Understanding ROS 2 Actions: The Fibonacci Example

In ROS 2, actions handle long-running tasks that need progress updates and can be cancelled. While services are for quick requests, actions are for complex operations that take time. Think of actions as asynchronous tasks with feedback - like starting a download that shows progress percentage.

The Fibonacci Analogy:

Imagine you're asking a mathematician to calculate Fibonacci numbers 🧮:

- You (Action Client): "Calculate Fibonacci sequence up to 8 numbers" → *Action Goal*
- Mathematician (Action Server): Starts calculating and sends updates:
  - "25% complete: [0, 1, 1]" → *Feedback*
  - "50% complete: [0, 1, 1, 2, 3]" → *Feedback*
  - "100% complete: [0, 1, 1, 2, 3, 5, 8, 13]" → *Result*
- You: Can cancel at any time: "Stop! That's enough!" → *Cancellation*

Action Definition:

```
bash

# Goal

int32 order

---

# Result

int32[] sequence

---

# Feedback

int32[] partial_sequence
```

What We're Building:

In this exercise, we'll create a Fibonacci Calculator Action with two nodes:

- `fibonacci_server`: The "math expert" that generates Fibonacci sequences with progress updates
- `fibonacci_client`: The "student" that requests sequences and monitors progress

Key Action Characteristics:

- ⏱️ Long-running: Tasks that take seconds to complete
- 📊 Progress feedback: Regular updates during execution
- 🛑 Cancellable: Can stop the task anytime
- 🎯 Goal-oriented: Clear start and end points
- 🔄 Asynchronous: Client doesn't block while waiting

This pattern is essential for robot navigation, complex computations, data processing, and any operation where you need to track progress and potentially cancel.

🚀 Simple Fibonacci Action Example

Step 1: Create Package

```
bash
```

```
cd ~/ros2_ws1/src
```

```
ros2 pkg create fibonacci_action --build-type ament_python --dependencies
rclpy example_interfaces
```

Step 2: Create Fibonacci Server

File: `~/ros2_ws1/src/fibonacci_action/fibonacci_action/fibonacci_server.py`

```
python
```

```python
#!/usr/bin/env python3
"""
Fibonacci Action Server
Generates Fibonacci sequences with progress feedback
"""

import rclpy
from rclpy.action import ActionServer
from rclpy.node import Node

# 🎯 Use built-in Fibonacci action
from example_interfaces.action import Fibonacci

class FibonacciServer(Node):
    def __init__(self):
        super().__init__('fibonacci_server')

        # 🎯 Create action server
        self._action_server = ActionServer(
            self,
            Fibonacci,           # Action type
            'fibonacci',         # Action name
            self.execute_callback
        )
```

```
        self.get_logger().info('🧮 Fibonacci Server Ready!')
        self.get_logger().info('📡 Action: /fibonacci')

    async def execute_callback(self, goal_handle):
        """
        Execute Fibonacci sequence generation
        """
        # 🎯 Get the goal (how many Fibonacci numbers to generate)
        order = goal_handle.request.order

        self.get_logger().info(f'🎯 Received goal: generate {order}
Fibonacci numbers')

        # Validate input
        if order <= 0:
            goal_handle.abort()
            self.get_logger().error('❌ Invalid order: must be
positive')
            result = Fibonacci.Result()
            result.sequence = []
            return result

        # Initialize Fibonacci sequence
        sequence = [0, 1]

        # 🎯 Send initial feedback
```

```
        feedback_msg = Fibonacci.Feedback()
        feedback_msg.partial_sequence = sequence.copy()
        goal_handle.publish_feedback(feedback_msg)

        self.get_logger().info(f'📊 Initial sequence: {sequence}')

        # 🎯 Generate sequence with progress updates
        for i in range(1, order):
            # 🛑 Check if goal was cancelled
            if goal_handle.is_cancel_requested:
                goal_handle.canceled()
                self.get_logger().info('❌ Fibonacci calculation
cancelled')
                result = Fibonacci.Result()
                result.sequence = sequence
                return result

            # Calculate next Fibonacci number
            next_number = sequence[i] + sequence[i-1]
            sequence.append(next_number)

            # 🎯 Publish feedback (partial sequence)
            feedback_msg.partial_sequence = sequence.copy()
            goal_handle.publish_feedback(feedback_msg)
```

```
            # Calculate progress percentage
            progress = (len(sequence) / (order + 1)) * 100

            self.get_logger().info(
                f'📊 Progress: {progress:.1f}% - '
                f'Generated {len(sequence)} numbers: {sequence}'
            )

            # 🕒 Simulate computation time (1 second per number)
            await rclpy.sleep(1.0)

        # 🎯 Goal completed successfully
        goal_handle.succeed()

        # Prepare result
        result = Fibonacci.Result()
        result.sequence = sequence

        self.get_logger().info(f'✅ Fibonacci completed: {sequence}')

        return result

def main():
    rclpy.init()
```

```
    try:
        fibonacci_server = FibonacciServer()
        rclpy.spin(fibonacci_server)

    except KeyboardInterrupt:
        print("\n🛑 Fibonacci Server stopped by user")
    except Exception as e:
        print(f"❌ Error in Fibonacci Server: {e}")
    finally:
        if rclpy.ok():
            rclpy.shutdown()
        print("✅ Fibonacci Server shutdown complete")

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

Step 3: Create Fibonacci Client

File: `~/ros2_ws1/src/fibonacci_action/fibonacci_action/fibonacci_client.py`

```
#!/usr/bin/env python3
"""
Fibonacci Action Client
Requests Fibonacci sequences and monitors progress
"""
```

```python
import rclpy
from rclpy.action import ActionClient
from rclpy.node import Node
import sys

# 🎯 Use built-in Fibonacci action
from example_interfaces.action import Fibonacci

class FibonacciClient(Node):
    def __init__(self):
        super().__init__('fibonacci_client')

        # 🎯 Create action client
        self._action_client = ActionClient(self, Fibonacci, 'fibonacci')

        self.get_logger().info('🎮 Fibonacci Client Ready!')
        self._result_received = False

    def send_goal(self, order):
        """
        Send Fibonacci calculation goal
        """
        self.get_logger().info(f'📤 Sending goal: generate {order}
Fibonacci numbers')
```

```
        # 🎯 Wait for server to be available
        if not self._action_client.wait_for_server(timeout_sec=5.0):
            self.get_logger().error('❌ Server not available after 5 seconds')
            self.shutdown()
            return

        # 🎯 Create goal message
        goal_msg = Fibonacci.Goal()
        goal_msg.order = order

        # 🎯 Send goal asynchronously with feedback callback
        future = self._action_client.send_goal_async(
            goal_msg,
            feedback_callback=self.feedback_callback
        )

        # 🎯 Set up response callback
        future.add_done_callback(self.goal_response_callback)

        return future

    def feedback_callback(self, feedback_msg):
        """
```

```
        Handle progress feedback from server
        """
        feedback = feedback_msg.feedback
        sequence = feedback.partial_sequence

        self.get_logger().info(
            f'📊 Feedback: {len(sequence)} numbers generated - {list(sequence)}'
        )

    def goal_response_callback(self, future):
        """
        Handle server's response to our goal
        """
        goal_handle = future.result()

        if not goal_handle.accepted:
            self.get_logger().error('❌ Goal rejected by server')
            self.shutdown()
            return

        self.get_logger().info('✅ Goal accepted! Calculation started...')

        # 🎯 Get result when calculation completes
        result_future = goal_handle.get_result_async()
```

```
        result_future.add_done_callback(self.get_result_callback)


    def get_result_callback(self, future):
        """
        Handle final result from server
        """
        try:
            result = future.result().result
            sequence = result.sequence

            self.get_logger().info('=' * 50)
            self.get_logger().info(f'🎯 FINAL RESULT: {list(sequence)}')
            self.get_logger().info(f'📊 Total numbers generated: {len(sequence)}')
            self.get_logger().info('=' * 50)

        except Exception as e:
            self.get_logger().error(f'💥 Error getting result: {e}')
        finally:
            self._result_received = True
            self.shutdown()

    def shutdown(self):
        """Safe shutdown"""
        self.destroy_node()
```

```
        if rclpy.ok():
            rclpy.shutdown()


def main():
    rclpy.init()


    try:
        # Create client
        fibonacci_client = FibonacciClient()


        # Get order from command line or use default
        if len(sys.argv) > 1:
            order = int(sys.argv[1])
        else:
            order = 5  # Default: generate 5 numbers


        # Validate input
        if order <= 0:
            print("❌ Error: Order must be a positive integer")
            return


        print(f"\n🧮 Requesting Fibonacci sequence up to {order}
numbers...")


        # Send goal
```

```
        fibonacci_client.send_goal(order)

        # 🎯 Keep client running until result is received
        # This is the key fix - don't shutdown until we have a result
        try:
            rclpy.spin(fibonacci_client)
        except KeyboardInterrupt:
            print("\n🛑 Fibonacci Client stopped by user")

    except ValueError:
        print("❌ Error: Please provide a valid integer for the Fibonacci order")
    except Exception as e:
        print(f"❌ Error in Fibonacci Client: {e}")
    finally:
        # Only shutdown if still initialized
        if rclpy.ok():
            rclpy.shutdown()
        print("✅ Fibonacci Client shutdown complete")

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

Step 4: Package Configuration

File: `~/ros2_ws1/src/fibonacci_action/package.xml`

```
xml
```

```xml
<?xml version="1.0"?>
<?xml-model href="http://download.ros.org/schema/package_format3.xsd"
schematypelocation="http://download.ros.org/schema/package_format3.xsd
package_format3.xsd"?>
<package format="3">
  <name>fibonacci_action</name>
  <version>0.0.0</version>
  <description>Fibonacci Action Example</description>
  <maintainer email="you@example.com">Your Name</maintainer>
  <license>Apache License 2.0</license>

  <depend>rclpy</depend>
  <depend>example_interfaces</depend>  <!-- 🎯 Provides Fibonacci action
-->

  <buildtool_depend>ament_python</buildtool_depend>

  <test_depend>ament_copyright</test_depend>
  <test_depend>ament_flake8</test_depend>
  <test_depend>ament_pep257</test_depend>
  <test_depend>python3-pytest</test_depend>

  <export>
    <build_type>ament_python</build_type>
  </export>

</package>
```

File: `~/ros2_ws1/src/fibonacci_action/setup.py`

```
python
```

```python
from setuptools import setup

package_name = 'fibonacci_action'

setup(
    name=package_name,
    version='0.0.0',
    packages=[package_name],
    data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='your_name',
    maintainer_email='your_email@example.com',
    description='Fibonacci Action Example',
    license='Apache License 2.0',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            'fibonacci_server = fibonacci_action.fibonacci_server:main',
            'fibonacci_client = fibonacci_action.fibonacci_client:main',
        ],
    },
)
```

Step 5: Build and Run

```
bash
```

```bash
cd ~/ros2_ws1

# Build the package
colcon build --packages-select fibonacci_action

# Source the workspace
source install/setup.bash
```

Step 6: Test the Action System

Terminal 1 - Start Server:

```bash
ros2 run fibonacci_action fibonacci_server
```

Output:

```text
🧮 Fibonacci Server Ready!
📡 Action: /fibonacci
```

Terminal 2 - Run Client:

```bash
ros2 run fibonacci_action fibonacci_client 6
```

Output:

```text
🎮 Fibonacci Client Ready!
📤 Sending goal: generate 6 Fibonacci numbers
✅ Goal accepted! Calculation started...
📊 Feedback: 3 numbers generated - [0, 1, 1]
📊 Feedback: 4 numbers generated - [0, 1, 1, 2]
📊 Feedback: 5 numbers generated - [0, 1, 1, 2, 3]
📊 Feedback: 6 numbers generated - [0, 1, 1, 2, 3, 5]
📊 Feedback: 7 numbers generated - [0, 1, 1, 2, 3, 5, 8]
==================================================
🎯 FINAL RESULT: [0, 1, 1, 2, 3, 5, 8]
📊 Total numbers generated: 7
==================================================
```

Terminal 3 - Test with ROS 2 CLI:

```bash
# List all actions
ros2 action list

# Get action info
```

```
ros2 action info /fibonacci

# Send goal directly with feedback
ros2 action send_goal /fibonacci example_interfaces/action/Fibonacci
"{order: 4}" --feedback

# Test cancellation (run this quickly after sending goal)
ros2 action send_goal /fibonacci example_interfaces/action/Fibonacci
"{order: 10}" --feedback

# Then press Ctrl+C to cancel
```