Apex.AI

launch_testing in ROS 2

Presenter: Sumanth Nirmal
Authors: Peter Baughman, William Woodall, Michel Hidalgo

© 2019 Apex.AI, Inc.
Agenda

- Integration testing
- rostest
- ROS 2 launch
- rostest improvements
- launch_test
- launch_test example
- launch_test demo
Motivation for Integration Testing

Integration testing is where software modules are integrated logically and tested as a group
- Clear API boundary between processes being tested
- Integration tests checks the real program flow
- Integration tests check the real user interactions
- Unit tests often uses bespoke configurations and can miss issues with real configurations
- Unit tests are often single-threaded, and don't expose complex race conditions
- Performance testing

Lack of Integration tests!
2 Unit tests, 0 Integration tests
● It is an integration test suite based on `roslaunch`
● Has `<test>` tag in the xml file, which specifies the test nodes to run
● rotest’s `roscore` is restarted for each test
● By default rotest uses random ports, so many rotest’s can run in parallel and in isolation
● Launches all of the processes from a launch (xml) file, then waits for the test process to finish
● The final exit code is based on the exit code of the test process
launch and launch_ros in ROS 2

- *launch* is a ROS-agnostic tool to launch processes
- *launch_ros* is a tool to launch ROS nodes
- ROS 2 launch is exposed as a Python API
- ROS 2 launch API is extendable and supports adding custom actions and events
- ROS masters are no longer required for nodes to communicate
- *launch* can use events for feedback and have actions that create other actions
- *launch* descriptions can be introspected

New Challenges:
- ROS 2 launch descriptions can be arbitrarily complex and they themselves need to be tested
Rough Edges In rotest

- Tests might run before the processes under test were ready
  - This results in a test failure, which might be hard to reproduce
- Processes under test might crash, and tests would not notice this crash
  - Results in a test failure, which would be difficult to reproduce
- Debug output like `stdout` or `stderr` from tests might be hard to find
  - This makes it difficult to analyse the `stdout` or `stderr` streams to align output with the test cases
Improvements on rostest Functionality

• What if there was a way for the test author to explicitly coordinate when the test runs with the processes under test?
  ➢ _launch_testing_ has a _ReadyToTest_() action that coordinates test start with the rest of the launch

• What if tests could access process information like the exit codes of launched processes?
  ➢ _launch_testing_ has a _ProcInfoHandler_ object, which has exit code information for processes that were launched

• What if tests could access process information like _stdout_ or _stderr_ of launched processes?
  ➢ _launch_testing_ has an _IOHandler_ object, which has _stdout/stderr_ information for processes that were launched

• What if we could automatically generate test cases for the launched processes?
  ➢ Ability to introspect the launch description makes it easy to programmatically generate test cases
• **launch_testing** is a framework for integration testing based on the ROS 2 launch system
• The exit codes, `stdout` and `stderr` of all processes launched are available to the tests
• The command-line used to launch the processes are available to the tests
• Tests can run concurrently with the launched processes, or can run after the launched processes are shut down
• Output looks like regular unit-test output; each test case gets a line
• Parametrize the launch description
• **launch_testing_ros** can automatically generate a unique `ROS_DOMAIN_ID` to isolate tests from one-another to be able to run the tests in parallel and in isolation
Timeline of a launch_test Command

“Active” tests. May interact with running processes via ROS or other means, or just observe

“Post-Shutdown” tests. Can access recorded stderr, stdout, exit codes, or other data recorded during the test

Start Launch Description  Start Tests  Tests End  Shutdown

Start post-shutdown Tests  Tests End
Using the launch_testing API

- Auto-generate test cases from documentation
  - Check that launch files in documentation continue to work as documented
  - Check that process arguments and values continue to work as documented
  - Check that documented topics are present in a launched system
  - Check that output generated by processes matches what's in the documentation
- Auto-generate tests that always run for every process tested automatically
  - Check output for errors
  - Check exit codes of all processes
- Automatically remap topics and insert nodes that fuzz the data before republishing, or drop messages
Possible Future Development

- Add ROS 1 style test processes
  - Run `gtest` in a separate process
  - Run `pytest` in a separate process
  - Run an arbitrary process that can pass/fail based on an exit code
- Include tests in the launch description
  - These tests could easily introspect the launch context to help test the launch description itself
- `rosbag` integration to record data for analysis in post-shutdown tests
def generate_test_description():
    return launch.LaunchDescription([launch.actions.ExecuteProcess(
            cmd=[path_to_process],
        ),
    ),

    # Start tests right away - no need to wait for anything in this example.
    # In a more complicated launch description, we might want this action happen
    # once some process starts or once some other event happens
    launch_testing.actions.ReadyToTest()
# Part 2/3

# Active test

# These tests will run concurrently with the test in process. After all these tests are done,
# the launch system will shut down the processes that it started up

class TestGoodProcess(unittest.TestCase):

def test_count_to_four(self, proc_info, proc_output):
    # This will match stdout from any process. In this example there is only one process
    # running
    proc_output.assertWaitFor('Loop 1', timeout=10)
    proc_output.assertWaitFor('Loop 2', timeout=10)
    proc_output.assertWaitFor('Loop 3', timeout=10)
    proc_output.assertWaitFor('Loop 4', timeout=10)
These tests will run after with the test in process is shut-down.

```python
@launch_testing.post_shutdown_test()
class TestProcessOutput(unittest.TestCase):
    def test_exit_code(self, proc_info, proc_output):
        # Check that all processes in the launch (in this case, there's just one) exit
        # with code 0
        launch_testing.asserts.assertExitCodes(proc_info)

    def test_out_of_order(self, proc_info, proc_output):
        # This demonstrates that we notice out-of-order IO
        with self.assertRaisesRegexp(AssertionError, 'Loop 2 not found'):
            with assertSequentialStdout(proc_output, <process>) as cm:
                cm.assertInStdout('Loop 1')
                cm.assertInStdout('Loop 3')
                cm.assertInStdout('Loop 2')
```

Inherits from `unittest.TestCase` Decorated with `post_shutdown_test` descriptor

Asserts that the specified process exited with a particular exit code.

Asserts that stdout was seen in a particular order.

Asserts that a message is found in the `stdout` of the process.
Using CMake to run launch_test

- Declare a dependency on `launch_testing_ament_cmake` in `package.xml`

  `<test_depend>launch_testing_ament_cmake</test_depend>`

- In `CMakeLists.txt` file, add

  ```
  find_package(launch_testing_ament_cmake)
  add_launch_test(test/name_of_test.test.py)
  ```

- Optionally, arguments can also be passed to the tests:

  ```
  add_launch_test(
    test/test_with_args.test.py
    ARGS "arg1:=foo"
  )
  ```
Live Demo: Launch Test

Quick demonstration of launch test
Documentation and the ROS Buildfarm

Example packages using `launch_testing` in ROS Buildfarm:

2. https://github.com/ros2/rcl/blob/dashing/rcl/test/rcl/test_two_executables.py.in
3. https://github.com/ros2/demos/blob/dashing/demo_nodes_cpp/test/test_executablesutorial.py.in