Automatic Refactoring of Robotics Code

Audrey Seo

REUSE – ISR

Carnegie Mellon University

Problem

- ROS is a framework for programming robots
- The first version, ROS1, is not compatible with ROS2
- Want to automatically port ROS1 to ROS2

Motivation

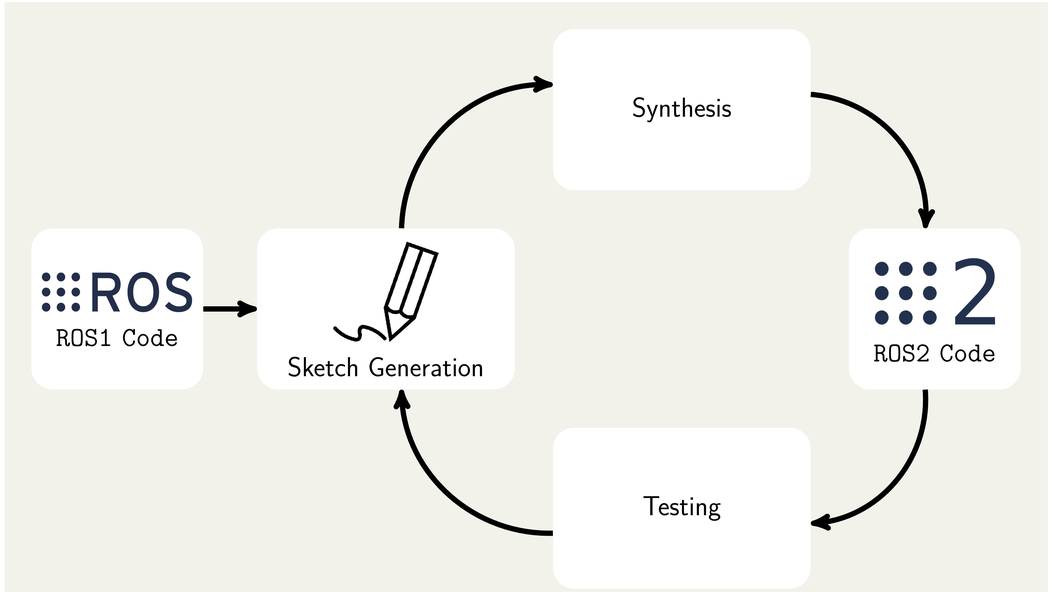
- Difficult to update code (e.g. Python 2.7 vs Python 3)
- Coding robots is more interesting and fun than refactoring
- ROS1 has major security vulnerabilities

```cpp
void chatterCallback(const std_msgs::String::ConstPtr&
  msg){
  ROS_INFO("I heard: [%s]", msg->data.c_str());
}

int main(int argc, char **argv){
  ros::init(argc, argv, "listener_node");
  ros::NodeHandle n;
  ros::Subscriber sub = n.subscribe("chatter", 1000,
  chatterCallback);
  ros::spin();
  return 0;
}
```

Figure 1: Original C++ code for a "listener" node, using ROS1 APIs.

```cpp
void chatterCallback(const std_msgs::String::ConstPtr&
  msg){
  char* string_template = "I heard: [%s]";
  ?#? //ROS_INFO, std_msgs::String::data::c_str
  }
}

int main(int argc, char **argv) {
  std::string node_name = "listener_node";
  ?#? // ros::init
  ?#? // ros::NodeHandle
  std::string topic_name = "chatter";
  int queue_size = 1000;
  ?#? // ros::NodeHandle::subscribe
  ?#? // ros::spin
  return 0;
}
```

Figure 2: The sketch created from the code in Fig. 1 by putting holes (denoted by ?#?) in the place of any ROS APIs.



Sketch Generation

A "sketch" could be generated using data flow analysis (DFA) to create blocks of code. However, in this first prototype, we have manually divided up the code into blocks.

ROS APIs were tagged with 16 different keys that encoded programmer intent (see Table 1).

As first test cases for refactoring, we chose two node programs: a "listener" and a "talker" that communicate over a channel called a *topic*.

	Overlap with						
Tag	other	sub	pub	constructor	ros	node	Total
node	0	4	4	14	0	-	66
ros	0	3	0	6	-	-	41
constructor	7	9	6	-	-	-	31
pub	7	0	-	-	-	-	16
sub	0	-	-	-	-	-	16
other	-	-	-	-	-	-	37

Table 1: Breakdown of the top 5 tags, and all others are in the "other" category. The average number of tags per method was 1.5, thus the numbers in the total column don't add up to the number of methods in the search space: 142.

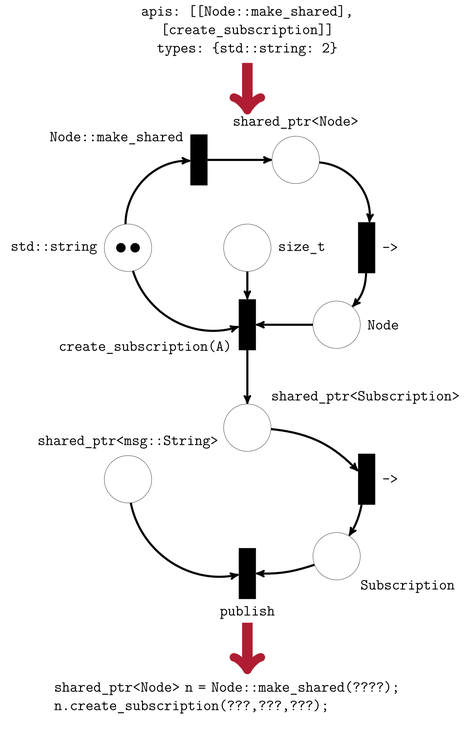


Figure 3: Partial petri net of ROS2 APIs for the listener example (see Fig. 1). Input and output for synthesis is shown above and below, respectively.

Synthesis

The ROS2 code synthesis is driven by searching a petri net (see Fig. 3), where the nodes are types and the transitions are various ROS2 APIs.

Results

The resulting petri net has

- 308 places (=types)
- 560 transitions (=ROS APIs)
- 1383 edges (=relationships between APIs and types)

Several candidate solutions are generated for each program.

- Listener: 16
- Talker: 64

Challenges

- C++ grammar: complicated and ambiguous
- C++ types: parametric polymorphism
- Diagnosing cause of failing test
- Slow ROS compile time → test cases not feasible