Towards Rule-based Dynamic Safety Monitoring for Mobile Robots

Sorin Adam¹, Morten Larsen¹, Kjeld Jensen², Ulrik Pagh Schultz²

¹ Conpleks ApS , ² University of Southern Denmark

Context: Safety of large robots in complex environments

Experimental platforms used in the paper
(Frobornod ROS architectural framework for field robots)
(Frobornod)
(Kobuki standard ROS package)

Future target platforms: field robots moving towards commercialization

Lack of isolation
Safety-critical parts of the software are distributed into different nodes
Risk that programming errors will cause violations of safety properties
Implicit assumptions regarding the reliability of the hardware

Open Source SOUP
Open-source software platforms and components enable massive advances
Open-source "software of unknown providence" (SOUP) significant challenge for safety certification

Safety certification
Key challenge: make the effort of safety-certifying a new software revision comparable with the effort required to develop the software revision

Concrete examples
Hardware faults (Frobimind):
- Sensors and actuators misbehaving
- Software errors (Kobuki simulator)
- Robot controller misbehaving

Problem: Complex software not explicitly addressing safety issues

Solution: Declaratively specified safety kernel

Architectural concept

Functionality (ROS) → Safety kernel (RuBaSS) → Hardware and OS

RuBaSS safety DSL
Basic idea: safety kernel that constrains robot behavior is automatically generated from RuBaSS declarative specification.

// action implemented by the underlying firmware
action primitive stop;

// each entity groups nodes according to functionality;
// contains rules declaring behavioral properties
entity drive_system : encoder_node, actuator_node;
{
  // each rule constrains topics data over time
  rules:
    actuator_erratic:
      (topic /cmd_vel_left.linear.x > 0.02m/s or
       topic /cmd_vel_left.linear.x < -0.02m/s) and
       topic /encoder_left.data == 0) for 0.4sec;

  // safety rules link entity properties to actions
  if (drive_system.actuator_erratic for 1 sec)
      then { stop; }
}

Experimental setup

Computer
Frobit
Fault injector
Safety monitor
Serial

Hardware controller
Motor drivers
Wheel encoders

Kobuki (simulator)

Contributions
- Safety-critical concerns regarding the robot software explicitly declared in terms of the overall functionality of the software.
- Addresses the functionality of the system as a whole in terms of component properties and communication.
- Proposal and proof-of-concept experiments of a simple yet expressive rule-based language for enforcing safety constraints on existing ROS-based software.
- Initial experimental documentation through experiments that test safety-oriented scenarios involving software and hardware failures.

Future work
- C++ code generation
- Support for service calls and other communication patterns
- Consistency using a component model that annotates physical units to components
- Improved fault-handling based on diagnosis and fault isolation
- Improving the RuBaSS language to statically detect overlapping safety rules or potential contradictions
- Systematic and realistic validation of RuBaSS

Concretes examples
Hardware faults (Frobimind):
- Sensors and actuators misbehaving
- Software errors (Kobuki simulator)
- Robot controller misbehaving

Open Source SOUP
Open-source software platforms and components enable massive advances
Open-source "software of unknown providence" (SOUP) significant challenge for safety certification

Key challenge: make the effort of safety-certifying a new software revision comparable with the effort required to develop the software revision

Concrete examples
Hardware faults (Frobimind):
- Sensors and actuators misbehaving
- Software errors (Kobuki simulator)
- Robot controller misbehaving

Solution: Declaratively specified safety kernel

Architectural concept

Functionality (ROS) → Safety kernel (RuBaSS) → Hardware and OS

RuBaSS safety DSL
Basic idea: safety kernel that constrains robot behavior is automatically generated from RuBaSS declarative specification.

// action implemented by the underlying firmware
action primitive stop;

// each entity groups nodes according to functionality;
// contains rules declaring behavioral properties
entity drive_system : encoder_node, actuator_node;
{
  // each rule constrains topics data over time
  rules:
    actuator_erratic:
      (topic /cmd_vel_left.linear.x > 0.02m/s or
       topic /cmd_vel_left.linear.x < -0.02m/s) and
       topic /encoder_left.data == 0) for 0.4sec;

  // safety rules link entity properties to actions
  if (drive_system.actuator_erratic for 1 sec)
      then { stop; }
}

Experimental setup

Computer
Frobit
Fault injector
Safety monitor
Serial

Hardware controller
Motor drivers
Wheel encoders

Kobuki (simulator)

Contributions
- Safety-critical concerns regarding the robot software explicitly declared in terms of the overall functionality of the software.
- Addresses the functionality of the system as a whole in terms of component properties and communication.
- Proposal and proof-of-concept experiments of a simple yet expressive rule-based language for enforcing safety constraints on existing ROS-based software.
- Initial experimental documentation through experiments that test safety-oriented scenarios involving software and hardware failures.

Future work
- C++ code generation
- Support for service calls and other communication patterns
- Consistency using a component model that annotates physical units to components
- Improved fault-handling based on diagnosis and fault isolation
- Improving the RuBaSS language to statically detect overlapping safety rules or potential contradictions
- Systematic and realistic validation of RuBaSS