Improving the Quality of ROS Applications with HAROS

Tutorial at IROS’21

Nuno Macedo and André Santos

1 - Introduction

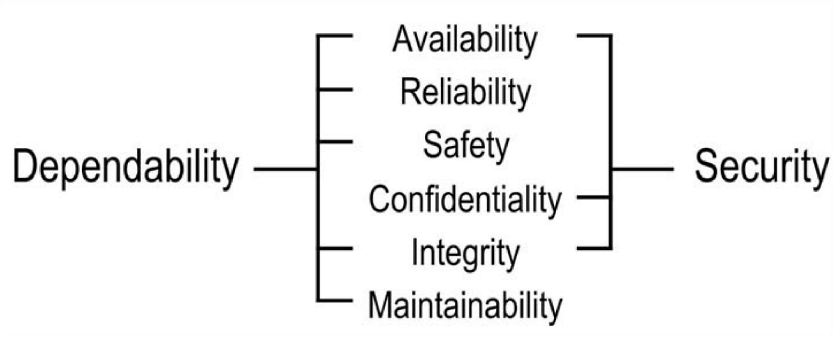
Challenge: Dependable Robotic Software

- Software-based robots are being deployed in safety critical contexts
- Complex systems, with several communicating components
    - Heterogenous, configurable, third-party components, ...
- Middlewares have been proposed to help building modern robots
    - **ROS** has emerged and the most popular, used in industrial contexts

How to guarantee that a ROS-based robot effectively acts as expected?

Quality Assurance in the SE community

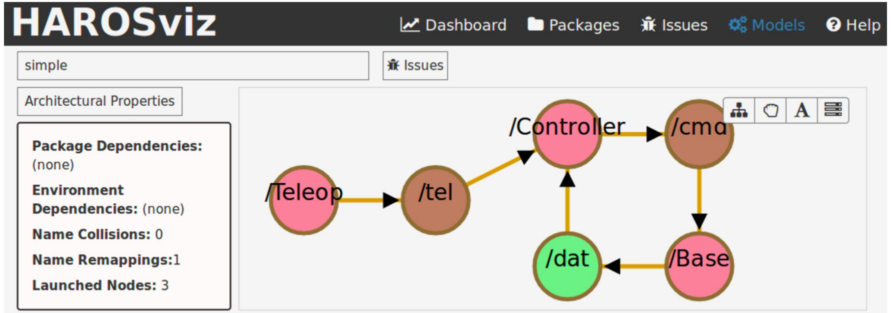
- Software engineers have long dealt with safety critical systems
- Several mechanisms have been developed for software quality control
  - static vs dynamic
  - automatic vs semi-automatic
  - design vs implementation
- Most techniques require user input
  - more advanced techniques may require formal methods expertise



Avižienis et al., 2004

High-Assurance ROS platform

- **HAROS** aims to bring SE techniques closer to roboticists
- Ecosystem aimed at ROS developers, minimal user input
- Automates analysis tasks and provides unified interface
- SE techniques wrapped in plug-ins, mostly opaque to end users



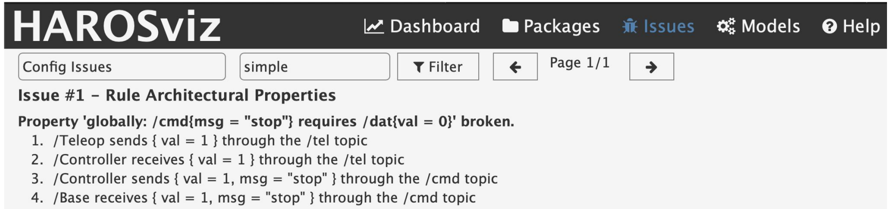
Package Dependencies: (none)

Environment Dependencies: (none)

Name Collisions: 0

Name Remappings:1

Launched Nodes: 3

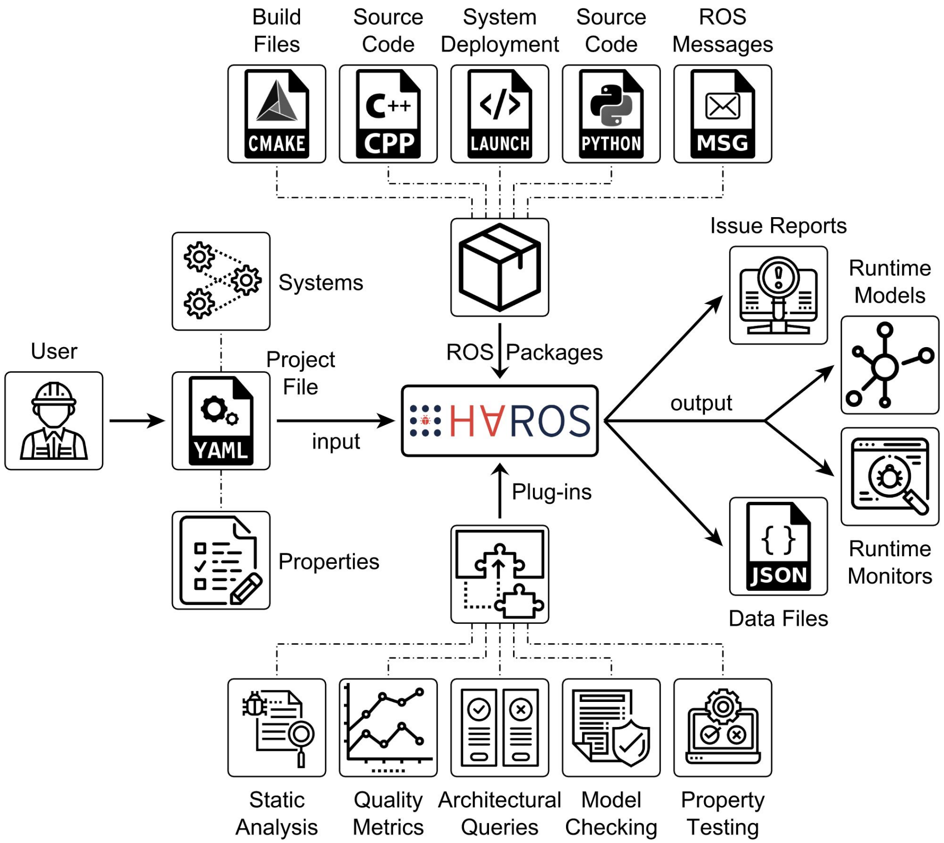


Issue #1 – Rule Architectural Properties

Property 'globally: /cmd{msg = "stop"} requires /dat{val = 0}' broken.

1. /Teleop sends { val = 1 } through the /tel topic
2. /Controller receives { val = 1 } through the /tel topic
3. /Controller sends { val = 1, msg = "stop" } through the /cmd topic
4. /Base receives { val = 1, msg = "stop" } through the /cmd topic

HAROS Overview



HAROS in the Wild

Community adoption

- ROS-Industrial

https://rosindustrial.org/

- ROS Quality Assurance Working Group

https://discourse.ros.org/c/quality/

Use cases

- Fraunhofer IPA Care-O-bot

- INESC TEC AgRob and FASTEN

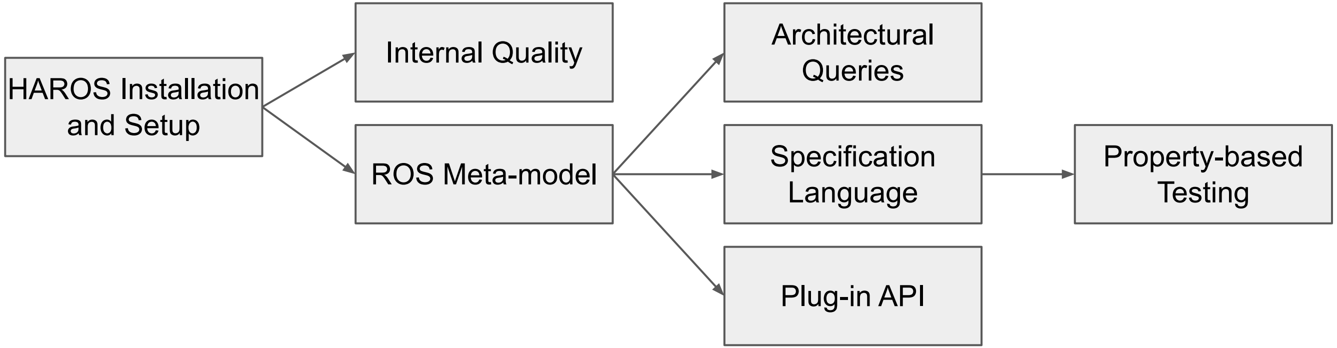
Goal of the Tutorial

Development of high-assurance robotic software with HAROS

- Brief introduction to quality assurance in software engineering
- Using HAROS to employ such techniques over ROS software

Outline of the Tutorial

- HAROS Installation and Setup
- Internal code quality
- ROS meta-model and its extraction
- Analysis of system architectures with architectural queries
- HAROS specification language
- Analysis of system behaviour with property-based testing
- HAROS plug-in API



Team

Nuno Macedo

Assistant professor, FEUP & INESC TEC

Experience

- teaching SE and formal methods
- developing lightweight formal techniques
- application to cyber-physical systems

André Santos

Research scientist, VORTEX CoLab

Experience

- developing QA techniques for robotic software
- application to ROS applications
- HAROS developer/maintainer

HAROS Quickstart

- HAROS ready Docker

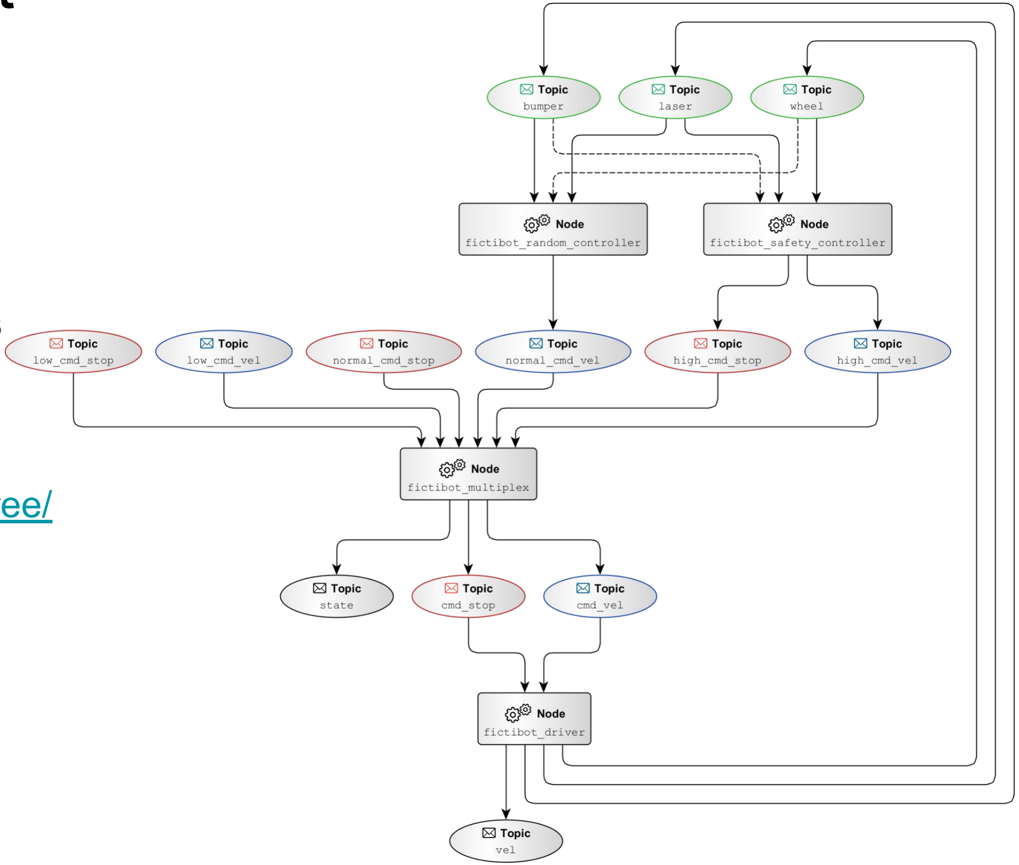
  https://github.com/git-afsantos/haros_tutorials/tree/master/docker

  - Build and run Docker
  - Compile example projects (Fictibot)
  - Run HAROS example scripts
- Installation demo

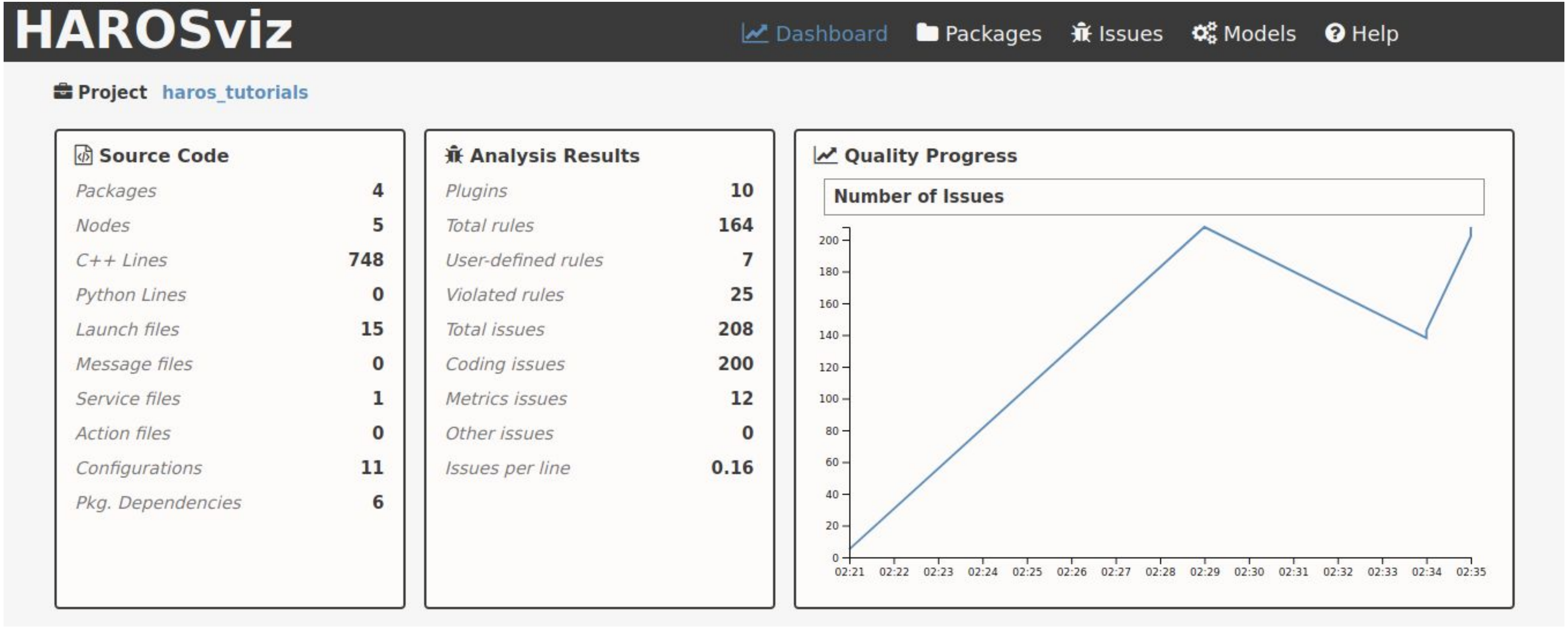
  https://youtu.be/c0LbC_D7nD8

Running Example: Fictibot

- Typical mobile robot base, inspired by TurtleBot2
- Few incorporated sensors and actuators
- Issues introduced to exercise the various HAROS functionalities
- Documentation: github.com/git-afsantos/haros_tutorials/tree/master/docs



HAROS Web Interface



Project haros_tutorials

Source Code

Packages	**4**
Nodes	**5**
C++ Lines	**748**
Python Lines	**0**
Launch files	**15**
Message files	**0**
Service files	**1**
Action files	**0**
Configurations	**11**
Pkg. Dependencies	**6**

Analysis Results

Plugins	**10**
Total rules	**164**
User-defined rules	**7**
Violated rules	**25**
Total issues	**208**
Coding issues	**200**
Metrics issues	**12**
Other issues	**0**
Issues per line	**0.16**

Quality Progress

Additional Resources

- Tutorial webpage, https://haslab.github.io/SAFER/iros21-tutorial
- Exercises and material, https://github.com/git-afsantos/haros_tutorials
- HAROS webpage, https://github.com/git-afsantos/haros/
- Demo videos, https://youtube.com/playlist?list=PLrXxXaugT0cwVhjhlnxY6DU0_WYPLEmgq
- A. Santos, A. Cunha, N. Macedo: **The High-Assurance ROS Framework**. RoSE@ICSE 2021: 37-40, https://doi.org/10.1109/RoSE52553.2021.00013
- A. Santos: **Safety Verification for ROS Applications.** PhD Thesis. University of Minho, Braga, Portugal, https://git-afsantos.github.io/publication/phd-thesis

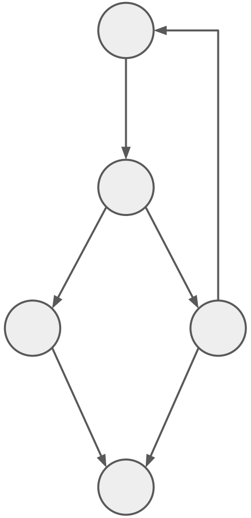
2 - Internal Code Quality

Internal Code Quality

- Directly impacts *maintainability*
  - How easy is it to understand and change the code?
  - Indirectly affects many other quality properties
- Can usually be detected through automatic *static code analysis*:
  - Quality metrics (complexity, modularity, …)
  - Coding styles (guidelines, standards, …)

Quality Metrics

- Measure characteristics of the source code
- Often related with complexity and modularity:
  - Lines of Code (LoC)
  - Cyclomatic complexity
  - Coupling
  - ...
- Violations occur when thresholds are met



Cyclomatic complexity of 3

Coding Styles

- Guidelines to improve readability and uniformity
  - Indentation
  - Naming conventions
  - ...
- Stricter standards forbid error-prone constructs
  - Always explicitly declare integer size
- May also impose quality metrics thresholds

```
int main(int argc, char **argv) {
    ros::init(argc, argv, "listener");
    ros::NodeHandle n;
    ...
    ros::spin();
    return 0;
}
```

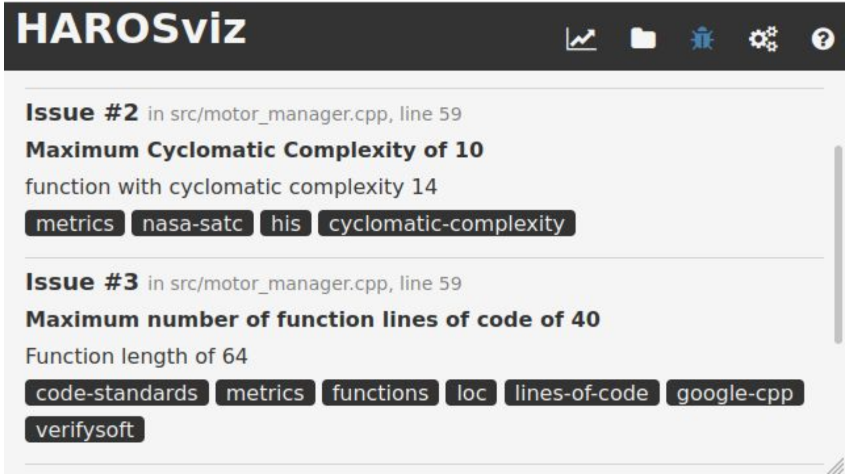
Violations to MISRA and ROS guidelines

HAROS Internal Quality Plug-ins

- Wrappers for existing static code analysis tools
    - `cccc` (C++)
    - `ccd` (C++)
    - `cppcheck` (C++)
    - `cpplint` (C++)
    - `pylint` (Python)
    - `lizard` (C++/Python)
    - `radon` (Python)
- Additional combined metrics on top
    - Maintainability Index (MI) calculator

HAROS Internal Quality Plug-ins

- Issues report violations of code rules and metrics exceeding thresholds
- Trace back to source code locations
- Tags allow finer inspection (e.g., kind of metric or related standards)
- Project can be configured to ignore full plug-ins or certain tags altogether



Issue #2 in src/motor_manager.cpp, line 59

Maximum Cyclomatic Complexity of 10

function with cyclomatic complexity 14

metrics nasa-satc his cyclomatic-complexity

Issue #3 in src/motor_manager.cpp, line 59

Maximum number of function lines of code of 40

Function length of 64

code-standards metrics functions loc lines-of-code google-cpp verifysoft

Hands-on Exercises

- Follow the link for exercises over Fictibot

  github.com/git-afsantos/haros_tutorials/tree/master/exercises/sec2-code-quality

- Identify and fix some code quality issues
- Demo and proposed solution

  https://youtu.be/xvoOMHa8RMw

Additional Resources

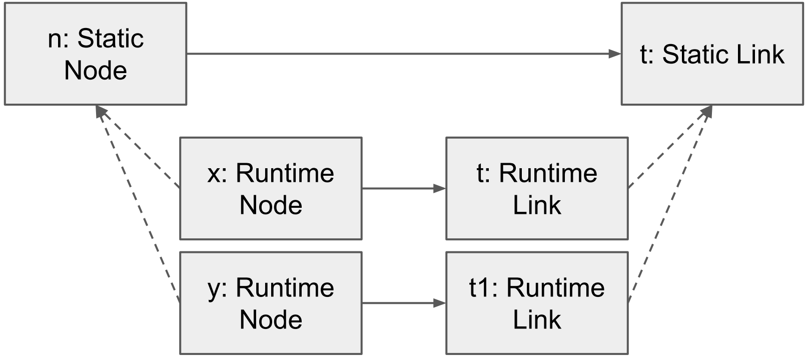
- A. Santos, A. Cunha, N. Macedo, C. Lourenço: **A framework for quality assessment of ROS repositories**. IROS 2016: 4491-4496, https://doi.org/10.1109/IROS.2016.7759661
- T. Neto, R. Arrais, A. Sousa, A. Santos, G. Veiga: **Applying Software Static Analysis to ROS: The Case Study of the FASTEN European Project**. ROBOT (1) 2019: 632-644, https://doi.org/10.1007/978-3-030-35990-4_51
- ROS style guide, http://wiki.ros.org/CppStyleGuide
- ROS code quality, http://wiki.ros.org/code_quality

3 - Analysis of System Architectures

ROS Architectural Meta-model

- Besides source code, ROS architectural models are also passed to plug-ins
- Not explicit in the code: extracted by HAROS through static analysis
- Two levels:
  - Compile-time artifacts, e.g., a programmed node
  - Runtime artifacts, e.g., a launched node

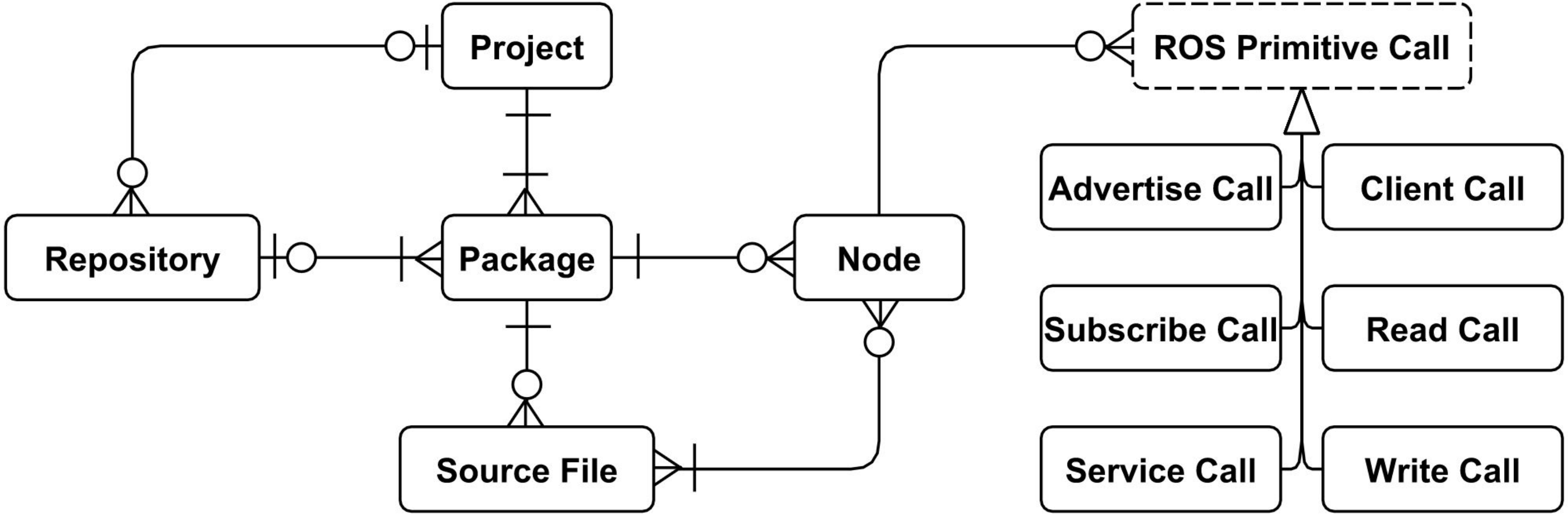
```
int main (...) {
        ros::init(..., ..., "n");
        ros::NodeHandle n;
        n.advertise<...>("/t", ...);
}
```



```
<launch>
    <node name="x" type="n" />
    <node name="y" type="n">
        <remap from="t" to="t1" />
    </node>
</launch>
```

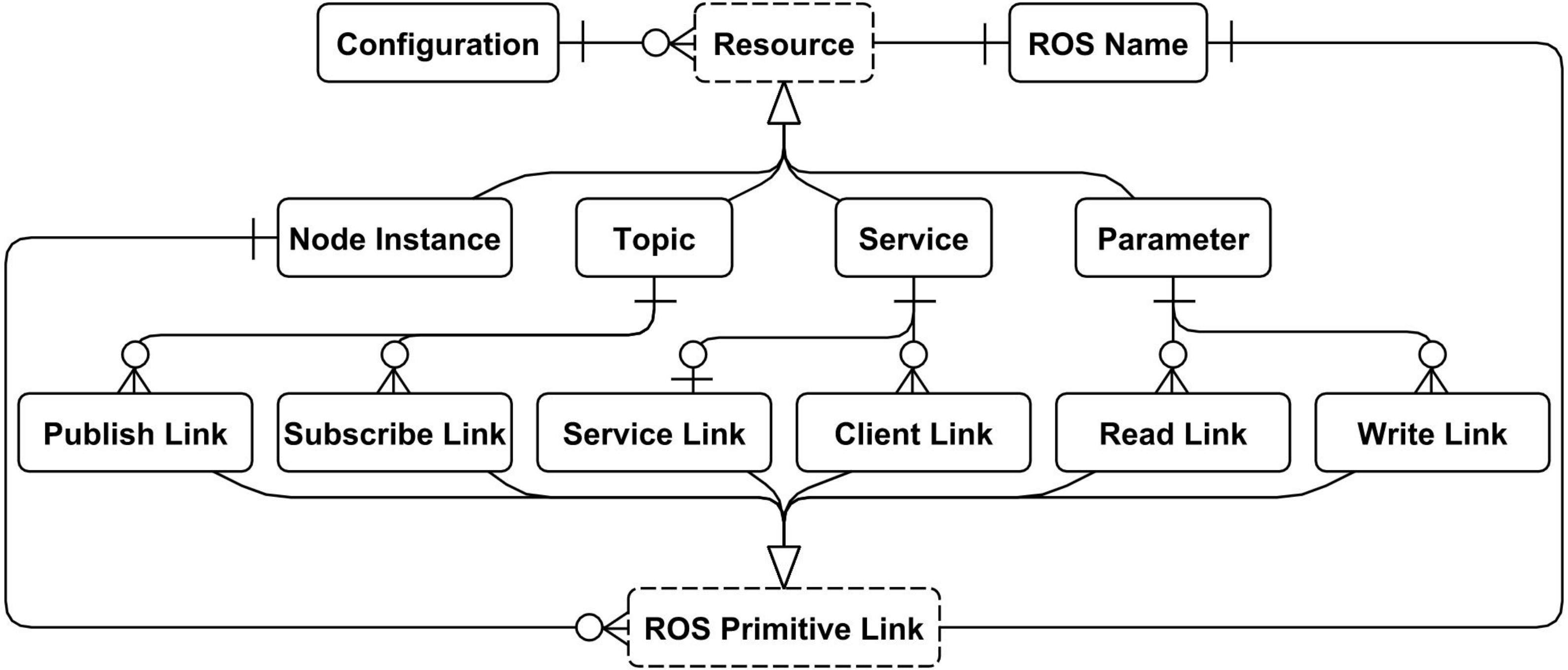
ROS Meta-model: Compile-time

- Project-oriented view (source code)



ROS Meta-model: Run-time

- Configuration-oriented view (computation graph)

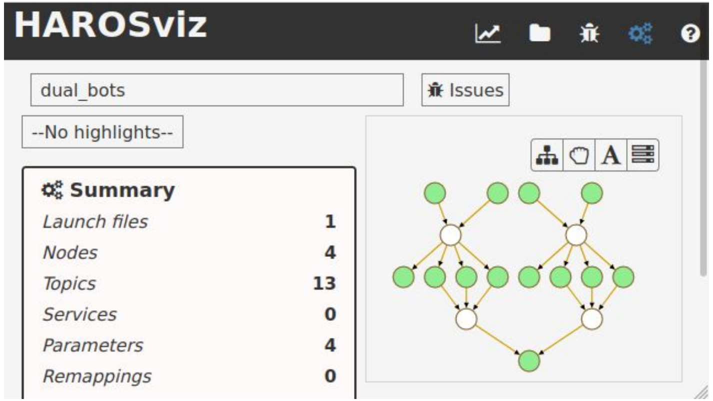


Extraction Process

- Model obtained *statically*, without running the program
    - Relies on source code static analysis techniques
- Source files for static-time perspective
- Launch files for run-time perspective (possibly multiple configurations)
- Optional elements have presence conditions registered
- Possibly incomplete process: users may provide *hints* to fill the gaps

HAROS Architectural Visualizer

- Allows the inspection of the different configurations specified for the project
- Traceability to static-time resources
- Conditional elements identified (and conditions)
- Can be used to explore runtime issues reported by plugins (see next session)

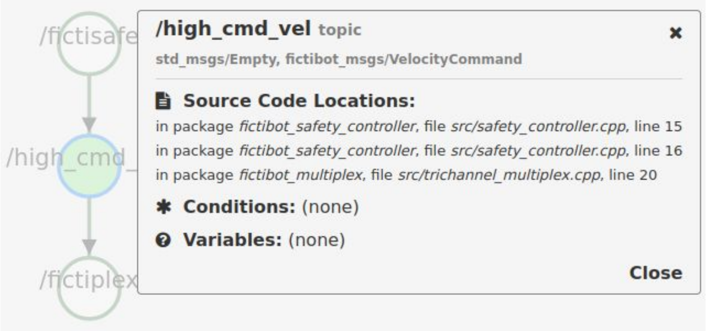


dual_bots

--No highlights--

Summary

Launch files	1
Nodes	4
Topics	13
Services	0
Parameters	4
Remappings	0



/high_cmd_vel topic ×

std_msgs/Empty, fictibot_msgs/VelocityCommand

Source Code Locations:

in package *fictibot_safety_controller*, file *src/safety_controller.cpp*, line 15
in package *fictibot_safety_controller*, file *src/safety_controller.cpp*, line 16
in package *fictibot_multiplex*, file *src/trichannel_multiplex.cpp*, line 20

Conditions: (none)

Variables: (none)

Close

Architectural Styles

- Likewise coding styles, *architectural styles* affect several quality metrics
  - Monolithic nodes vs many single-responsibility nodes
  - Use of namespaces vs single namespace
  - Nodes vs nodelets
  - etc.
- Less prone to automatic static analysis: architectures are not explicitly defined

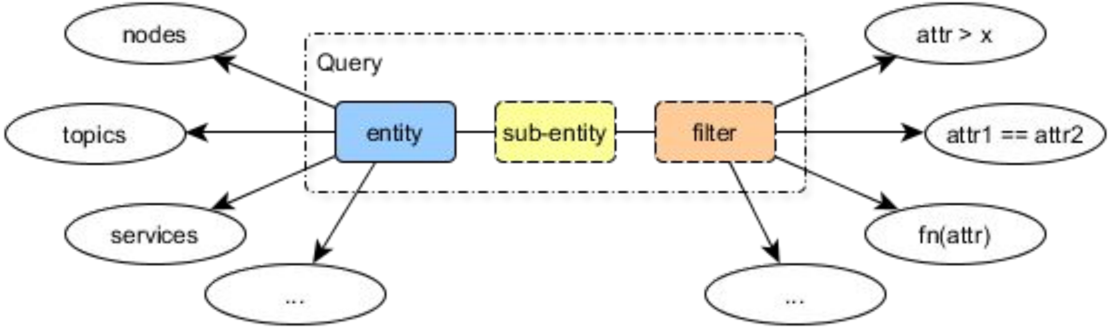
HAROS Architectural Query Plug-in

- Specification of architectural patterns over computation graphs
- Acts on the architectural models automatically extracted by HAROS
- Some built-in patterns:
    - Only 1 publisher per topic
    - Publisher and subscriber message types match
    - ...
- Provides *query language* for user-defined patterns

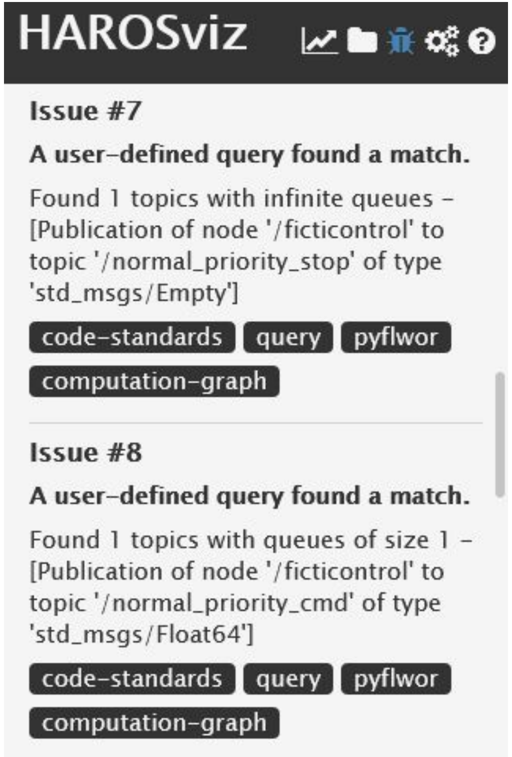
HAROS Architectural Query Language

- Queries follow the syntax of the PyFLWOR library, over the entities of the HAROS metamodel.
- Basic structure of a query:

```
nodes/publishers[len(self.queue_size) < 10]
```



HAROS Architectural Issues



Issue #7

A user-defined query found a match.

Found 1 topics with infinite queues - [Publication of node '/ficticontrol' to topic '/normal_priority_stop' of type 'std_msgs/Empty']

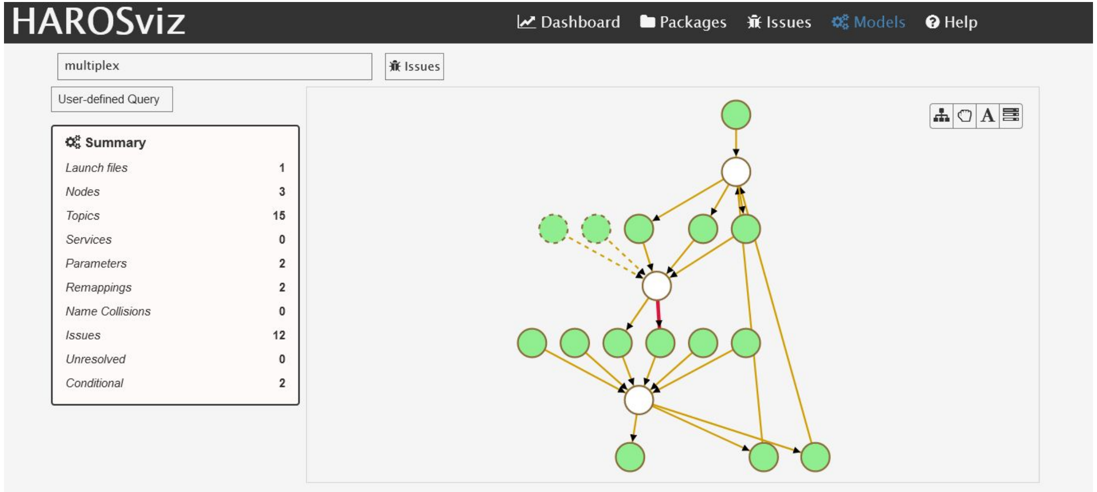
code-standards query pyflwor computation-graph

Issue #8

A user-defined query found a match.

Found 1 topics with queues of size 1 - [Publication of node '/ficticontrol' to topic '/normal_priority_cmd' of type 'std_msgs/Float64']

code-standards query pyflwor computation-graph



Summary

Launch files	1
Nodes	3
Topics	15
Services	0
Parameters	2
Remappings	2
Name Collisions	0
Issues	12
Unresolved	0
Conditional	2

Hands-on Exercises

- Follow the link for exercises over Fictibot

  github.com/git-afsantos/haros_tutorials/tree/master/exercises/sec3-architecture

- Explore the extracted architecture, fix issues and define architectural patterns
- Demo and proposed solution

  https://youtu.be/kD8chgLZ4yE

Additional Resources

- I. Malavolta, G. A. Lewis, B. R. Schmerl, P. Lago, D. Garlan: **How do you architect your robots?: state of the practice and guidelines for ROS-based systems**. ICSE (SEIP) 2020: 31-40, https://doi.org/10.1145/3377813.3381358
- A. Santos, A. Cunha, N. Macedo: **Static-Time Extraction and Analysis of the ROS Computation Graph**. IRC 2019: 62-69, https://doi.org/10.1109/IRC.2019.00018
- Hint language reference, https://github.com/git-afsantos/haros/blob/master/docs/USAGE.md#defining-custom-applications
- `pyflwor` language spec, https://github.com/timtadh/pyflwor

4 - Analysis of System Behaviour

Approaches to Behaviour Analysis

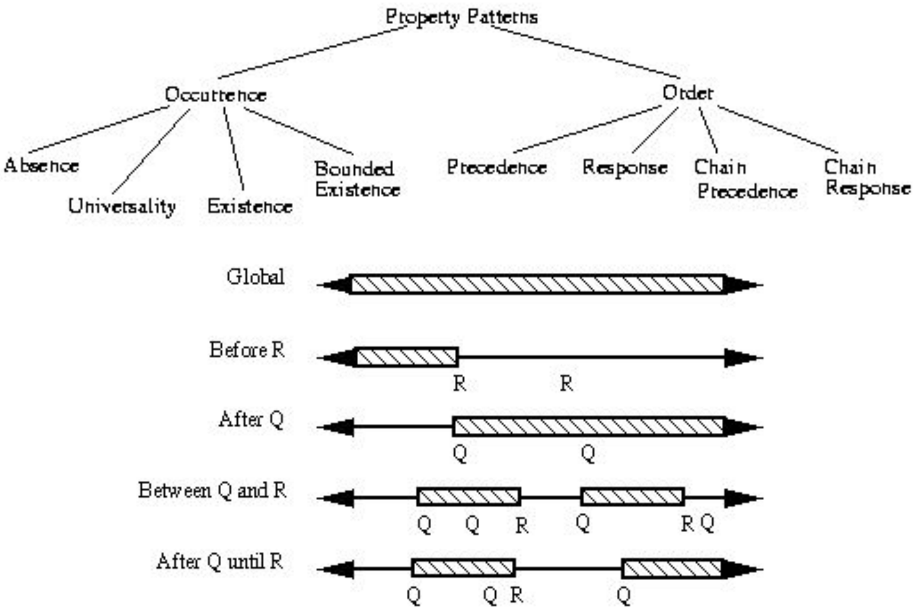
- Techniques seen so far focus on structural/architectural issues
- Unable to address (functional) *safety* properties

  Is the system guaranteed to never reach an invalid state?

- Must take into consideration the dynamic behaviour of the components
- Various approaches to verify correctness
  - Testing (runtime, incomplete)
  - Model checking (static, requires abstractions)
  - Deductive verification (static, semi-automatic)
  - ...

Property Specification

- To be sound, the expected behaviour must be formally defined
- Usually relying on some *temporal logic*
- To ease specification, well-known specification patterns have emerged
- Concrete *specification languages* further ease the writing of properties



Dwyer et al., 1999

Property Specification

- A typical formalism is *Linear Temporal Logic* (LTL)
- Propositional logic + temporal operators
  - Always (G)
  - Eventually (F)
  - Until (U)
  - ...

- Property *P* will eventually hold (existence)
  - F (*P*)
- Whenever *P* holds, *Q* will hold in the future (precedence)
  - F (*P* → F (*Q*))
- Property *P* will never hold after *Q* (scoped absence)
  - G (*Q* → G(!*P*))

HAROS Specification Language

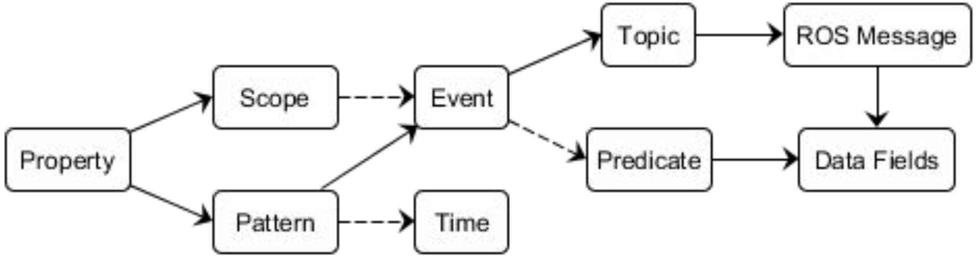
- HAROS provides a property specification language at the ROS level (**HPL**)
- Provides constructs for common specification patterns
- Passed to HAROS plugins that aim to check behaviours
- Specify both individual *node* and *system-wide* behaviour

Available HAROS plugins

- Property-based testing
- System-wide model checking
- Node model checking (WIP)

HAROS Specification Language

- Acts at the *message-passing* level
    - Events are publications at topics
- Relevant events may be filtered by predicates on message content
- Absence, existence, precedence, response patterns
- Global or restricted by scope



HAROS Specification Language

- Linear velocity in the `/cmd_vel` topic should never be above 1 m/s

```
globally: no /cmd_vel {linear.x > 1.0}
```

- A `/cmd_vel` with a velocity of zero is published as a consequence of a `/laser` message with `data` of 64 or lower, within 200 milliseconds

```
globally: /laser {data <= 64}
  causes /cmd_vel {linear = 0.0 and angular = 0.0}
  within 200 ms
```

Classical testing

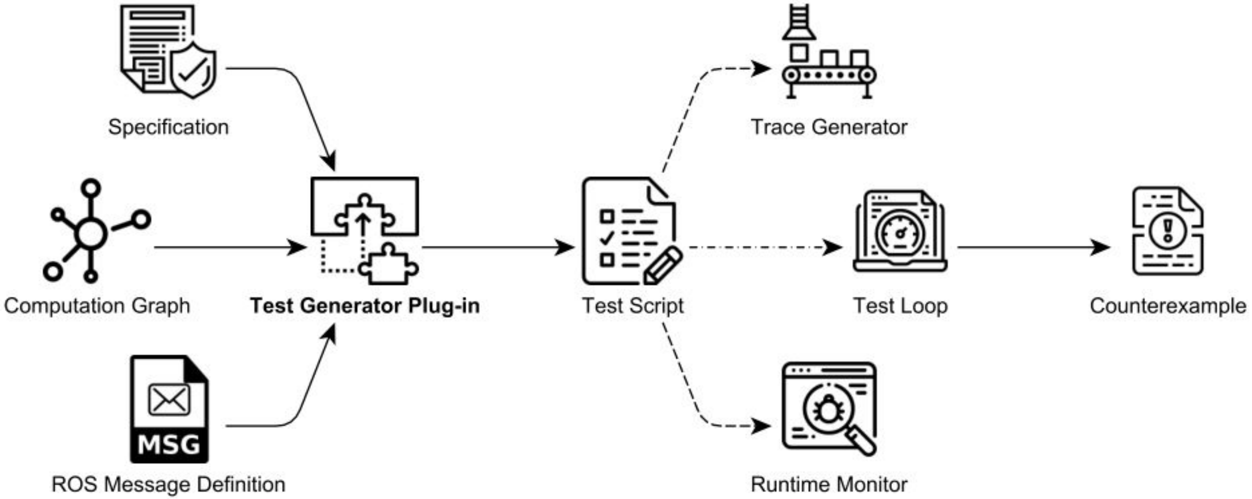
- How to test the system against a sequence of messages in ROS?
  - A unit test that sends X and Y?
  - What about interfering messages between them?
  - What about timing and delay issues?

Property-based testing

- Classical testing requires manual definition of input/output pairs
- Property-based testing
  - Outputs tested against oracle (the system spec)
  - Inputs automatically generated
  - Shrinking
- Not intended to replace specific unit tests

HAROS PBT Plugin

- HAROS spec language to specify expected behaviour
- Input traces generated from spec and architectural model
- Monitors to assert spec in runtime



HAROS PBT Plugin

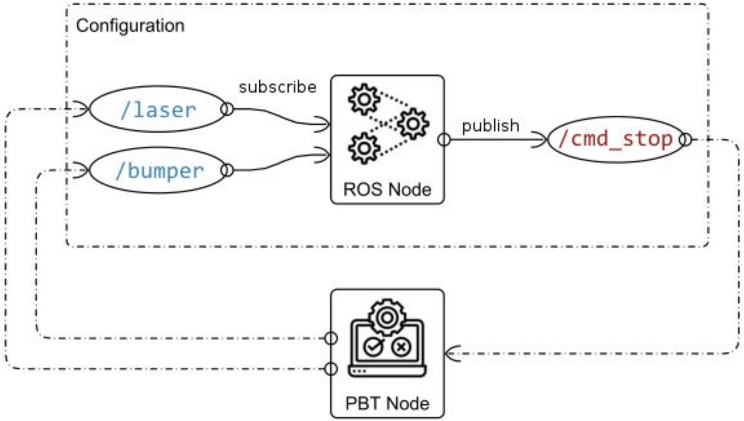
- Properties only over subscribed topics are used as axioms for the test generator
- Generated inputs try to uphold axiomatized properties
- Properties over published topics are used as test goals

Axiom:

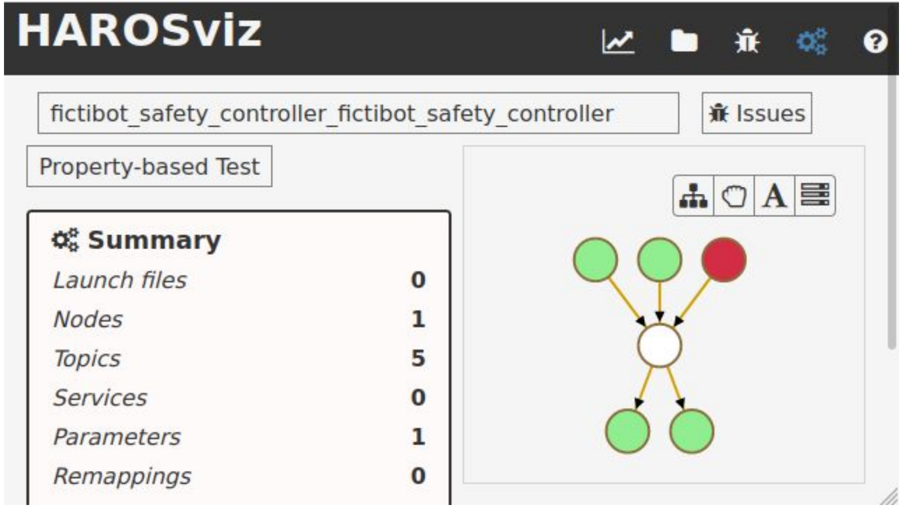
```
globally: no /laser {not data in [0 to 127]}
```

Test:

```
globally: /laser {data <= 32} causes
/cmd_stop within 200 ms
```

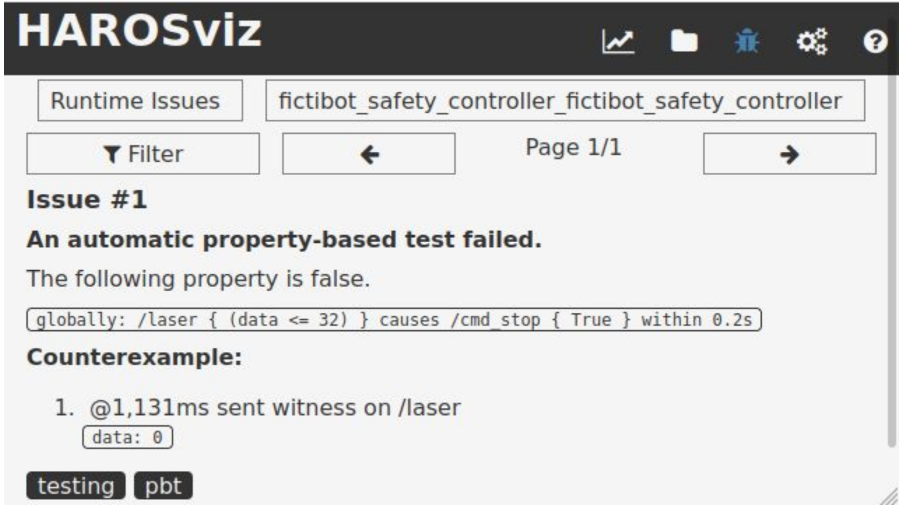


HAROS PBT Plugin



Summary

Launch files	**0**
Nodes	**1**
Topics	**5**
Services	**0**
Parameters	**1**
Remappings	**0**



Issue #1

An automatic property-based test failed.

The following property is false.

```
globally: /laser { (data <= 32) } causes /cmd_stop { True } within 0.2s
```

Counterexample:

1. @1,131ms sent witness on /laser

   ```
   data: 0
   ```

Hands-on Exercises

- Follow the link for exercises over Fictibot

  https://github.com/git-afsantos/haros_tutorials/tree/master/exercises/sec4-behaviour

- Explore the expected behaviour, specify properties and fix issues
- Demo and proposed solution

  https://youtu.be/6sHyu6bSJ-U

Additional Resources

- M. B. Dwyer, G. S. Avrunin, J. C. Corbett: **Patterns in Property Specifications for Finite-State Verification**. ICSE 1999: 411-420, https://doi.org/10.1145/302405.302672
- A. Santos: **Safety Verification for ROS Applications**. PhD Thesis. University of Minho, Braga, Portugal, https://git-afsantos.github.io/publication/phd-thesis
- A. Santos, A. Cunha, N. Macedo: **Property-based testing for the robot operating system**. A-TEST@ESEC/SIGSOFT FSE 2018: 56-62, https://doi.org/10.1145/3278186.3278195
- R. Carvalho, A. Cunha, N. Macedo, A. Santos: **Verification of system-wide safety properties of ROS applications**. IROS 2020: 7249-7254, https://doi.org/10.1109/IROS45743.2020.9341085
- HPL repository: https://github.com/git-afsantos/hpl-specs

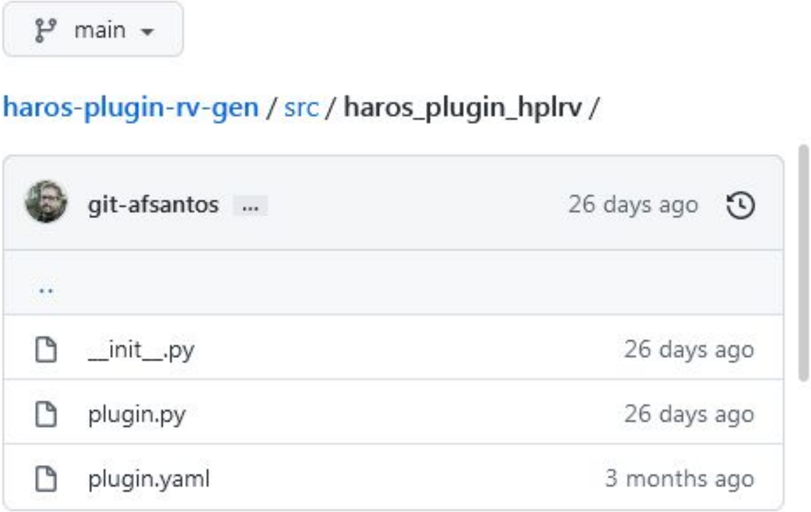
5 - Conclusion

Extending HAROS Analyses

- HAROS was built with extensibility in mind
- Python modules
- Alternative entry-points
    - Project-level (source code)
    - Configuration-level (architectural meta-model)
- Report issues with traceability to related elements

HAROS Plug-in Structure

- Python package containing:



haros-plugin-rv-gen / src / haros_plugin_hplrv /

git-afsantos ... 26 days ago

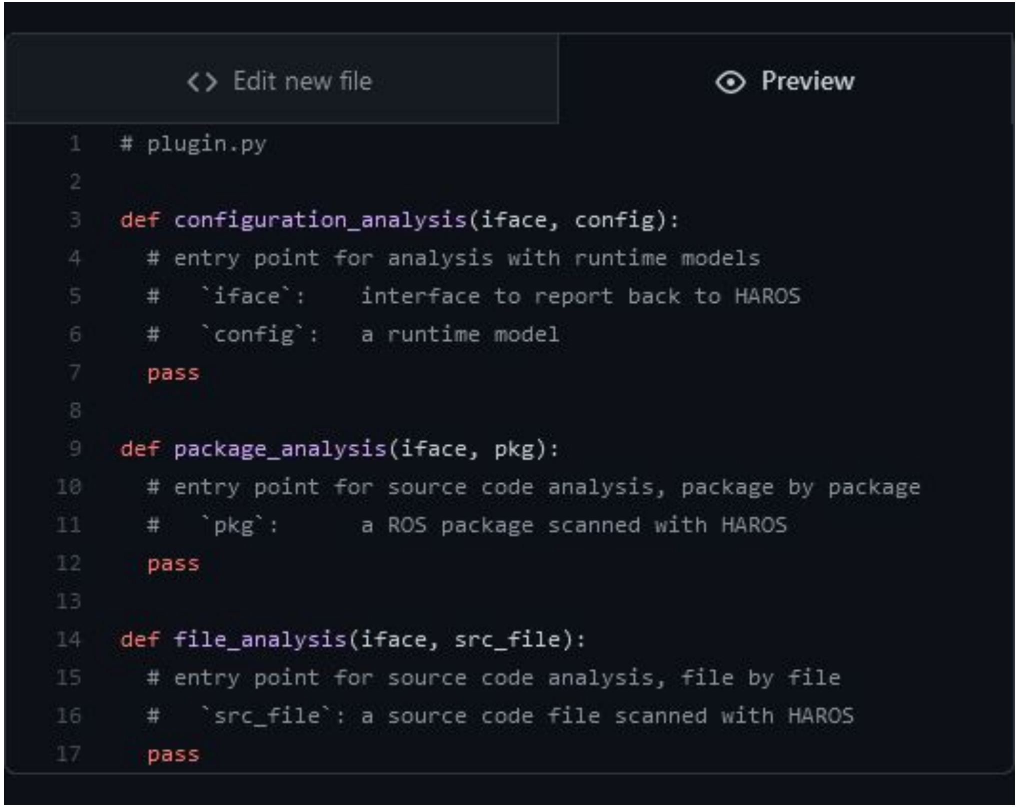
..

__init__.py 26 days ago

plugin.py 26 days ago

plugin.yaml 3 months ago

HAROS Plug-in Interface



```python
# plugin.py

def configuration_analysis(iface, config):
  # entry point for analysis with runtime models
  #   `iface`:    interface to report back to HAROS
  #   `config`:   a runtime model
  pass

def package_analysis(iface, pkg):
  # entry point for source code analysis, package by package
  #   `pkg`:      a ROS package scanned with HAROS
  pass

def file_analysis(iface, src_file):
  # entry point for source code analysis, file by file
  #   `src_file`: a source code file scanned with HAROS
  pass
```

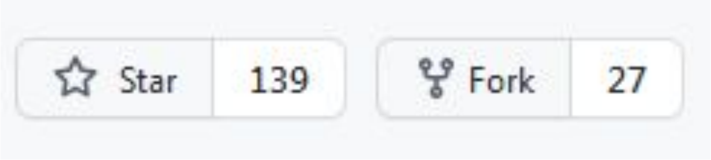
Future/Ongoing Work

- Support for ROS2 applications
- Feedback in continuous integration
- Variability-aware meta-model and analyses
- New plug-ins (alternative model checkers, control flow analysis, …)

Final Remarks

- HAROS aims to bridge the gap between **robotics** and **software engineering**
- It offers a variety of analyses, focusing on **automation** and **minimal user input**
- You are welcome to contribute - fork it, submit Pull Requests, report bugs, or simply answer a short user survey

https://forms.gle/aZE867Y3sMukVT6m6



Improving the Quality of ROS Applications with HAROS

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).