ROS Software Quality Assessment through Static Code Analysis and Property-Based Testing

ROS-Industrial Conference 2018

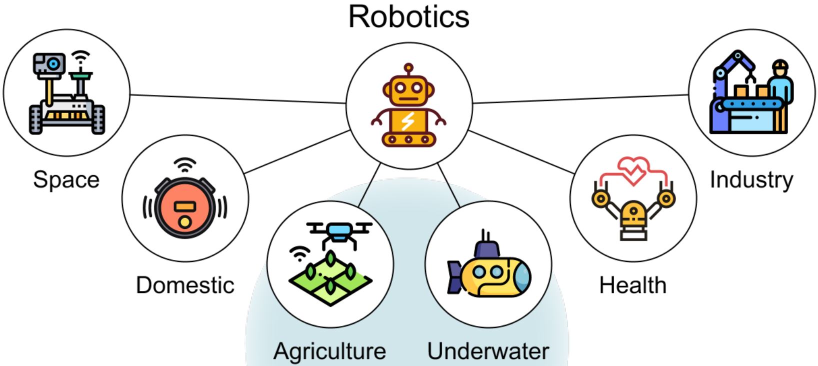
André Santos Alcino Cunha Nuno Macedo

INESC TEC & University of Minho, Portugal

December 12, 2018

Software Quality Matters

Many robotic applications can be considered safety-critical systems.



Detecting defects early reduces costs and development time.

Software Quality Matters

There are many excellent static analysis tools already available.

HelixQAC

coverity®
A Synopsys Company

Cppcheck
A tool for static C/C++ code analysis

Semmle™

frama C

klocwork®
a Rogue Wave Company

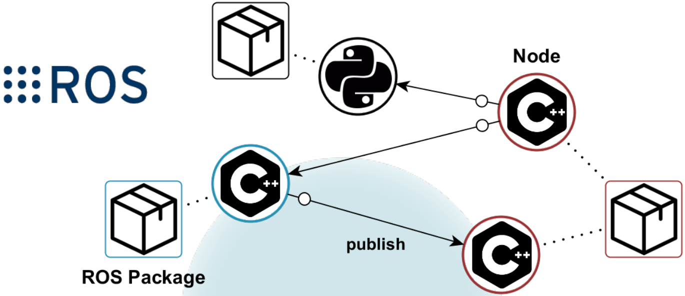
sonarqube

PARASOFT

But none provide a ROS-specific analysis.

The ROS Computation Graph

The network of nodes and resources in a ROS system is called the Computation Graph.

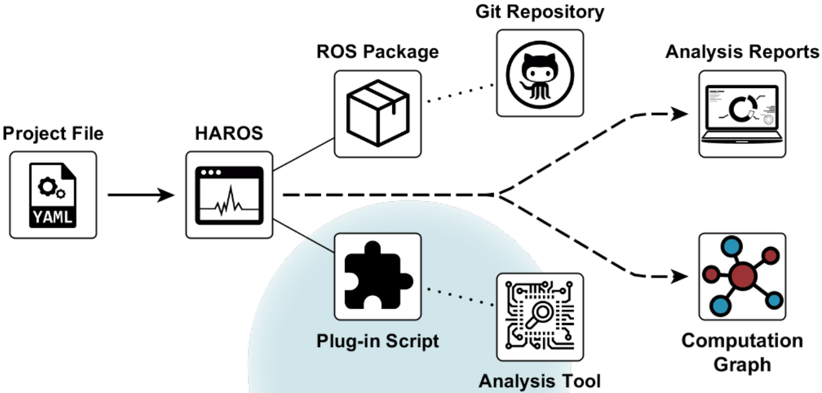


Expressing properties at the level of the Computation Graph is more intuitive than at the source code level.

How can we check that such properties hold?

The HAROS Framework

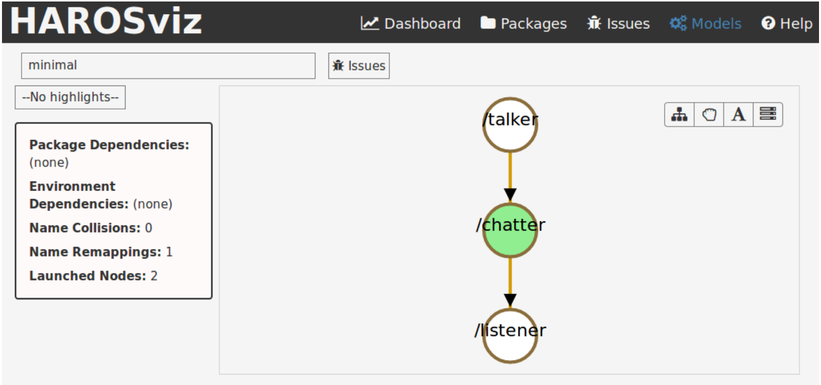
HAROS (High-Assurance ROS) is a static analysis framework for ROS.



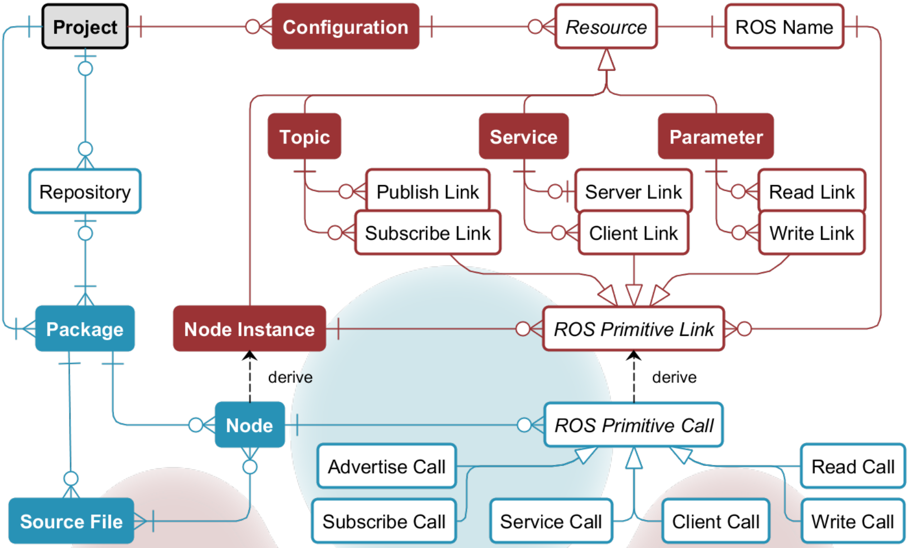
[New in v3.0]
HAROS is capable of reverse-engineering the Computation Graph.

The HAROS Framework

HAROS provides visualisation of the extracted Graph.



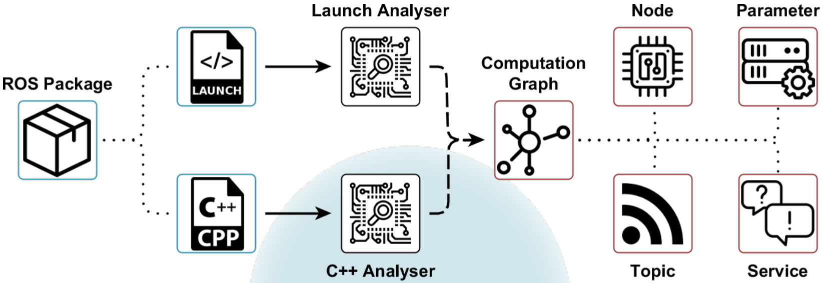
A Metamodel for ROS



HASLab HIGH-ASSURANCE SOFTWARE LABORATORY

Model Extraction from Source Code

Extracting such a model from source code requires parsing C++ and launch files.

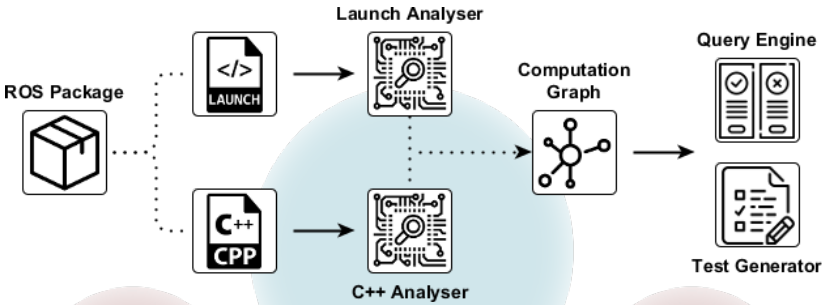


Conditionals and unknown values are also recorded during this process.

Users can resolve unknown values by providing extraction hints.

Applications of the Computation Graph

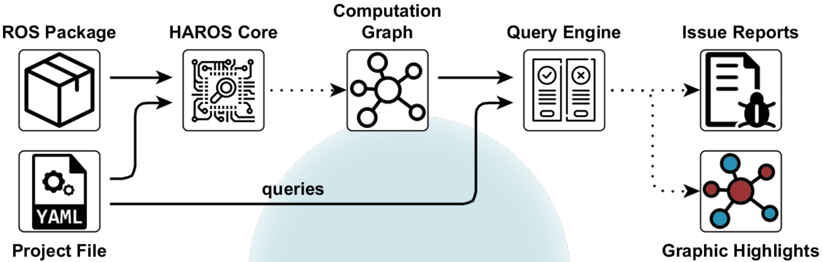
A reconstructed Computation Graph enables the verification of architecture-related properties through various methods.



HASLab
HIGH-ASSURANCE SOFTWARE LABORATORY

Querying the Computation Graph

User-defined queries are a way to implement simple checks over the structure of the extracted graph.



Queries are written with PyFlwor (/timtadh/pyflwor).

Querying the Computation Graph

At most, one publisher per topic:

```
topics[len(self.publishers) > 1]
```

Parameters are read-only:

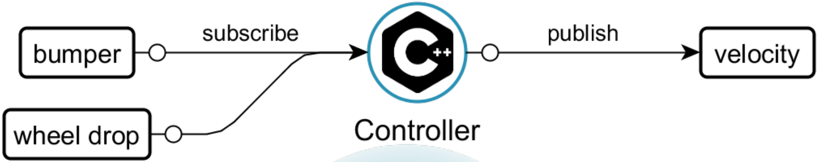
```
parameters[len(self.writes) > 0]
```

Type checking for topics:

```
for t1 in <nodes/publishers | nodes/subscribers>,
    t2 in <nodes/publishers | nodes/subscribers>
where t1.topic_name == t2.topic_name
      and t1.type != t2.type
return t1, t2
```

Testing in ROS

Some properties cannot (or are hard to) be checked at static time.



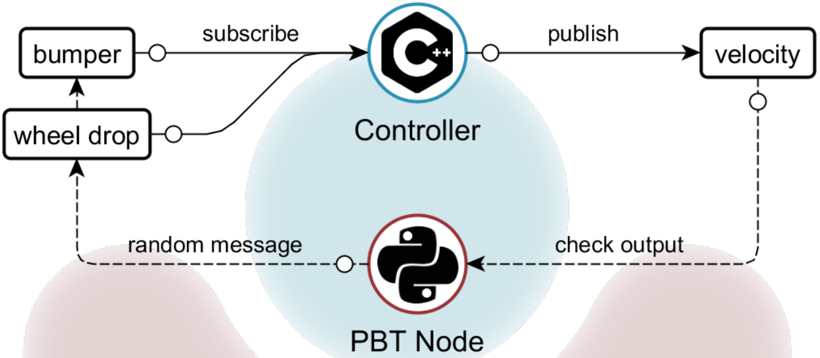
If a "*stop*" message is published, then a "*bumper pressed*" message was received before.

Testing node interfaces is often done with manual test cases.

Property-based Testing for ROS

Testing component interfaces in terms of their properties is one of the scenarios where Property-based Testing shines.

The same principle can be applied to a ROS configuration. The major challenge is asynchronous communication.



Property-based Testing for ROS

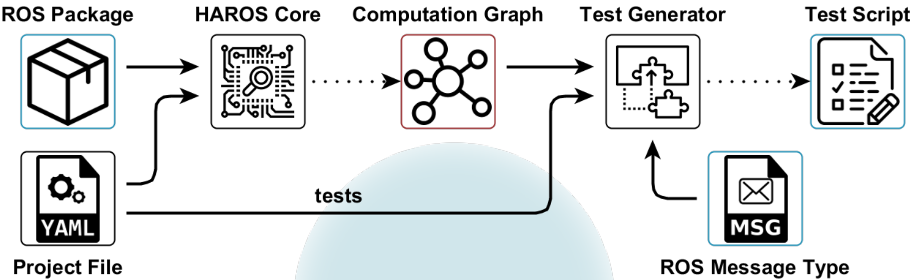
The setup for a PBT ROS node is always the same.

1. Initialise ROS node.
2. Advertise a number of topics (test input data).
3. Subscribe a number of topics (test output data).
4. Publish random messages, in a random sequence.
5. Check properties when messages are received or when a time interval has elapsed.

The Computation Graph and its properties are the variables.
Thus, we can implement a test script generator.

Property-based Testing for ROS

Our test generator takes in a ROS Computation Graph and produces a template test node.



We used Hypothesis (/HypothesisWorks/hypothesis) as the base Property-based Testing library.

Property-based Testing for ROS

By default, the generated script just tests for crashes.

It contains a blank internal state, for users to specify properties.

```
class InternalState(object):
    def __init__(self):
        self.on_setup()

    def on_setup(self):
        pass

    def on_pub__events__bumper(self, event):
        pass

    def on_sub__cmd_vel(self, event):
        pass
```

Property-based Testing for ROS

Stop message published ⇒ a bumper is pressed.

```
class InternalState(object):
  def __init__(self):
    self.on_setup()

  def on_setup(self):
    self.bumper_pressed = False

  def on_pub__events__bumper(self, event):
    self.bumper_pressed = \
        event.msg.state == BumperEvent.PRESSED

  def on_sub__cmd_vel(self, event):
    if event.msg.linear.x == 0:
      assert self.bumper_pressed
```

Property-based Testing for ROS

When a counterexample is found, a trace is produced.

```
FAILED (failures=1)
=======================================================
state = RosRandomTester()
state.pub__events__wheel_drop(msg={wheel: 1, state: 1})
state.spin()
state.teardown()
-------------------------------------------------------
Time spent on testing  (s): 0.576339435
Time spent on sleeping (s): 5.8
Time spent setting up  (s): 57.640097381
```

Sneak Peek: A Property Specification Language

The language should be able to express:

- constraints over message fields (e.g. enumerations, ranges);

```
publish(c, cmd_vel) where c.linear.x in -0.5 to 1.5
```

- node publication rates and timeouts;

```
publish(c, cmd_vel) at 10 hz
```

- reactive behaviours (e.g. receive X leads to publish Y).

```
receive(e, events/bumper) where e.state = PRESSED
leads to publish(c, cmd_vel)
within 0.1 s where c.linear.x == 0
```

This language can be used to generate property-based tests, runtime monitors, node templates, documentation ...

Final Remarks

Summary

- Static analysis has the potential to find defects early on.
- Common static analysis tools are not ROS-specific.
- HAROS can statically extract the Computation Graph . . .
  - to enable static analysis with queries;
  - to generate property-based tests.

Near future goals

- Apply the analysis to industrial robots at INESC TEC.
- Finish the property specification language.
- Enhance the model extraction capabilities of HAROS.

Questions?

Demonstration video at
https://youtu.be/vuDxybomXd4

IROS and A-TEST papers at
haslab.uminho.pt/afsantos/publications

HASLab
HIGH-ASSURANCE
SOFTWARE LABORATORY

This work is financed by the ERDF – European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation - COMPETE 2020 Programme and by National Funds through the Portuguese funding agency, FCT – Fundação para a Ciência e a Tecnologia within project PTDC/CCI-INF/29583/2017 (POCI-01-0145-FEDER-029583).

COMPETE 2020 | PORTUGAL 2020 | UNIÃO EUROPEIA Fundo Europeu de Desenvolvimento Regional | FCT Fundação para a Ciência e a Tecnologia