Lab Activity: Orchestrating ROS 2 Lifecycle Nodes

Objective:

By the end of this session, you will understand how to:

1. Create a node that implements the standard ROS 2 Lifecycle state machine.
2. Create a separate Supervisor node that triggers state transitions (Configure → Activate).
3. Observe how a node exists but remains silent until explicitly activated.

Prerequisites: ROS 2 (Jazzy) installed.

Part 1: Create the Package

First, we create a Python package to hold our scripts. Open a terminal and execute the following commands:

```
# Go to your workspace src folder
cd ~/ros2_ws/src

# Create the package
ros2 pkg create --build-type ament_python lifecycle_demo --dependencies rclpy
std_msgs lifecycle_msgs --license Apache-2.0
```

Part 2: The "Musician" (Lifecycle Node)

This node will be capable of publishing "Hello World", but it won't do so until it enters the **Active** state.

1. Navigate to `~/ros2_ws/src/lifecycle_demo/lifecycle_demo`.
2. Create a file named **`lc_talker.py`**.
3. Paste the following code:

```
import rclpy
from rclpy.lifecycle import Node, State, TransitionCallbackReturn
from std_msgs.msg import String
import time

class LifecycleTalker(Node):
```

```
    def __init__(self):
        super().__init__('lc_talker')
        self.get_logger().info('INSTRUCTOR: Node created (Unconfigured). I am silent.')
        self._pub = None
        self._timer = None

    # TRANSITION 1: Configure
    # Allocates memory, sets up publishers/timers, but DOES NOT start them.
    def on_configure(self, state: State) -> TransitionCallbackReturn:
        self.get_logger().info('INSTRUCTOR: Configuring... creating publisher.')
        self._pub = self.create_lifecycle_publisher(String, 'chatter', 10)
        self._timer = self.create_timer(1.0, self.publish_message)
        self._timer.cancel() # Pause timer immediately
        return TransitionCallbackReturn.SUCCESS

    # TRANSITION 2: Activate
    # Actually starts the work.
    def on_activate(self, state: State) -> TransitionCallbackReturn:
        self.get_logger().info('INSTRUCTOR: Activating... I will now start publishing!')
        super().on_activate(state) # Crucial: Enables the lifecycle publisher
        self._timer.reset() # Unpause timer
        return TransitionCallbackReturn.SUCCESS

    # TRANSITION 3: Deactivate
    # Pauses the work but keeps the setup.
    def on_deactivate(self, state: State) -> TransitionCallbackReturn:
        self.get_logger().info('INSTRUCTOR: Deactivating... Stopping publication.')
        self._timer.cancel()
        super().on_deactivate(state)
        return TransitionCallbackReturn.SUCCESS

    # TRANSITION 4: Cleanup
    # Destroys the publisher/timer to free memory.
    def on_cleanup(self, state: State) -> TransitionCallbackReturn:
        self.get_logger().info('INSTRUCTOR: Cleaning up... Destroying publisher.')
        self.destroy_publisher(self._pub)
        self.destroy_timer(self._timer)
        return TransitionCallbackReturn.SUCCESS

# TRANSITION 5: Shutdown
    def on_shutdown(self, state: State) -> TransitionCallbackReturn:
        self.get_logger().info('INSTRUCTOR: Shutting down. Node is being finalized.')
```

```
        return TransitionCallbackReturn.SUCCESS



    def publish_message(self):
        msg = String()
        msg.data = "Hello World: Lifecycle is Active!"
        if self._pub.is_activated:
            self._pub.publish(msg)
            self.get_logger().info(f'Publishing: "{msg.data}"')

def main():
    rclpy.init()
    node = LifecycleTalker()
    rclpy.spin(node)
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

Part 3: The "Conductor" (Supervisor Node)

This node acts as the system manager. It connects to the lc_talker's service and tells it to switch states.

1. Create a file named **`supervisor.py`** in the same folder.
2. Paste the following code:

```
import rclpy
from rclpy.node import Node
from lifecycle_msgs.srv import ChangeState
from lifecycle_msgs.msg import Transition
import time

class Supervisor(Node):

    def __init__(self):
        super().__init__('supervisor_node')
        self.client = self.create_client(ChangeState, '/lc_talker/change_state')
```

```
    def change_state(self, transition_id):
        self.get_logger().info(f'SUPERVISOR: Requesting transition {transition_id}...')

        # Wait for service
        while not self.client.wait_for_service(timeout_sec=1.0):
            self.get_logger().info('Waiting for /lc_talker/change_state service...')

        req = ChangeState.Request()
        req.transition.id = transition_id

        # Send Request
        future = self.client.call_async(req)
        rclpy.spin_until_future_complete(self, future)

        if future.result() is not None:
            self.get_logger().info(f'SUPERVISOR: Transition Result: {future.result().success}')
        else:
            self.get_logger().error('SUPERVISOR: Service call failed')

def main():
    rclpy.init()
    supervisor = Supervisor()

    # Step 1: Configure
    time.sleep(2) # Wait a moment to see the initial state
    supervisor.change_state(Transition.TRANSITION_CONFIGURE)

    # Step 2: Activate
    time.sleep(2)
    supervisor.change_state(Transition.TRANSITION_ACTIVATE)

    # Step 3: Run for 5 seconds then Deactivate
    time.sleep(5)
    supervisor.change_state(Transition.TRANSITION_DEACTIVATE)

    # Step 4: Cleanup
    time.sleep(2)
    supervisor.change_state(Transition.TRANSITION_CLEANUP)

    # Step 5: Shutdown  # This moves the node from 'Unconfigured' to 'Finalized'
    time.sleep(2)
    supervisor.get_logger().info('SUPERVISOR: Requesting Shutdown...')
    supervisor.change_state(Transition.TRANSITION_UNCONFIGURED_SHUTDOWN)
```

```
    supervisor.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

Part 4: Setup and Build

1. Open setup.py in the lifecycle_demo folder.
2. Add the entry points inside the console_scripts bracket so ROS2 can find your nodes:

```
Python
```

```
    entry_points={
        'console_scripts': [
            'talker = lifecycle_demo.lc_talker:main',
            'supervisor = lifecycle_demo.supervisor:main',
        ],
    },
```

3. Build the package (run from ~/ros2_ws):

```
cd ~/ros2_ws
colcon build --packages-select lifecycle_demo
source install/setup.bash
```

Part 5: Running the Activity

Terminal 1 (The Musician):

Run the lifecycle talker. Notice it prints "Node created", but **it does not publish "Hello World" yet.** It is waiting.

```
source ~/ros2_ws/install/setup.bash
```

```
ros2 run lifecycle_demo talker
```

Terminal 2 (The Conductor):

Run the supervisor. Watch the logs in Terminal 1 as the supervisor issues commands.

```
source ~/ros2_ws/install/setup.bash
ros2 run lifecycle_demo supervisor
```

What you should see:

1. **Terminal 1:** INSTRUCTOR: Configuring...
2. **Terminal 1:** INSTRUCTOR: Activating...
3. **Terminal 1:** Publishing: "Hello World: Lifecycle is Active!" (Repeats for 5 seconds)
4. **Terminal 1:** INSTRUCTOR: Deactivating... (Publishing stops)
5. **Terminal 1:** INSTRUCTOR: Cleaning up...
6. **Terminal 1:** INSTRUCTOR: Shutting down ... The node is now finalized.

On Terminal 2:

SUPERVISOR: Requesting transition 1...

SUPERVISOR: Transition Result: True

SUPERVISOR: Requesting transition 3...

SUPERVISOR: Transition Result: True

SUPERVISOR: Requesting transition 4...

SUPERVISOR: Transition Result: True

SUPERVISOR: Requesting transition 2...

SUPERVISOR: Transition Result: True

SUPERVISOR: Requesting Shutdown...

SUPERVISOR: Requesting transition 5...

SUPERVISOR: Transition Result: True

Discussion Questions for Students:

1. Why might this "wait to activate" behavior be useful for a robot with a laser that takes 10 seconds to warm up?
2. If the on_configure step failed (returned FAILURE), would the Supervisor be able to ACTIVATE the node?