Verification of system-wide safety properties of ROS applications

Renato Carvalho, Alcino Cunha, Nuno Macedo, André Santos

INESCTEC

University of Minho
School of Engineering

IROS 2020

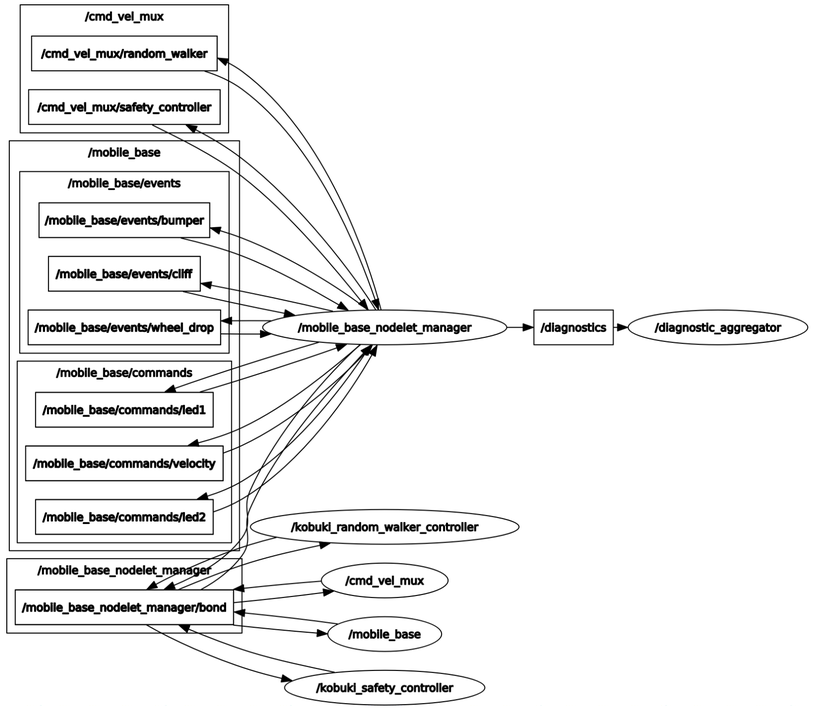
Verifying system-wide safety properties

A toy example

How to guarantee that a *bumper* event eventually triggers a stop *velocity?*

Verifying system-wide safety properties

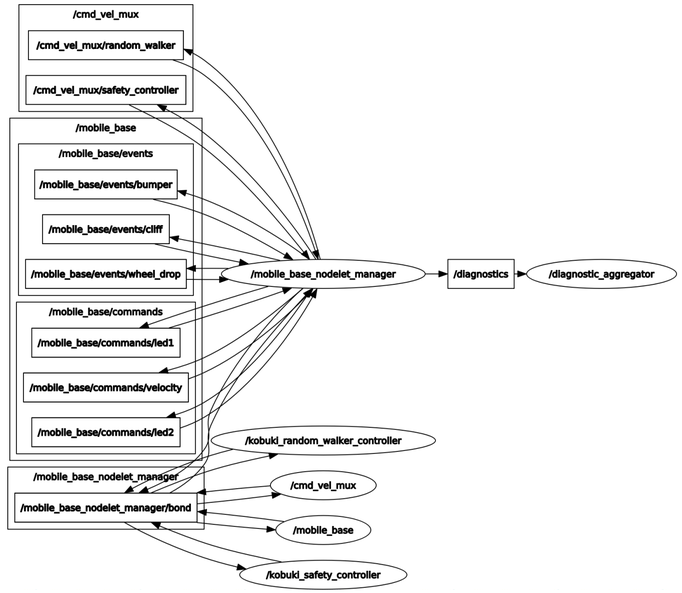
A toy example

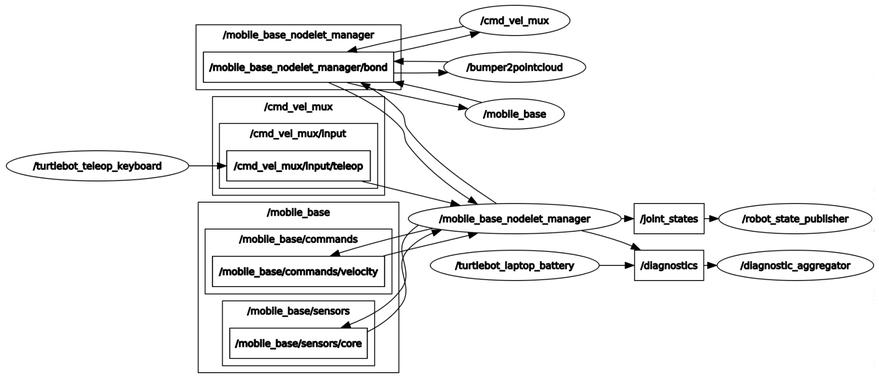


How to guarantee that a *bumper* event eventually triggers a stop *velocity?*

Verifying system-wide safety properties

A toy example

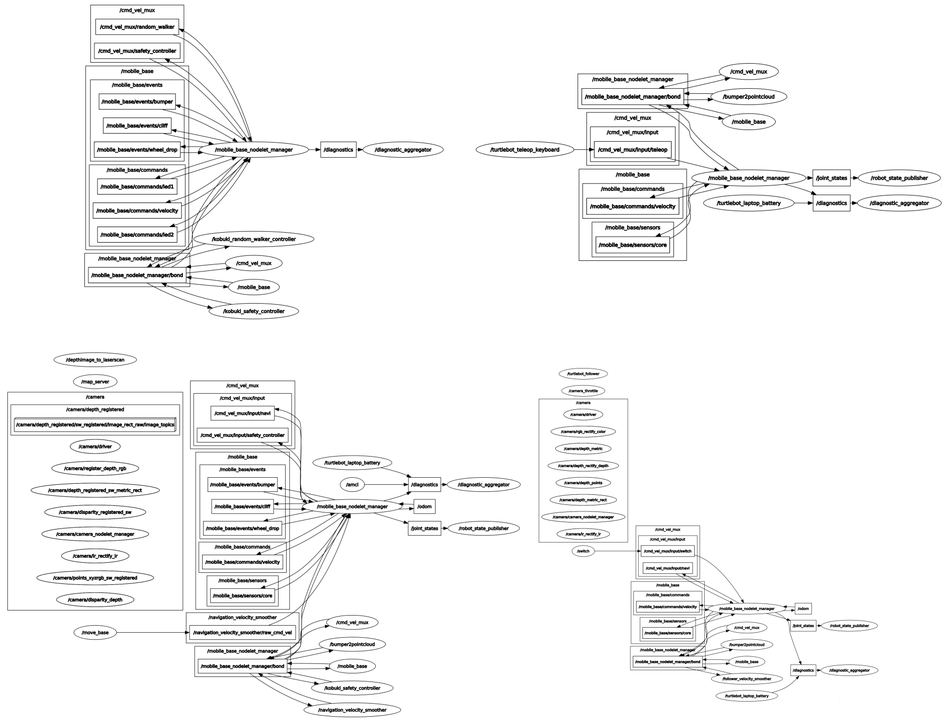




How to guarantee that a *bumper* event eventually triggers a stop *velocity?*

Verifying system-wide safety properties

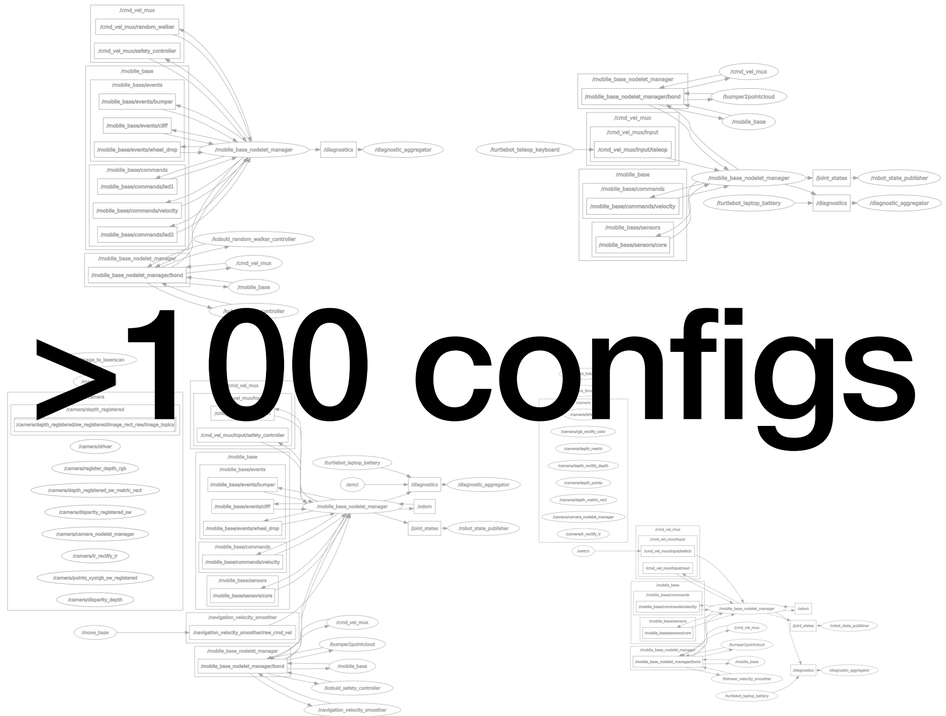
A toy example



How to guarantee that a *bumper* event eventually triggers a stop *velocity?*

Verifying system-wide safety properties

A toy example



>100 configs

How to guarantee that a *bumper* event eventually triggers a stop *velocity?*

Verifying system-wide safety properties

The challenges

- How to automatically extract architectures from ROS code?
- How to enable roboticists to specify node and system behaviour?
- Which formal languages support rich architectures with loose behaviour?
- Which formal analysis support the automatic verification of safety properties?

Verifying system-wide safety properties

The challenges

- How to automatically extract architectures from ROS code?
- How to enable roboticists to specify node and system behaviour?

- Which formal languages support rich architectures with loose behaviour?
- Which formal analysis support the automatic verification of safety properties?

Verifying system-wide safety properties

The challenges

- How to automatically extract architectures from ROS code?
- How to enable roboticists to specify node and system behaviour?

- Which formal languages support rich architectures with loose behaviour?
- Which formal analysis support the automatic verification of safety properties?

Verifying system-wide safety properties

The challenges

- How to automatically extract architectures from ROS code?
- How to enable roboticists to specify node and system behaviour?

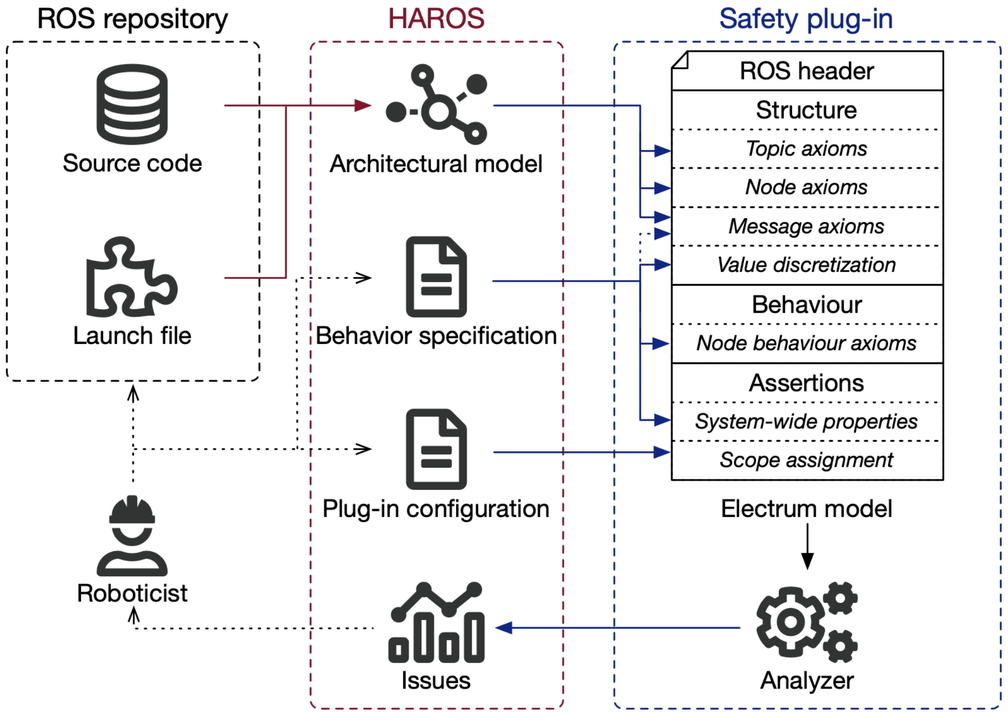
- Which formal languages support rich architectures with loose behaviour?
- Which formal analysis support the automatic verification of safety properties?

Verifying system-wide safety properties

Our proposal

A model checking plug-in for HAROS

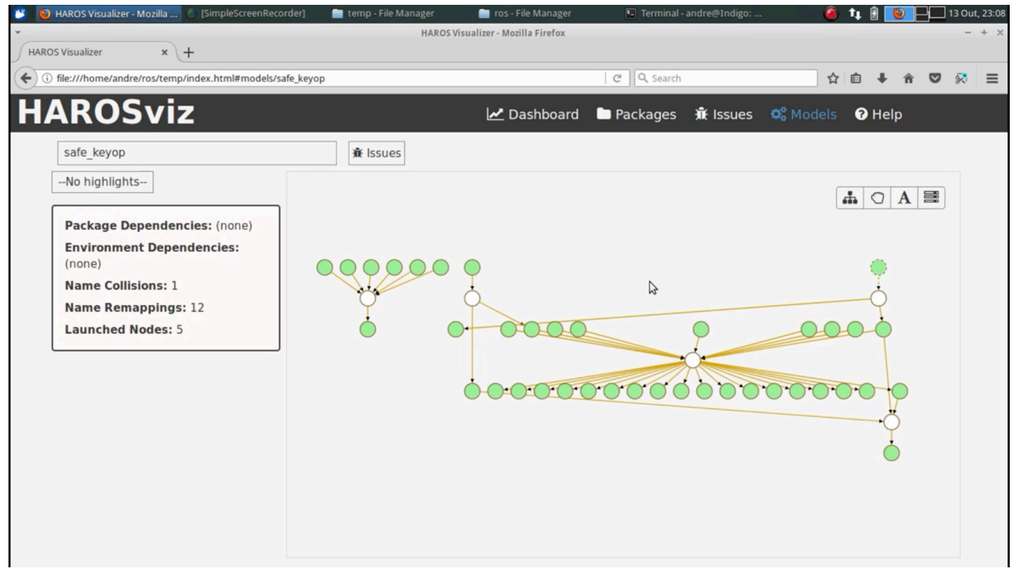
- User only required to loosely specify node behaviour and desired properties
- Automatically reports counter-examples in a ROS-friendly interface



HAROS

The quality assurance platform

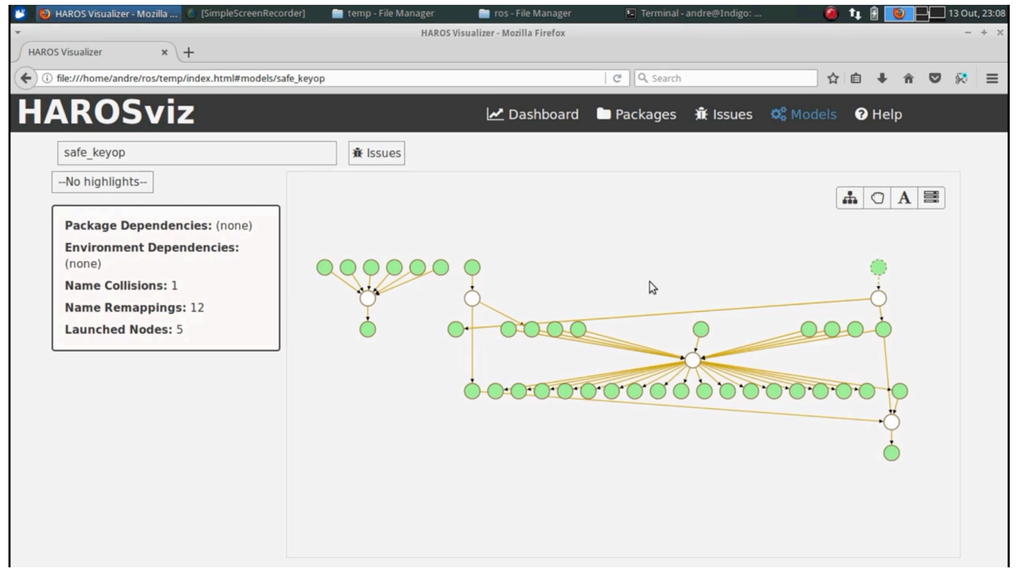
- Plug-in based platform for ROS quality assessment in continuous integration
- Automatically extracts arquitectural models from ROS code
- Detected issues are presented in a unified interface, with traceability to code



HAROS

The quality assurance platform

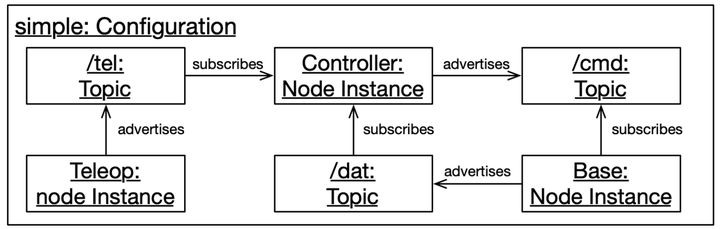
- Plug-in based platform for ROS quality assessment in continuous integration
- Automatically extracts arquitectural models from ROS code
- Detected issues are presented in a unified interface, with traceability to code



HAROS

The specification language

- HAROS provides a DSL to specify behaviours as well-known patterns
- A specification has an activation scope and a property on message events
- Can be used to specify the behaviour of individual nodes and of the whole system



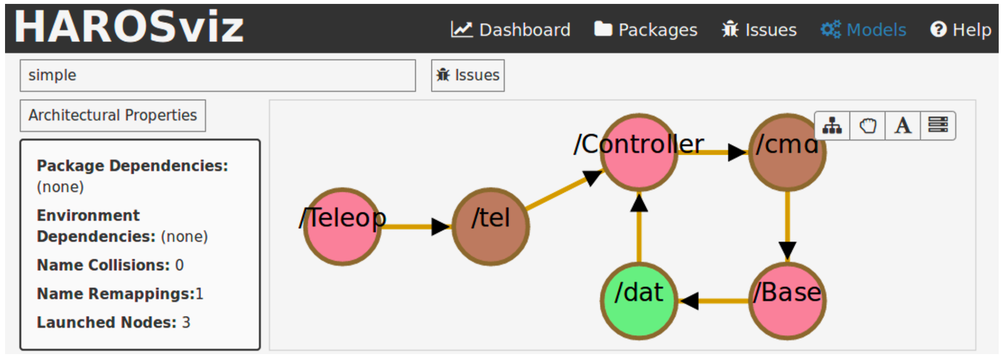

```
Teleop:
  globally: no /tel{val not in 0 to 100}
Controller:
  after /dat{val=0} until /dat{val!=1}:
    no /cmd{val!=0}
   …
  globally: /cmd{val!=0} as m requires /tel{val=$m.val}
  globally: /dat{val=0} causes /cmd{val=0, msg="stop"}

simple:
  globally: /cmd{msg="stop"} requires /dat{val=0}
```

HAROS

The model checking plug-in

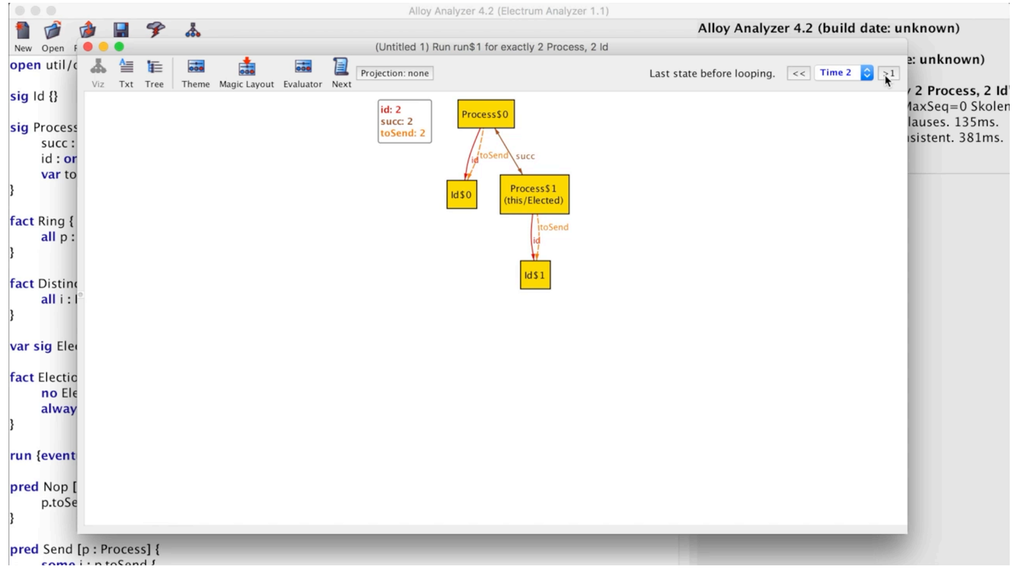
- Translates a ROS architecture, node behaviour and desirable properties into Alloy
- Reports back counter-examples at the ROS-level, either textually or graphically at the architecture



Alloy6 (Electrum)

A model checker for relational linear temporal logic

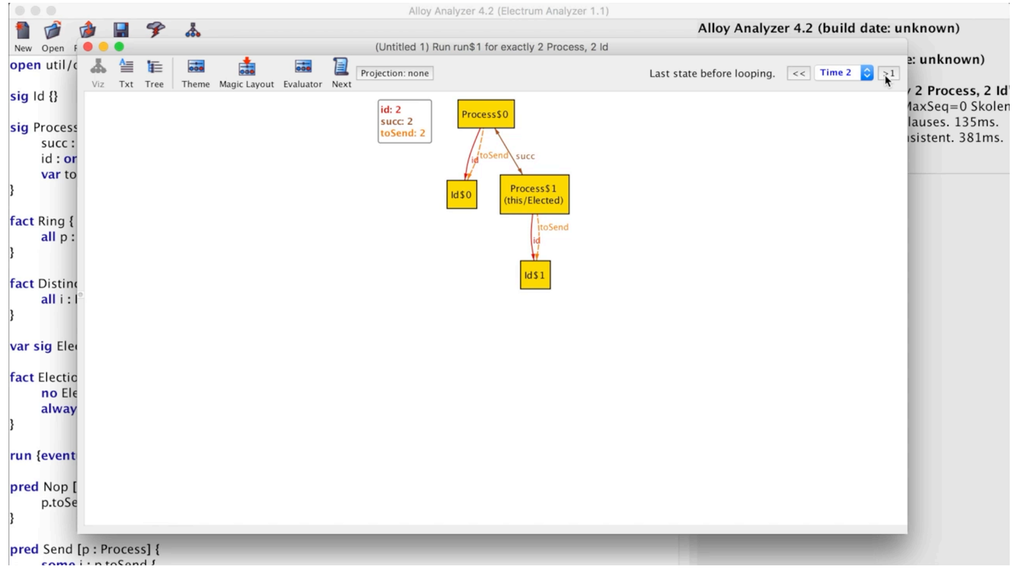
- Formal specification language with structural and dynamic constructs
- Declarative specifications, behaviour can be under-specified
- Automatic verification through SAT solving, returns counter-example traces
- Varying universe size provides increasing confidence



Alloy6 (Electrum)

A model checker for relational linear temporal logic

- Formal specification language with structural and dynamic constructs
- Declarative specifications, behaviour can be under-specified
- Automatic verification through SAT solving, returns counter-example traces
- Varying universe size provides increasing confidence



Alloy6 (Electrum)
Encoding ROS architectures

- Architectures encoded in a straight-forward way as fixed signatures
- Messages (and their values) are left unrestricted
  - An arbitrary number will be considered (within scopes)
- Large types (eg, integers) would overwhelm analysis
  - Relevant ranges inferred from the specs, only those classes are encoded

```
abstract sig Topic, Field, Value {}
sig IntVal, StrVal extends Value {}
abstract sig Node { subs, advs       : set Topic,
                    var inbox, outbox: set Message }
sig Message { topic: one Topic,
              val  : Field->lone Value }

one sig tel, dat, cmd extends Topic {}
one sig Teleop, Base, Controller extends Node {}
fact Links {
  advs = Teleop->tel + Base->dat + Controller->cmd
  subs = Controller->(tel+dat) + Base->cmd }

one sig tel_val, dat_val, cmd_val, cmd_msg extends Field {}
fact Fields {
  all m: topic.cmd {
    m.val in cmd_val->IntVal + cmd_msg->StrVal
    m.val in (cmd_val+cmd_msg)->one (IntVal+StrVal) }
  … }

lone sig Int_0,Int_1 in IntVal {}
sig Int_0_10,Int_0_100 in IntVal {}
fact Values {
  Int_0+Int_1 in Int_0_10 and Int_0_10 in Int_0_100
  no Int_0&Int_1 }
```

Alloy6 (Electrum)
Encoding ROS behaviours

- The message-passing behaviour is loosely specified
  - Messages in the outbox will eventually reach the inboxes of subscribers
- Semantics of the specification language is provided in (metric) linear temporal logic
- Node behaviour imposed as a fact, desirable system properties as assertions to be checked

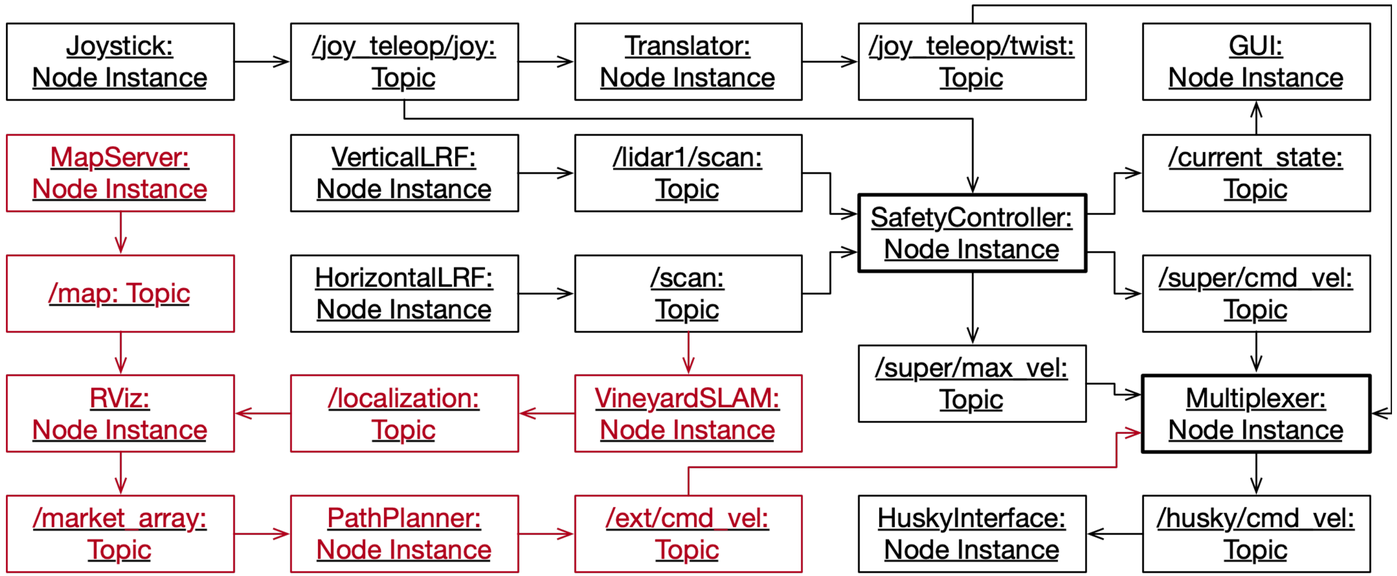
```
fact Messages { always {
  all m: Node.outbox {
    all n: subs.(m.topic) | eventually m in n.inbox
    eventually m not in Node.outbox }
  all m: Node.inbox |
    before once m in advs.(m.topic).outbox }
  … }

fact NodeBehavior { always {
  no m: Teleop.outbox&topic.tel |
    tel_val.(m.val) not in Int_0_100
  …
  all m: Controller.inbox&topic.dat |
    dat_val.(m.val) = Int_0 implies after eventually
      some m0: Controller.outbox&topic.cmd {
        cmd_val.(m0.val) = Int_0
        cmd_msg.(m0.val) = Str_stop } } }

assert simple1 { always {
  all m: Node.outbox&topic.cmd |
    cmd_msg.(m.val) = Str_stop implies before once
      some m0: Node.outbox&topic.dat |
        dat_val.(m0.val) = Int_0 } }
```

Evaluation

Case study: AgRob V16



Modular robot, alternative configurations for navigation

Evaluation

Case study: AgRob V16

```
SafetyController:
  globally: /current_state{data[0]=6} requires /joy_teleop/joy{button[0]=1}
  globally: no /super/cmd_vel{linear.x not in 0 to 10}
  globally: /super/cmd_vel{linear.x in 3.8 to 4.2} requires
                /joy_teleop/joy{button[0]=0, button[1]=1} ||
                /joy_teleop/joy{button[4]=1, button[5]=0}

map:
  globally: /agrobv16/current_state{data[0]=3} requires
                /joy_teleop/joy{button[0]=0, button[1]=1}
  globally: /husky/cmd_vel{linear.x=0, angular.x in -100 to 100} requires
                /scan{ranges[0] in 0 to 4} ||
                /joy_teleop/joy{button[0]=1}
```

Modular robot, alternative configurations for navigation

Evaluation

Case study: AgRob V16

- Finds a counter-example caused by change in architecture
  - Wrong specification
  - Less than 1 minute for 10 Messages

[Ver se tenho a VM a funcionar e sacar screenshow]

Conclusions

Limitations and future work

- Scopes (size of universe and length of trace) must be specified manually

Conclusions

Limitations and future work

- Scopes (size of universe and length of trace) must be specified manually
  - Mechanisms to infer sensible scopes and support the user

Conclusions

Limitations and future work

- Scopes (size of universe and length of trace) must be specified manually
  - Mechanisms to infer sensible scopes and support the user
- No timed properties

Conclusions

Limitations and future work

- Scopes (size of universe and length of trace) must be specified manually
  - Mechanisms to infer sensible scopes and support the user
- No timed properties
  - Further abstractions could address certain timed issues

Conclusions

Limitations and future work

- Scopes (size of universe and length of trace) must be specified manually
  - Mechanisms to infer sensible scopes and support the user
- No timed properties
  - Further abstractions could address certain timed issues
- Loose specified behaviour may lead to false positives

Conclusions

Limitations and future work

- Scopes (size of universe and length of trace) must be specified manually
  - Mechanisms to infer sensible scopes and support the user
- No timed properties
  - Further abstractions could address certain timed issues
- Loose specified behaviour may lead to false positives
  - Explore the combination of static and runtime analysis

Verification of system-wide safety properties of ROS applications

Renato Carvalho, Alcino Cunha, Nuno Macedo, André Santos

University of Minho
School of Engineering

UNIÃO EUROPEIA

Fundo Europeu
de Desenvolvimento Regional