Composable Nodes in ROS2

ROSCon 2019

Michael Carroll, Open Robotics
Michel Hidalgo, Ekumen Labs
William Woodall, Open Robotics

Overview

- A composable node is any ROS 2 node that supports composition
  - Composition is aggregation of nodes in a process
  - Node implies node-like (same API)
- In ROS 2, composable nodes are the successor to nodelets.

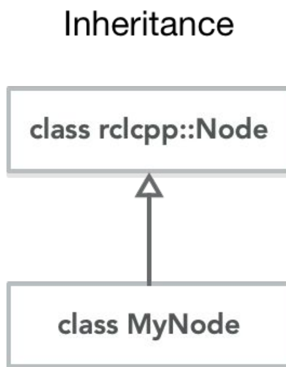
Rationale

- Simplifies code reuse
- Defers process layout decisions to deploy-time
  - Nodes in different processes
    - Allow for fault isolation
    - Are easier to debug
  - Nodes in the same process
    - Allow resource sharing
    - Allow for optimizations

Writing a Component

- Write your node as a class
    - via inheritance

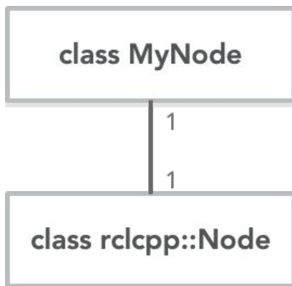
Inheritance




```
// \file include/talker_component.hpp

class Talker : public rclcpp::Node
{
public:
 COMPOSITION_PUBLIC
 explicit Talker(const rclcpp::NodeOptions & options);

protected:
 void on_timer();

private:
 size_t count_;
 rclcpp::Publisher<std_msgs::msg::String>::SharedPtr
pub_;
 rclcpp::TimerBase::SharedPtr timer_;
};
```

Writing a Component

- Write your node as a class
  - via inheritance
  - via composition




```
// \file include/node_like_listener_component.hpp

class NodeLikeListener
{
public:
 COMPOSITION_PUBLIC
 explicit NodeLikeListener(
   const rclcpp::NodeOptions & options);

 COMPOSITION_PUBLIC
 rclcpp::node_interfaces::NodeBaseInterface::SharedPtr
 get_node_base_interface() const;

private:
 rclcpp::Node::SharedPtr node_;
 rclcpp::Subscription<std_msgs::msg::String>::SharedPtr
sub_;
};
```

Writing a Component

- Write your node as a class
  - via inheritance
  - via composition
- Don’t have your own main()

```
// \file src/talker_component.cpp

Talker::Talker(const rclcpp::NodeOptions & options)
: Node("talker", options), count_(0)
{
 pub_ =
create_publisher<std_msgs::msg::String>("chatter", 10);
 timer_ = create_wall_timer(1s,
std::bind(&Talker::on_timer, this));
}

void Talker::on_timer()
{
 auto msg = std::make_unique<std_msgs::msg::String>();
 msg->data = "Hello World: " + std::to_string(++count_);
 pub_->publish(std::move(msg));
}
```

How to build and register a component

- Make your class dynamically loadable
  - via registration macro in C++ sources

```
// \file src/talker_component.cpp

namespace composition
{
// ...
}

#include "rclcpp_components/register_node_macro.hpp"

RCLCPP_COMPONENTS_REGISTER_NODE(composition::Talker)
```

How to build and register a component

- Make your class dynamically loadable
  - via registration macro in C++ sources
- Make it a library and discoverable
  - via CMake macro in your CMakeLists.txt

```
# \file CMakeLists.txt
find_package(rclcpp_components REQUIRED)
# ...
add_library(
  talker_component SHARED src/talker_component.cpp)
# ...
rclcpp_components_register_node(
  talker_component
  PLUGIN “composition::Talker”
  EXECUTABLE talker)
```

How to build and register a component

- Alternatively, build a library of multiple components:

```
# \file CMakeLists.txt
find_package(rclcpp_components REQUIRED)
# ...
add_library(
  demo_components SHARED
  src/talker_component.cpp src/listener_component.cpp)
# ...
rclcpp_components_register_nodes(
  demo_components
  "composition::Talker" "composition::Listener")
```

How to build and register a component

What we have so far:

- Shared library with component(s)
- Standalone executable for convenience
- Registration with tooling

Now how do we use this?

How to deploy a component

- Use ROS 2 Service APIs:
  - Load components via service calls
  - Load components via command line tools
  - Load components via launch
- Alternatively:
  - Manual composition
  - Run standalone executables

How to deploy components

- Using `ros2 component` CLI (live demo)

Terminalizer

```
(eloquent) $
```

How to deploy components

- Using `ros2 component` CLI
- Using launch
  - Run your node in a container

```
# :file: launch/composition_launch.py
def generate_launch_description():
    container = ComposableNodeContainer(
        node_name='my_container'
        package='rclcpp_components',
        node_executable='component_container',
        composable_node_descriptions=[
            ComposableNode(
                package='composition',
                node_plugin='composition::Talker',
                node_name='talker'),  # ...
        ]
    )
    return LaunchDescription([container])
```

How to deploy components

- Using `ros2 component` CLI
- Using launch
  - Run your node in a container
  - Load your node into a running container

```python
# :file: launch/composition_launch.py
def generate_launch_description():
    container = ComposableNodeContainer(
        node_name='my_container',
        package='rclcpp_components',
        node_executable='component_container')
    loader = LoadComposableNodes(
        composable_node_descriptions=[
            ComposableNode(
                package='composition',
                node_plugin='composition::Talker',
                node_name='talker'),  # ...
        ],
        target_container=container,
    )
    return LaunchDescription([container, loader])
```

Design Considerations

- Flexibility
  - Components can be easily reused
  - Only rclcpp components are currently available.
- Efficiency
  - Components are loaded into the same process!
  - Tip: Enable intra process communication to get a performance boost for free

Design Considerations (continued)

- Reliability
  - Crashing one component will crash the entire process
  - Tip: It’s best to run prototype code in a separate process or container
- Security
  - Components in the same process share memory space
  - Tip: Disable dynamic loading of components with secure components
  - Tip (advanced): Authorize trusted node to access dynamic loading services.

Motivating Examples

- Library of common functionality
  - Perception processing pipelines
  - Common topic tools or transformations
- Components as plugin providers
  - Navigation - Loading planners or behaviors
  - SLAM - Loading frontends or backends
- Sharing hardware resources
  - Device drivers

Summary

- Composition allows code reuse and defers process layout decisions.
  - Can execute a composable node in its own process for isolation and security.
  - Can execute multiple composable nodes in a single process for performance and speed.
- Composition is the new best practice when developing ROS2 nodes.

Useful Links:

- https://github.com/ros2/design/pull/206
- https://github.com/ros2/rclcpp/tree/master/rclcpp_components
- https://github.com/ros2/ros2cli/tree/master/ros2component