ROS Basics

ARRA / AR2A

Advancements for **R**obotics in **R**escue **A**pplications

March 13, 2016

Overview

1 Introduction

2 Concepts
- ROS File System
- Computation Graph

3 Useful Means

ARRA/AR2A

What do we want?

- ARRA / AR2A aims to improve the current state of technology of robotics in rescue applications.

Who are we?

- A volunteer non-profit organisation of robotic enthusiasts.

How can you help?

- Check us out at `https://github.com/ar2a`

License information

- **CC-BY-SA 4.0**
  `https://creativecommons.org/licenses/by-sa/4.0/`

Section 1

Introduction

History

- 2007: Beginnings as project **switchyard** at the Stanford Artificial Intelligence Laboratory
- 2008: Further development by R&D company Willow Garage ▸ PR2
    - First release: ROS 1.0 (31st December 2012)
- 2013: Adoption of the project responsibilty by the **Open Source Robotics Foundation** (OSRF)

- **Versions**: ROS 1.0, Box Turtle, C Turtle, Diamondback, Electric Emys, Fuerte Turtle, Groovy Galapagos, Hydro Medusa, **Indigo Igloo**, **Jade Turtle**
    - Latest stable version: **Indigo Igloo**
    - Latest version: **Jade Turtle**

What is ROS?

Definition (ROS)

The Robot Operating System (ROS) is a flexible framework for writing robot software. It is a collection of tools, libraries and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms.

Why ROS?

- $>$ **3000** 3rd party packages in the ROS repositories provide solutions for various, in every robotic system reoccuring problems
- Integration with Gazebo, OpenCV, PointCloudLibrary, MoveIt
- Extensive documentation and support by a **vibrant community**
    - $>$ **5700** users at the ROS Q&A-Page (70%-Answer-Rate)
    - $>$ **22000** ROS-Wiki pages
- Permissive **BSD**-Lizenz

ROS
=
Plumbing
+
Tools
+
Capabilities
+
Ecosystem

Figure: Source: [1]

Section 2

Concepts

Design Philosophy 1

- **Peer-To-Peer**
  - Most robotic applications demand a **huge processing power**
  - **Distributed computing** which performs calculations both on on- and offboard systems required
  - Interface for data exchange becomes performance bottleneck in systems with a centralised architecture (central server necessary)
- **Tool based**
  - Despite the name **no operating system but framework** (microkernel approach)
  - + stability
  - + expandability
  - e.g. roscd, roslaunch, rostopic, rosgraph, ...

Design Philosophy 2

- **Multilingual**
  - C++, Python, Octave, Lisp
- **Thin**
  - Encapsulation of algorithms in libraries
    - Communication via defined interfaces
    - Reusability (algorithms are decoupled from hardware)
    - Easy to test
- **Open Source**
  - BSD license allows for commercial applications without limitations

Design Philosophy 3

Definition (Client Library)

A Client library implements the core ROS concepts to create nodes, publish and subscribe to topics, write and call services and use the Parameter Server. Such a library can be implemented in any programming language.

- ROS focus is on robust C++ and Python support.
- ROScpp is designed for high runtime performance
- ROSpy favors implementation speed (i.e. developer time) over runtime performance. ROS Master, roslaunch are developed in ROSpy.

ROS File System

Components

- Packages
- Metapackages
- Package Manifests
- Message types
- Service types

Packages

Definition (Package)

Packages are the most atomic build and release item in ROS. The most granular thing you can build and release is a package.

A package can contain:

- ROS runtime processes (nodes)
- a ROS-dependent library
- datasets
- configuration files
- third party piece of software
- or anything else that is usefully organized together

It's goal is to provide software in an easy to use (reusable) manner.

Metapackages

Definition (Metapackages)

A specialized package, which only serves to represent a group of related other packages.

- Metapackages do not install files and they do not contain any tests, code, files, or other items usually found in packages.
- Most commonly metapackages are used as a backwards compatible place holder for converted rosbuild Stacks.

A Metapackage is like a package with an export tag:

```
<export>
        <metapackage />
</export>
```

Package Manifests

Definition (Package Manifest)

The package manifest is an XML file called package.xml. It provides metadata about a package and it must be included with any catkin-compliant package's root folder.

metadata includes:

- name
- version
- description
- license information
- dependencies
- other meta information like exported packages

Message types

Definition (Message type)

A message type is the name of the .msg file and it describes the message (data structure, ...)

- Nodes can communicate by publishing Messages to **topics** or by exchanging a request and response message as part of a ROS **service** call.
- The .msg file describes the structure of the message (types, arrays, structs, ...).
- The message type is the name of the package + / + msg + / + name of the .msg file.
  Example: my_package/msg/MyMessageType.msg
- ROS tools automatically generate source code for the message type in several target languages.

Service types

Definition (Service type)

A service type is the name of the .srv file and it describes the request and response message of the service.

- A Service is a pair of messages - request and response.
- Therefore a service type consists of two message types - one for the request and one for the response.
- The service type is the name of the package + / + serv + / + name of the .srv file.
  Example: my_package/srv/MyServiceType.srv

ROS Computation Graph

The Computation Graph is the peer-to-peer network of ROS processes that are processing data together.

- Nodes
- Messages
- Topics
- Services
- Master
- Parameter Server
- Bags

Names

There are two types of names:

- Graph Resource Names
  - Every computational component (resource = nodes, topics, services, parameters, ...) has an unique name
  - Each resource is defined within a namespace
  - A namespace can contain many resources
  - A resource can create other resources within it's namespace and it can access other resources within or above it's namespace
  - By default names are resolved relatively
  - Namespaces provide encapsulation and help to choose the right resource (if there are many with the same name)

Every ROS client library supports command-line remapping of names, which means a compiled program can be reconfigured at start-up time to operate in a different Computation Graph topology.

Names

- Package Resource Names
  - naming convention to refer to files and datatypes (e.g. Message, Service or Node types) on disk.
  - Package Resource Names are similar to file paths, but they are shorter, because ROS assumes their location.
    Example: Message descriptions are always stored in the msg subdirectory and have the .msg extension, so std_msgs/String is shorthand for path/to/std_msgs/msg/String.msg.
  - Package Resource Names have strict naming rules as they are often used in auto-generated code.

Nodes

Definition (Node)

A node is a process performing calculations.

- Typical applications consist of multiple nodes
- Naming results from graphical representation of all running processes and the communication between those processes as **directed graph** (Tool: *rqt_graph*)
- Identification of a process by a **graph ressource name** ("Nodename")

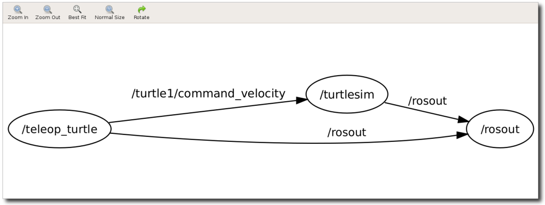


Figure: Source: [1]

Nodes

Advantages of nodes:

- More fault tolerance as crashes are isolated to individual nodes
- Code complexity is reduced
- Implementation details are well hidden as the nodes expose a minimal API to the rest of the graph

Messages

Definition (Messages)

Messages are a means of communication. A message is a simple data structure comprising typed fields.

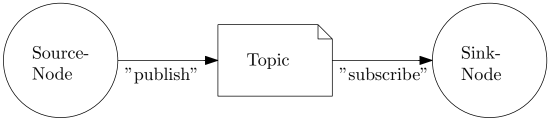
Topics and services use messages, as already describe at the Message Types slide.

Topic

Definition (Topic)

A topic is a unique name which allows nodes to locate their communication partner for the transmission and reception of data.

- Creation and consumption of data is **decoupled** (Producer/Consumer-Pattern)
  - Source's (**publish**) data to a topic
  - Sink's (**subscribe**) data from a topic



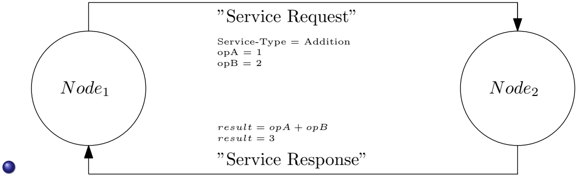
- **Unidirectional** communication
- *Point to Point* and *Point to Multipoint*

Service

Definition (Service)

Services allow for bidirectional communication between two nodes in form of request/response couples.

- **Request** → *Node*$_1$ requests a service from *Node*$_2$ (e.g. addition)
- **Response** → *Node*$_2$ executes service and returns result to *Node*$_2$



- Concrete implementation as a **remote procedure call**
- **Bidirectionale** (*Point to Point*) communication

Master

Definition (Master)

The ROS Master provides name registration and lookup to the rest of the Computation Graph.

- acts as a nameservice
- stores topics and service registration information for ROS nodes
- nodes ask the master for information about other nodes, then the nodes connect to other nodes directly (TCPROS: TCP/IP sockets) - the Master only provides lookup information
- the master makes callbacks to these nodes when the registration information changes - nodes can create dynamic connections as new nodes are run

Without the Master nodes would not be able to find each other, exchange messages, or invoke services.

Master Example

There are two Nodes. A Camera node and an Image_viewer node. The Camera notifies the master that it wants to publish images on the topic "images":

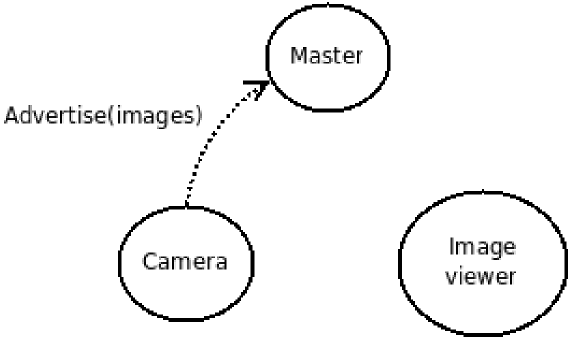


Figure: Master Example 1

Master Example

Camera publishes images to the "images" topic, but nobody is subscribing to that topic yet so no data is actually sent.
Then the Image_viewer subscribes to the topic "images" to see if there are some images there:

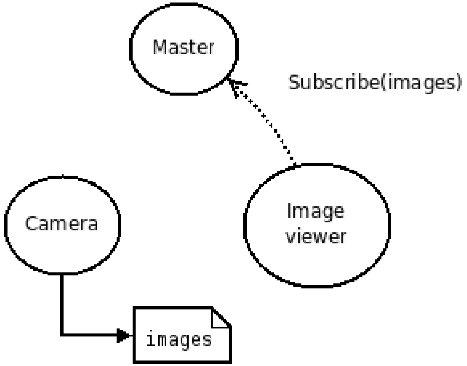


Figure: Master Example 2

Master Example

Now that the topic "images" has both a publisher and a subscriber, the master node notifies Camera and Image_viewer about each others existence so that they can start transferring images to one another:

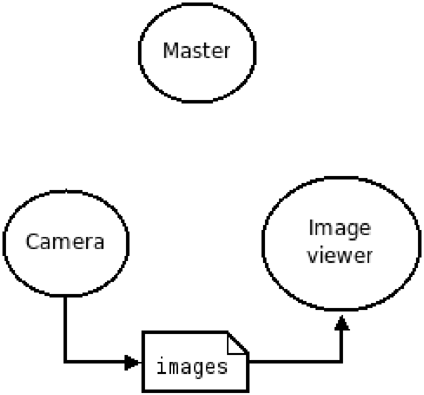


Figure: Master Example 3

Parameter Server

Definition (Parameter Server)

The Parameter Server allows data to be stored by key in a central location. It is a shared, multi-variate dictionary that is accessible via network APIs.

- The Parameter Server is part of the Master.
- It's globally viewable so tools can easily inspect the configuration state of the system and modify it if necessary.

Example: Define the following parameters:
/gains/P = 10.0
/gains/I = 1.0
/gains/D = 0.1
Read separately: /gains/P → returns 10.0
or read the whole dictionary: /gains → { 'P': 10.0, 'I': 1.0, 'D' : 0.1 }

ROS Core

Definition (ROS Core)

roscore is a collection of nodes and programs that are pre-requisites of a ROS-based system (they have to run in order for ROS nodes to communicate).

- use "roscore" command to launch it ("roslaunch" command starts roscore automatically too)
- the roscore includes the master, parameter server and a rosout logging node

Bags

Definition (Bag)

A bag is a file format in ROS for storing ROS message data. It's a primary mechanism in ROS for data logging and has a variety of offline uses.

- Bags are created by a tool like rosbag, which subscribes to one or more ROS topics, and stores the serialized message data in a file as it is received.
- The bag files can be played back in ROS to the same topics they were recorded from, or even remapped to new topics.

Bags are a format for saving and playing back ROS message data. Bags are an important mechanism for storing data, such as sensor data, that can be difficult to collect but is necessary for developing and testing algorithms.

Section 3

Useful Means

Useful Means

- **tf package:** A framework for calculating the positions of multiple coordinate frames over time.
- **actionlib package:** The actionlib defines a common, topic-based interface for preemptible tasks in ROS.
- **common_msgs Stack:**
  - actionlib_msgs: messages for representing actions
  - diagnostic_msgs: messages for sending diagnostic data
  - geometry_msgs: messages for representing common geometric primitives
  - nav_msgs: messages for navigation
  - sensor_msgs: messages for representing sensor data

Useful Means

- **pluginlib:** It provides a library for dynamically loading libraries in C++ code.
- **filters package:** It provides a C++ library for processing data using a sequence of filters.
- **urdf package:** It defines an XML format for representing a robot model and provides a C++ parser.
- **rqt:** rqt is a Qt-based framework for GUI development for ROS.
- **ROS Cheat Sheet**

dynamic_reconfigure

Definition (dynamic_reconfigure)

dynamic_reconfigure is a package which allows you to reconfigure a subset of a node's parameters at any time without having to restart the node.

- Client programs (e.g. GUIs) can query a node for it's set of reconfigureable parameters (name, type, range of the param)
- the API is still under development → significant changes expected
- a reconfigure_gui tool is provided in the rqt library
- the dynparam command-line tool is stable → rosrun dynamic_reconfigure dynparam COMMAND
- the python API is stable, a C++ API is not yet implemented, but you can use the system command as a workaround.

References I

"Ros wiki," 11 2015.
http://www.ros.org/.

The End