RCLAda: the Ada client library for ROS2

A.R. Mosteo, D. Tardioli (CUD)
F. Chouteau, Y. Moy (Adacore)
• About us
• Motivation
  – Why ROS2
  – Why Ada
• Architecture
  – Packages
  – CMake helpers
• 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/
AdaCore

www.adacore.com

Avionics
Defense
Space
ATM
Rail
Automotive
Security

https://www.adacore.com/industries

GNU NYU Ada Translator / FSF GNAT-GCC / SPARK
We can now write ROS2 nodes in Ada 2012
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


1991-1997: DoD mandate years
  ● From 450 to 37 languages by 1998

Today: niche in many critical industries
• Structured
  – Separate specifications
• Strongly, statically typed
  – Named types (even pointers)
• Imperative (Pascal-like)
  – Object oriented, optionally
• High-level concurrency
  – Tasks, Rendezvous, Monitors
• Design-by-contract
  – Pre-, post-conditions
  – Type, loop invariants
• Comparable in purpose to C++
  – Emphasis in
    • Maintainability
    • Correctness
    • Early error detection

with Ada.Text_IO; use Ada.Text_IO;

procedure Hello is
begin
  Put_Line ("Hello, ROSCon!");
end Hello;

type Speeds is new Float;
type Lengths is new Float;

spd : Speeds := 0.0;
len : Lengths := spd;
-- Bzzzt

procedure Inc (X : in out Integer)
  with Pre => X < Integer’Last;

type Prime is new Positive
  with Predicate =>
    (for all D in 2 .. Prime / 2 =>
      Prime mod D /= 0);
type Robot_ID is new Natural;  -- Type compatibility is by name
type Task_ID is new Natural;  --
type Distance is range 0 .. 1_000_000_000;  -- Explicit bounds

type Coordinate is range -180.0 .. 180.0;  -- Floating point with range

type Probability is digits 5
  range 0.0 .. 1.0;  -- Floating point with minimum guaranteed precision

type Laser_Readings is delta 10.0 / 2**8
  range 0.0 .. 10.0;  -- Binary fixed point

type Euros is delta 0.001 digits 12;  -- Decimal fixed point

type Weekdays is (Monday, Tuesday, Wednesday, Thursday, Friday);
type Escaped_Robot_Counter is array (Weekdays) of Natural
  with Default_Component_Value => 0;  -- Arbitrarily indexed arrays
• 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)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Packages</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Generics</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Derived ADTs</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Object orientation (tagged types)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Multiple inheritance (abstract interfaces)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Design by Contract</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Numeric types (fixed, floating, decimal, custom)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Tasks</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Monitors</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Real-time systems annex</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Ravenscar profile</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Multiprocessor affinities, Multiprocessor Ravenscar</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Parallel constructs (blocks, loops)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>

Source: https://www.adacore.com/about-ada
ROS2

• Emphasis on
  – Embedded
  – Real-time

• Traditional strong points of Ada
  – Annex C: systems programming
    • Interrupts, atomics, volatiles, task identification
  – Annex D: (hard) real time
    • Priorities, schedulers, monotonic clock, RAVENSCAR
  – Industries requiring certification (aero but… autonomous robots?)

• Related: SPARK Ada subset for formal proofs on code
ROS2 client support

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

Original diagram by Deanna Hood, William Woodall: https://goo.gl/oCHR7H
- user code
- client libraries
  - rclcpp
  - rclpy
- ROS2 client API (rcl)
- ROS2 middleware (rmw)
- DDS implementations
  - FastRTPS
  - Connext
- Ada user code
- rclada
Ada user code

rclada

rcl

interface description language (rosidl)
Ada user code

rclada

rcl

interface description language (rosidl)

threading model

intra-process comm
Actual packages

Ada ROS2 packages
Ada regular tools
ROS2 packages
Ada ROS2 packages

- rclada_client_skeleton
  - Empty quickstart package
- rclada_examples
  - Usual suspects (talker, listener, add_two_ints, …)
- rclada
  - Client facilities (nodes, topics, services, …) & self-tests
- rosidl_generator_ada
  - Message support
- rclada_common
  - CMake functions
  - Ada build system (ARM ch.10)
- gprbuild
  - Ada build system (ARM ch.10)
- gnat
  - Ada compiler (gnat-gcc) and tools

Ada binder ≈ CMake

- Check unit consistency
  - No outdated *.o
- Find safe elaboration order
  - No “init order fiasco”
- Generate main procedure
  - Perform initialization
  - Call user’s main
  - Perform cleanup
package RCL.Logging is

    procedure Initialize;
    -- public methods

private

    -- protected methods
    -- 1-pass compiler needed info

end RCL.Logging;

package body RCL.Logging is

    procedure Initialize is
    begin
        -- do whatever has to be done
        end Initialize;
        -- private methods

    end RCL.Logging;

with RCL.Logging;

procedure RCL.Talker is
begin
    Logging.Initialize;
end RCL.Talker;
• 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 builtin_interfaces_*_h is ...
package rosidl_*_h is ...
package std_msgs_*_h is ...
-- generated on first colcon build
```
• 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

• 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;

Obtain message type
Reference to fields
- No data copy
1D vector indexing
- Bounds checked
Matrix indexing
- Tuple of indices
• 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_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;
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

  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
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

Constrained by initialization with unknown size

Constrained by declaration with static size
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;
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

```
$ 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 => 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
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
• **SPARK**
  - Subset of Ada
    - Same compiler
    - Extra tools for verification/proofs
  - Historically: special comments about code
  - Since Ada 2012/SPARK 2014: Ada contracts
    - Checked by the compiler

• **Can prove:**
  - Absence of runtime errors (exceptions)
    - Runtime checks can be safely disabled
  - Properties of the program
    - Guided by the programmer

• **Similar in some respects to Frama-C**
  - But Ada has fewer undefined behaviors
  - The SPARK subset grows with each version

• **If interested:**
  - Take a free book!
  - Drop by comp.lang.ada (yes, NNTP)

```plaintext
procedure Increment (X : in out Integer)
with Global => null,
Depends => (X => X),
Pre     => X < Integer'Last,
Post    => X = X'Old + 1;
```

```plaintext
type Prime is new Positive
with Dynamic_Predicate =>
  (for all D in 2 .. Prime / 2 =>
   Prime mod D /= 0);
```
**CubeSat from Vermont Tech**
http://www.cubesatlab.org

- Three years of flight time (2013-2016)
- Others: 1x4 month, 2x<1 week, 8x Unheard of

**SND navigation algorithm**
https://github.com/riveras/spark-navigation

- IROS 2014 paper on errors in robotic navigation algorithms
- SND reimplemented in SPARK
- Proven without runtime errors
- Possible target to integrate with RCLAda

**IRONSIDES DNS server**
https://ironsides.martincarlisle.com/

Proven free of (among others):
- Buffer overflows
- Integer overflows
- Information leaks
- Race conditions

**Tokeneer ID station project (NSA)**
https://www.adacore.com/tokeneer

Not only SPARK but full development methodology
- Formal language (Z) for specification
- ~10KLOC
- 4 defects since delivery
DISTINGUISHING FEATURES

• No heap allocations (in RCL)
  – 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
  – Enforced safe program initialization / task completion
  – Strong static type system (incl. numerics) (plus predicates)
  – A convenient path to formal verification with SPARK
  – SPARK is compiled with the same Ada toolset

• Strong backwards & cross-platform compatibility
THANKS FOR YOUR ATTENTION

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

amosteo@unizar.es

@mosteobotic

Acknowledgements:
Dirk Thomas
William Woodall
Esteve Fernandez
ROS answers

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