Formal Verification of ROS-based Robotic Applications using Timed-Automata

Raju Halder & José Proença & Nuno Macedo & André Santos
(Guillermina Cledou)

FORMALISE 2017
Motivation

Kobuki Robot

http://kobuki.yujinrobot.com/

(a) Template for Sensors
(b) SafetyController-Update: safety controller which updates internal state.
(c) Template for safety controller’s Subscriber Queues QWheel, QBumper, and QCli
(d) “callAvailable (ros::WallDuration TimeOut)” method to process all callbacks currently in the Callback Queue for Safety-Controller.

Figure 8: Implementation of the module SafetyController-Update using Uppaal Model Checker

Analyse with Timed-automata
Outline

- How is it used
- Real-time challenges
- Starting point of our analysis
- What is it
- What we model
- What we verify

**ROS**

**Timed-automata**

- $t = \text{spinRate}
- \text{spinOnce!}$
- $t \leq \text{spinRate}$
- $t = 0$
Why robotics?

- modern robotics are applied in industrial, agricultural, medical and domestic domains
- must be flexible, configurable and adaptive
- ever-closer human-robot interaction
Why robotics?

- modern robotics are applied in industrial, agricultural, medical and domestic domains
- must be **flexible, configurable and adaptive**
- intended to have closer **human-robot interaction**

**requires software controllers**

**safety verification is critical**
Why ROS?
Why ROS?

- middleware for developing robots
- modular, portable and configurable
- *thousands* of publicly available libraries
ROS Architecture

- Component-based, *nodes* interacting with each other through *topics*
- Synchronous (RPC) and asynchronous (publish-subscribe) communication
- Use of explicit timeouts at application level
- Manually configured message queues and processing rates
ROS Architecture

- Component-based, *nodes* interacting with each other through *topics*
- Synchronous (RPC) and asynchronous (publish-subscribe) communication
- Use of explicit timeouts at an application level
- Manually configured message processing rates error-prone!
ROS Example

TurtleBot 2 (Kobuki base)
ROS Example

TurtleBot 2 (Kobuki base)

Cliff
Bumper
Wheel Drop

Topic
Random
Muxer

Teleop
Left Motor

will the cliff sensor always stop the robot?
Code analysis

```c
int main(int argc, char **argv) {
    ros::init(argc, argv, "talker");
    ros::NodeHandle n;
    ros::Publisher chatter_pub =
        n.advertise<std_msgs::String>("chatter", 1000);
    ros::Rate loop_rate(10);
    while (ros::ok()) {
        std_msgs::String msg;
        //... do some work ...
        chatter_pub.publish(msg);
        loop_rate.sleep();
    }
    return 0;
}
```

```c
void chatterCallback(const
    std_msgs::String::ConstPtr msg) {
    //... do some work ...
}
```

```c
int main(int argc, char **argv) {
    ros::init(argc, argv, "listener");
    ros::NodeHandle n;
    ros::Subscriber sub =
        n.subscribe<std_msgs::String>("chatter", 1000, chatterCallback);
    ros::Rate loop_rate(10);
    while (ros::ok()) {
        //... do some work ...
        ros::spinOnce();
        loop_rate.sleep();
    }
    return 0;
}
```
Code analysis

```cpp
int main(int argc, char **argv) {
    ros::init(argc, argv, "talker");
    ros::NodeHandle n;
    ros::Publisher chatter_pub =
        n.advertise<std_msgs::String>("chatter", 1000);
    ros::Rate loop_rate(10);
    while (ros::ok()) {
        std_msgs::String msg;
        //... do some work ...
        chatter_pub.publish(msg);
        loop_rate.sleep();
    }
    return 0;
}

void chatterCallback(const
    std_msgs::String::ConstPtr msg) {
    //... do some work ...
}
```

Publish/Subscribe

Not done automatically!
Two examples

Safety-controller combining safety sensors + random walker

2 publishers
1 subscriber
Two examples

1. Clarify how communication works
2. Illustrates how to model with timed-

1. More complex scenario
2. Illustrate what can we learn in a more complex application

Safety sensors + random walker
change in the state variables of the safety controller. For
will use the left wheel sensor—may fail to trigger the desired
sensors in the presence of overflows (Subsection 5.3.1), and that
queue overflow, this section identifies some desirable, con-

5.3 Finding problems in Kobuki

4.3 Verification in UPPAAL

The corresponding implementation of the models in UP-

8 automata
communication =
shared actions (or shared variables)

Timed automaton - publishes every “PubTime” seconds
UPPAAL
UPPAAL

Simulate (discover traces)

run and view 8 automata in parallel
UPPAAL

We can check temporal properties
A [] (not queueSub.Overflow) failed:
the queue of the subscriber can overflow
## Experimenting with parameters

<table>
<thead>
<tr>
<th>Queue Sizes</th>
<th>Transmission Time</th>
<th>Callback Time</th>
<th>Publishing Time-gap</th>
<th>Spin Time-gap</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Q_{1\rightarrow1}$</td>
<td>$Q_{2\rightarrow1}$</td>
<td>$Q_{3\leftarrow1}$</td>
<td>$T_{min}$</td>
<td>$T_{max}$</td>
<td>CB_{min}</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

**List desired properties**

**Find the good combinations of parameters**

*(Manual process)*
Repeat approach for large example

Find desired properties
Experiment with parameters

Find out exactly when:
- Sensor messages get lost (too many)
- Remote never manages to go through (sensors have priority)
Wrapping up

ROS program communicating components

Extract

Find parameters that obey timed properties

model as

timed automata

(a) Template for Sensors
(b) SafetyController-Update: safety controller which updates internal state.
(c) Template for safety controller's Subscriber Queues QWheel, QBumper, and QCli.
(d) "callAvailable (ros::WallDuration TimeOut)" method to process all callbacks currently in the Callback Queue for Safety-Controller.

Figure 8: Implementation of the module SafetyController-Update using Uppaal Model Checker.

Find parameters that obey timed properties
Future work:
- Automatic code analysis
- Automatic search for optimal parameters
- ROS as software product lines of components

Find parameters that obey timed properties