_generate_binding(TARGET SRCDIR GPRFILE INCLUDE)`
  
  Invokes the binding generator in the context of an Ada project
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;

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);
}
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);
}
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);
    (Support, "/chatter");
    (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;
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;

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_;
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;

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);
    (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;
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;

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);
}
# LaserScan.msg: Single scan from a planar laser range-finder

```idl
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;
RCLAda Architecture
ROS2 client support

- User code
  - Client libraries
    - rclcpp
    - rclpy
  - ROS2 client API (rcl)
  - ROS2 middleware (rmw)
  - DDS implementations
    - FastRTPS
    - Connext

- Threading model
- Intra-process comm
- Namespaces
- Time
- Parameters
- Console logging
ROS2 client support

- **user code**
  - Ada user code

- **client libraries**
  - rclcpp
  - rclpy

- **ROS2 client API (rcl)**

- **ROS2 middleware (rmw)**

- **DDS implementations**
  - FastRTPS
  - Connext

- **rclada**
Ada user code

rclada

rcl

interface description language (rosidl)
- **rclada_client_skeleton**: Empty quickstart package
- **rclada_examples**: talker, listener, add_two_ints, …
- **rclada**: Client API (nodes, topics, services, …) & self-tests
- **rosidl_generator_ada**: Message support
- **rclada_common**: CMake functions
- **gprbuild**: GNAT build system
- **gnat**: Ada compiler (gnat-gcc) and tools
Actual packages

- rclada_client_skeleton
- rclada_examples
- rclada
- rclada_common
- rcl
- rmw
- rosidl_generator_ada
- gprbuild
- gnat
- rosidl_generator_c
- rosidl_typesupport_interface
- rosidl_typesupport_introspection

Ada ROS2 packages
Ada regular tools
ROS2 packages
• **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

```ada
with RCL.Nodes;
procedure My_Shiny_Node is

package RCL is ...
  -- And children RCL.*
  -- Manually written

package rcl_allocator_h is ...
package rcl_client_h is ...
package rcl_node_h is ...
package rcl_*_h is ...
  -- Generated on first colcon build

package ROSIDL is ...
  -- And children ROSIDL.*
  -- Manually written

package builtinInterfaces_*_h is ...
package rosidl_*_h is ...
package std_msgs_*_h is ...
  -- Generated on first colcon build
```
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
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;
ROS2 allocators ↔ Ada storage pools

- Ada defines Storage_Pool type for different:
  - memory areas (typical in some small boards) (associated to pointer types)
  - allocation policies (including user-defined)
- ROS2 allocators mapped into Ada storage pools
  - transparent use in Ada programs
  - immediate testing of RCLAda & ROS2 use of allocators via GNAT.Debug_Pools

```bash
$ 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
```

```ada
type Int_Ptr is access Integer -- named pointer type
with Storage_Pool => Debug_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
```
API & Examples
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);
} 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
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");
begin
  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);
      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;
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;
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

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)
Indefinite concurrent executor type

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;
• 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
THANK YOU FOR YOUR ATTENTION

https://github.com/ada-ros/ada4ros2/
amosteo@unizar.es
@mosteobotic

Acknowledgements:
Dirk Thomas
William Woodall
Esteve Fernandez
AdaCore
Industrial partner

AdaCore

www.adacore.com

Avionics  Defense  Space  ATM

Rail  Automotive  Security

https://www.adacore.com/industries

GNU NYU Ada Translator / FSF GNAT-GCC / SPARK