ROS2 for Android, iOS and Universal Windows Platform

Esteve Fernandez
esteve@apache.org

Outline

- Introduction
- The ROS2 architecture
- Overview of the changes needed in ROS2
- rcljava, rclobjc and rcldotnet APIs compared to rclcpp and rclpy
- How to integrate a new language into ROS2
- Conclusions

Introduction (I)

- Started in late 2015
- Original goal was to run ROS2 on the Raspberry Pi and the Lego EV3 Mindstorm

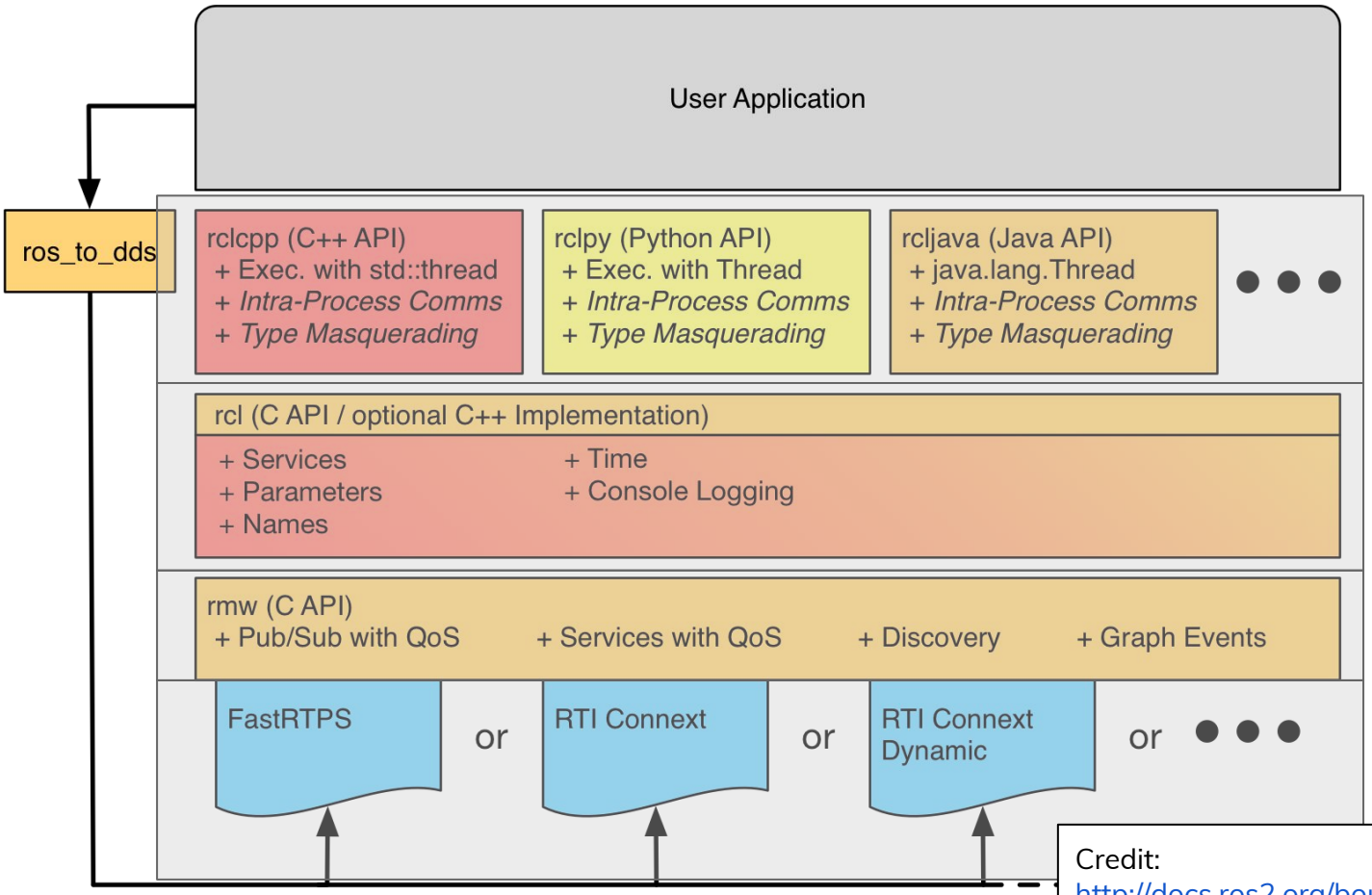
Introduction (II)

- ROS2 is ideal for single-board computers
  - Written in C and C++
  - No master
  - Low memory footprint
- Unfortunately building ROS2 on them natively takes a significant amount of time
  - Solution: Crosscompile!

Raspberry Pi 2
ARM Cortex A7 @ 900 MHz
Main memory
1 GB (shared with GPU)

Lego EV3
TI Sitara AM1808 (ARM926EJ-S core)
@300 MHz
Main Memory
64 MB RAM **16 MB** Flash

The ROS2 architecture



* *Intra-Process Comms* and *Type Masquerading* could be implemented in the client library, but may not currently exist.

Credit:
http://docs.ros2.org/bouncy/developer_overview.html#internal-api-architecture-overview

Crosscompiling ROS2 to Linux-based ARM Single-board computers (I)

- OpenSplice's build system was not easy to work with and the project had become stagnant for a while
  - This has changed since then, it's now a very active Eclipse project under the name Cyclone DDS https://github.com/eclipse/cyclonedds

- Fortunately, the ROS2 architecture allows us to switch DDS implementations
  - FastRTPS was not as mature, but was more active and it used CMake, so cross-compiling it was a bit easier

Crosscompiling ROS2 to Linux-based ARM Single-board computers (II)

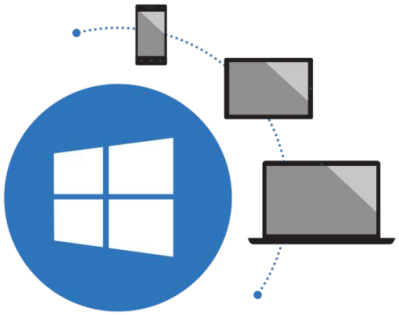
- catkin workspaces are not easily relocatable
  - In ROS2 ament_tools and colcon no longer have a devel space, so copying a workspace to the target platform is much easier

- Minor changes to ROS2 were required to ensure that all basic dependencies can be cross-compiled (e.g. PCRE, tinyxml, tinyxml2)

- Scripts for crosscompiling ROS2 to the Raspberry Pi 2 (or any other ARM-based Single-board computer)
  https://github.com/esteve/ros2_raspbian_tools

Why Android, iOS and UWP?

- Smart devices come with plenty of standard sensors: IMU, GPS, cameras, etc.
- iOS and Android have support for depth sensors for visual-inertial odometry and motion tracking: ARKit and ARCore/Tango
- Not only smart devices! Mixed reality devices (HoloLens) and IoT (Android Things and Windows 10 IoT Core)



I 

C++11

Changes in ROS2 for Android, iOS and UWP (I)

- ROS2 itself is very portable thanks to strict adherence to C11 and C++11/14

- The only changes to rmw and rcl were minor compatibility fixes, for example:

  - iOS 9 only supports pthread-based thread-local storage
  - UWP can’t access environment variables

Changes in ROS2 for Android, iOS and UWP (II)

- FastRTPS required significant changes:
  - Migrate to C++11
  - Replace Boost with embedded-friendly alternatives
  - Add support for Android, iOS and UWP
  - Add support for C typesupport

- Support statically built executables (for iOS)

- With C typesupport and rcl, any language that has a C API can be added

Integrating a new language

- There are many ways to add a new language
  - The following is just how it was done for rclpy, rcljava, rclobjc and rcldotnet

- Adding a new language requires a code generator and a client library
  - The code generator
  - The client library
  - (Optional) The build system

Integrating a new language: the code generator (I)

- Create an ament_cmake package for the generator
  - Add `<member_of_group>rosidl_generator_packages</member_of_group>` in package.xml so that it'll get picked up when message generation is triggered
  - Add `ament_index_register_resource("rosidl_generator_packages")` in CMakeLists.txt to add it to the ament registry
  - Add an ament extension with the CMake code that will trigger the generator and register it with
    `ament_register_extension("rosidl_generate_interfaces" "rosidl_generator_mylang" "PATH_TO_THE_CMAKE_FILE_THAT_TRIGGERS_THE_GENERATOR.cmake")`
  - Depend on `rosidl_generator_c`
  - You can write the generator itself in any language, but the code generators in ROS2 are written in Python and use EmPy

Example: std_msgs.msg.String (Java)

```
public final class String implements MessageDefinition {
...
  private java.lang.String data = "";

  public String setData(final java.lang.String data) {

    this.data = data;
    return this;
  }

  public java.lang.String getData() {
    return this.data;
  }
...
}
```

Example: std_msgs.msg.String (Java)

```
public final class String implements MessageDefinition {
  static {
    try {
      JNIUtils.loadTypesupport(String.class);
    } catch (UnsatisfiedLinkError ule) {
      logger.error("Native code library failed to load.\n" + ule);
      System.exit(1);
    }
  }

  public static native long getDestructor();
  public static native long getFromJavaConverter();
  public static native long getToJavaConverter();
  public static native long getTypeSupport();
...
}
```

Example: std_msgs.msg.String (JNI)

```
std_msgs__msg__String * std_msgs_String__convert_from_java(jobject _jmessage_obj,
std_msgs__msg__String * ros_message)
{
...
 if (ros_message == nullptr) {
   ros_message = std_msgs__msg__String__create();
 }
 auto _jfield_data_fid = env->GetFieldID(_jstd_msgs__msg__String_class_global, "data",
"Ljava/lang/String;");
 jstring _jvaluedata = static_cast<jstring>(env->GetObjectField(_jmessage_obj,
_jfield_data_fid));

 if (_jvaluedata != nullptr) {
   const char * _strdata = env->GetStringUTFChars(_jvaluedata, 0);
   rosidl_generator_c__String__assign(
     &ros_message->data, _strdata);
   env->ReleaseStringUTFChars(_jvaluedata, _strdata);
 }
 return ros_message;
}
```

Example: std_msgs.msg.String (JNI)

```
jobject std_msgs_String__convert_to_java(std_msgs__msg__String * _ros_message, jobject
_jmessage_obj)
{
...
 if (_jmessage_obj == nullptr) {
   _jmessage_obj = env->NewObject(_jstd_msgs__msg__String_class_global,
_jstd_msgs__msg__String_constructor_global);
 }
 auto _jfield_data_fid = env->GetFieldID(_jstd_msgs__msg__String_class_global, "data",
"Ljava/lang/String;");
 if (_ros_message->data.data != nullptr) {
   env->SetObjectField(_jmessage_obj, _jfield_data_fid,
env->NewStringUTF(_ros_message->data.data));
 }
 return _jmessage_obj;
}
```

Integrating a new language: the client library

- Create an ament_cmake package for the client library
  - This will include the API that your users will be familiar with
  - Depending on how the target language interoperates with C, you may:
    - Use the language's standard API for C (e.g. rcljava)
    - Link directly against rcl (e.g. rclobjc)
    - Rely on a Foreign-Function Interface "FFI" (e.g. rcldotnet)
  - Depending on the language, CMake might have support for it. If not, you'll need to write the appropriate functions and macros to compile the client library (e.g. rcldotnet)

Example: Publisher (Java)

```
public class PublisherImpl<T extends MessageDefinition> implements Publisher<T> {
...
  private long handle;

  private static native <T extends MessageDefinition> void nativePublish(
      long handle, long messageDestructor, T message);

  public final void publish(final T message) {
    nativePublish(this.handle, message.getDestructorInstance(), message);
  }
...
}
```

Example: Publisher (JNI)

```
JNIEXPORT void JNICALL
Java_org_ros2_rcljava_publisher_PublisherImpl_nativePublish(
 JNIEnv * env, jclass, jlong publisher_handle, jlong jmsg_destructor_handle, jobject jmsg)
{
 rcl_publisher_t * publisher = reinterpret_cast<rcl_publisher_t *>(publisher_handle);
 jclass jmessage_class = env->GetObjectClass(jmsg);
 jmethodID mid = env->GetStaticMethodID(jmessage_class, "getFromJavaConverter", "()J");
 jlong jfrom_java_converter = env->CallStaticLongMethod(jmessage_class, mid);
 convert_from_java_signature convert_from_java =
   reinterpret_cast<convert_from_java_signature>(jfrom_java_converter);
 void * raw_ros_message = convert_from_java(jmsg, nullptr);
 rcl_ret_t ret = rcl_publish(publisher, raw_ros_message);
 destroy_ros_message_signature destroy_ros_message =
   reinterpret_cast<destroy_ros_message_signature>(jmsg_destructor_handle);
 destroy_ros_message(raw_ros_message);
...
}
```

Integrating a new language: the build system

- ament_tools and colcon are build tools which integrate build systems
  - CMake and setuptools included by default

- Modern languages have their own build system

- Adding a new build system requires:

  - Adding a plugin for ament_tools and colcon written in Python
  - Integrate ament into your build system if you have ROS-flavored packages

Example: Gradle

- Gradle
  - Projects contain a build.gradle file
  - Directory layout
    - src/main/{java,kotlin,scala}
    - src/test/{java,kotlin,scala}
- Gradle plugin for ament
  https://plugins.gradle.org/plugin/org.ros2.tools.gradle
  - Configure output directories
  - Resolve ROS dependencies and update CLASSPATH
  - Copy JAR files and libraries for packaging Android apps
  - Generate launch scripts for Java applications

Example: build.gradle

```
apply plugin: 'java'
apply plugin: 'org.ros2.tools.gradle'
sourceCompatibility = JavaVersion.VERSION_1_8
targetCompatibility = JavaVersion.VERSION_1_8

buildscript {
  repositories {
    maven {
      url "https://plugins.gradle.org/m2/"
    }
  }

  dependencies {
    classpath 'gradle.plugin.org.ros2.tools.gradle:ament:0.7.0'
  }
}

ament {
    entryPoints {
        consoleScripts = [
            'subscriber_node = org.ros2.rcljava.examples.composition.SubscriberNode',
        ]
    }
}
```

Examples!

- Both composition and old-style are supported
    - But the examples only show ROS1-like programs

rcljava talker/listener example

```
RCLJava.rclJavaInit();
Node node = RCLJava.createNode(
  "minimal_publisher");

Publisher<std_msgs.msg.String> publisher =
  node.<std_msgs.msg.String>createPublisher(
    std_msgs.msg.String.class, "topic");

std_msgs.msg.String message =
  new std_msgs.msg.String();

int publishCount = 0;
while (RCLJava.ok()) {
  message.setData(
    "Hello, world! " + publishCount);
  publishCount++;
  System.out.println(
    "Publishing: [" + message.getData() +
    "]");
  publisher.publish(message);
  RCLJava.spinSome(node);
  Thread.sleep(1000);
}
```

```
RCLJava.rclJavaInit();
Node node = RCLJava.createNode(
  "minimal_subscriber");

Subscription<std_msgs.msg.String> sub =
  node.<std_msgs.msg.String>createSubscription(
    std_msgs.msg.String.class, "topic",
    msg -> System.out.println("I heard: [" +
    msg.getData() + "]");

RCLJava.spin(node);
```

rclobjc talker/listener example

```
[ROSRCLObjC rclInit];
ROSNode *node =
  [ROSRCLObjC createNode:@"talker"];

ROSPublisher<ROS_std_msgs_msg_String *>
  *publisher =
    [node createPublisher:
      [ROS_std_msgs_msg_String class
    ]:@"chatter"];

ROS_std_msgs_msg_String *msg =
  [[ROS_std_msgs_msg_String alloc] init];
int i = 1;

while ([ROSRCLObjC ok]) {
  [msg setData:
    [NSString stringWithFormat:
      @"Hello World: %d", i]];
  i++;
  NSLog(@"%@\n",
  [NSString stringWithFormat:
    @"Publishing: '%@'", [msg data]]);
  [publisher publish:msg];
  [ROSRCLObjc spinOnce:node];
  [NSThread sleepForTimeInterval:1.0f];
}
```

```
[ROSRCLObjC rclInit];
ROSNode *node =
  [ROSRCLObjC createNode:@"listener"];

ROSSubscription<ROS_std_msgs_msg_String *>
  *sub =
    [node createSubscriptionWithCallback:
      [ROS_std_msgs_msg_String class]
    :@"chatter"
    :^(ROS_std_msgs_msg_String *msg) {
      NSLog(@"%@\n", [msg data]);
    }
  ];

while ([ROSRCLObjC ok]) {
  [ROSRCLObjC spinOnce:node];
}
```

rcldotnet talker/listener example

```
RCLdotnet.Init ();
```

```
INode node = RCLdotnet.CreateNode ("talker");
```

```
IPublisher<std_msgs.msg.String> chatter_pub =
  node.CreatePublisher<std_msgs.msg.String>
    ("chatter");
```

```
std_msgs.msg.String msg =
  new std_msgs.msg.String ();

int i = 1;
while (RCLdotnet.Ok ()) {
  msg.Data = "Hello World: " + i;
  i++;
  Console.WriteLine (
    "Publishing: \"" + msg.Data + "\"");
  chatter_pub.Publish (msg);
  RCLdotnet.SpinOnce (node);
  Thread.Sleep (1000);
}
```

```
RCLdotnet.Init ();
```

```
INode node = RCLdotnet.CreateNode ("listener");
```

```
ISubscription<std_msgs.msg.String> chatter_sub =
  node.CreateSubscription<std_msgs.msg.String> (
    "chatter",
    msg => Console.WriteLine (
    "I heard: [" + msg.Data + "]"));
```

```
RCLdotnet.Spin (node);
```

rcldotnet talker/listener example (even VB.NET!)

```
RCLdotnet.Init()

Dim node As INode =
  RCLdotnet.CreateNode("talker")

Dim chatter_pub As IPublisher(
  Of std_msgs.msg.String) =
    node.CreatePublisher(
      Of std_msgs.msg.String)("chatter")
Dim msg As std_msgs.msg.String =
  New std_msgs.msg.String()
Dim i As Integer = 1

While RCLdotnet.Ok()
  msg.Data = "Hello World: " & i
  i += 1
  Console.WriteLine(
    "Publishing: """ & msg.Data & """")
  chatter_pub.Publish(msg)
  RCLdotnet.SpinOnce (node)
  Thread.Sleep(1000)
End While
```

```
RCLdotnet.Init()

Dim node As INode =
  RCLdotnet.CreateNode("listener")

Dim chatter_sub As ISubscription(
  Of std_msgs.msg.String) =
    node.CreateSubscription(
      Of std_msgs.msg.String)(
        "chatter",
        Function(msg)
          Console.WriteLine(
            "I heard: [" & msg.Data & "]"))

RCLdotnet.Spin(node)
```

rcljava client/service example (service)

```
public static void handleService(final RMWRequestId header,
  final example_interfaces.srv.AddTwoInts_Request request,
  final example_interfaces.srv.AddTwoInts_Response response) {
    System.out.println("request: " + request.getA() + " + " + request.getB());
    response.setSum(request.getA() + request.getB());
  }
```

```
public static void main(final String[] args) throws InterruptedException, Exception {
  RCLJava.rclJavaInit();

  Node node = RCLJava.createNode("minimal_service");

  Service<example_interfaces.srv.AddTwoInts> service =
    node.<example_interfaces.srv.AddTwoInts>createService(
      example_interfaces.srv.AddTwoInts.class, "add_two_ints",
        (RMWRequestId header, example_interfaces.srv.AddTwoInts_Request request,
          example_interfaces.srv.AddTwoInts_Response response)
          -> AddTwoIntsService.handleService(header, request, response));

  RCLJava.spin(node);
}
```

rcljava client/service example (client)

```
RCLJava.rclJavaInit();
Node node = RCLJava.createNode(
  "minimal_client");
```

```
Client<example_interfaces.srv.AddTwoInts> client =
  node.<example_interfaces.srv.AddTwoInts>createClient(
    example_interfaces.srv.AddTwoInts.class, "add_two_ints");
```

```
example_interfaces.srv.AddTwoInts_Request request =
  new example_interfaces.srv.AddTwoInts_Request();
request.setA(2);
request.setB(3);
```

```
Future<example_interfaces.srv.AddTwoInts_Response> future =
  client.asyncSendRequest(request);
```

```
System.out.println(
        "result of " + request.getA() + " + " + request.getB() + " = " + future.get().getSum());
```

```
RCLJava.shutdown();
```

Demo!

Credit: Emiliano Borghi and Ekumen

Conclusions

- One of ROS2's many goals is to support Linux, Windows and macOS from the very beginning
  - And still, it was still possible to add support for Android, iOS and UWP without changing the architecture at all

- Ensuring portability tends to increase the quality of the codebase

- The modular model of ROS2 allows developers to pick and choose whatever components they may need

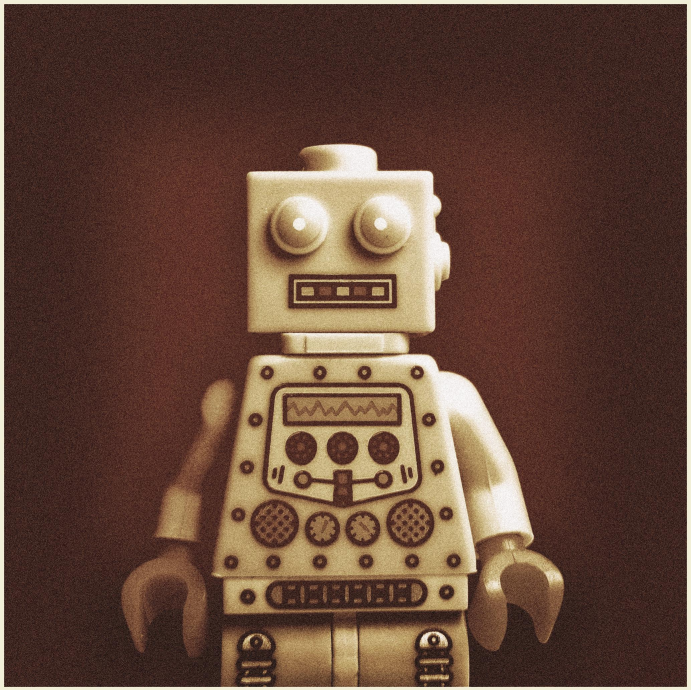
- The new build tools (ament_tools and colcon) make cross-compiling code and adding new build systems (e.g. Gradle for Java) much easier

- Without C11, C++11 and C++14 this probably wouldn't have been possible

Links

- ros2-java / ros2-android
  - https://github.com/esteve/ros2_java
  - https://github.com/esteve/ros2_android
  - https://github.com/esteve/ament_java
  - https://github.com/esteve/ament_gradle_plugin
  - https://github.com/esteve/ros2_java_examples
  - https://github.com/esteve/ros2_android_examples
- ros2-objc / ros2-ios
  - https://github.com/esteve/ros2_objc
  - https://github.com/esteve/ros2_ios_examples
  - https://github.com/esteve/ros2_objc_examples
- ros2-dotnet / ros2-uwp
  - https://github.com/esteve/ros2_dotnet
- Fast-RTPS changes for C++11, Android, iOS and UWP
  - https://github.com/eProsima/Fast-RTPS/pull/26
  - https://github.com/ros2/rmw_fastrtps/pull/27
- Scripts for crosscompiling ROS2 to Raspberry Pi (and ARM-based SBCs)
  - https://github.com/esteve/ros2_raspbian_tools

Questions?



Credit:
https://www.flickr.com/photos/kaptainkobold/9164005127/

@esteve

github.com/esteve

esteve@apache.org

Thanks!