Johannes Mey\textsuperscript{1}, Ariel Podlubne\textsuperscript{1,2}, René Schöne\textsuperscript{1}, Paul Gottschaldt\textsuperscript{1,2}, Diana Göhringer\textsuperscript{1,2}, Uwe Aßmann\textsuperscript{1,2}

\textsuperscript{1}Technische Universität Dresden
\textsuperscript{2}Centre for Tactile Internet with Human-in-the-Loop (CeTI)

Systematic Testing of a ROS Interface Specification Backend

6th International Workshop on Robotics Software Engineering (RoSE’24)
Lisbon, April 15th 2024
The System Under Test: FIRM [Pod+21]

“FIRM”:

- FPGA (VHDL)
- ROS 1 and ROS 2 Middleware

Goal:

- Receive ROS messages on the hardware (PL) bypassing the CPU (PS)

The System Under Test: FIRM [Pod+21]

“FIRM”:
- FPGA (VHDL)
  ROS 1 and ROS 2 Middleware

Goal:
- Receive ROS messages on the hardware (PL) bypassing the CPU (PS)

Creating a ROS Middleware for FPGAs

ROS Middleware
- Communication components in **library**
- **Generated bindings** for each ROS message type

ROS Message Types
- Custom format in ROS1
- Mapped to OMG IDL in ROS2

Challenges
- Support **all** ROS 1 and 2 versions
- Support **multiple** FPGA vendors/VHDL dialects
- ROS message **complexity**
- Testing on **FPGA-hardware**
- Distributed **skills**

```plaintext
1 std_msgs/Header header
2    uint32 seq
3    time stamp
4    string frame_id
5 uint32 height
6 uint32 width
7 string encoding # unconstrained size
8 uint8 is_bigendian
9 uint32 step
10 uint8[] data

# nested message
# unconstrained size
```
System under Test

FIRM [Pod+21] Model-driven Code Generation Tool

Legend
- input provided by middleware/user
- configurable internal specification
- generated code/specification
- tool/component
- model
- output
Systematic Testing of a ROS Interface Specification Backend
Johannes Mey, Ariel Podlubne, René Schöne, Paul Gottschaldt, Diana Göhringer, Uwe Aßmann
April 15th 2024
System under Test

FIRM [Pod+21] Model-driven Code Generation Tool

Code Generation Component

- ROS 1 Hardware Description Template (HDT)
- ROS 2 HDT

- ROS 1 msg parser
- ROS 1 msg model
- message-dependent template config generator
- model-to-text
- template config
- template engine

- intermediate model converter
- intermediate message model

- Test Generation Component

- ROS 1 test generator
- ROS 1 Test Template
- ROS 1 Test Nodes (C++)
- test build files

- ROS 2 msg parser
- ROS 2 msg model
- config model
- model-to-text
- template config
- template engine

- Test Generation Component

Legend

- input provided by middleware / user
- configurable internal specification
- generated code / specification
- tool / component
- model
- output

Systematic Testing of a ROS Interface Specification Backend
Johannes Mey, Ariel Podlubne, René Schöne, Paul Gottschaldt, Diana Göhringer, Uwe Aßmann
April 15th 2024

Slide 4 of 16
System under Test

FIRM [Pod+21] Model-driven Code Generation Tool

Code Generation Component

ROS 1 Frontend Component
- ROS 1 msg parser
- ROS 1 msg model
- intermediate model converter
- intermediate message model
- message-dependent template config generator
- template config
- template engine

ROS 2 Frontend Component
- ROS 2 msg parser
- ROS 2 msg model
- intermediate model converter
- intermediate message model
- message-dependent template config generator
- template config
- template engine

Test Generation Component

ROS 1 test generator
- config model
- model-to-text
- template config
- template engine

ROS 2 test generator
- config model
- model-to-text
- template config
- template engine

Validation Architecture

message-dependent IP (VHDL)

ROS test nodes (C++)

Template config

ROS 1 Noetic
ROS 1 msg
ROS 1 Hardware Description Template (HDT)

ROS 2 Humble
ROS 2 msg
ROS 2 HDT

ROS 2 Iron
ROS 2 msg

Legend

input provided by middleware / user
configurable internal specification
generated code / specification
tool / component
model
output

Systematic Testing of a ROS Interface Specification Backend
Johannes Mey, Ariel Podlubne, René Schöne, Paul Gottschaldt, Diana Göhringer, Uwe Aßmann
April 15th 2024
Test Stages

**Frontend Tests (TS1)**
- ROS integration
- Parser

**Code Generation tests (TS2)**
- Regression tests

**Runtime tests (TS3)**
- Generate messages
- Pipe through FPGA (sim)
- Compare input/output
- Only frontend shared with FIRM
Execution

- Dockerized Gitlab CI Pipeline
- Automatic ROS1/2 switch based on ROS system variable → add new ROS version = add new base image
Strategies / Insights / Lessons Learned

- Specification
- Test in Stages
- Use Analysis
- Manage Test Effort
- Assess Coverage
Specification

ROS 1

- Informal specification
- Assumption:
  “It's a ROS message if it works in Python and C++”
Specification

ROS 1
- Informal specification
- Assumption: “It's a ROS message if it works in Python and C++”

ROS 2
- DDS → based on OMG IDL
- Message format itself still informal
- Transformation ROS to IDL informal
Specification

**ROS 1**
- Informal specification
- Assumption: “It’s a ROS message if it works in Python and C++”

**ROS 2**
- DDS → based on OMG IDL
- Message format itself still informal
- Transformation ROS to IDL informal

→ Is testing all existing ROS messages enough?
Specification

ROS 1
- Informal specification
- Assumption: “It’s a ROS message if it works in Python and C++”

ROS 2
- DDS → based on OMG IDL
- Message format itself still informal
- Transformation ROS to IDL informal

→ Is testing all existing ROS messages enough?
   Not in paper: Combination of fuzzing and Controllable Combinatorial Coverage.
Structure of ROS Messages: Analysis and Metrics

Implementation using Reference Attribute Grammars [Hed00] with JastAdd [EH07]

→ Analysis capabilities

Properties
- containsSubmessages
- containsUnconstrainedSubmessages
- containsUnconstrainedVariables
- containsStrings
- containsConstants
- isPartOfAction
- ...

Metrics
- nestingDepth
- numberOfDataFields
- distinctTypes
- distinctPrimitiveTypes
- distinctMessageTypes
- ...

Distribution of ROS Messages in ROS1 Noetic

+ 44 larger messages with up to 1234 (unrolled) fields
→ moveit_msgs/MoveGroupAction

Total Messages
Number of contained fields
Number of distinct types in message
Message nesting depth
Test Runtime Analysis

Runtime of Tests

- Getting ROS message **expensive**
- Constant build time

Correlation to Properties

- ROS version (*ROS2 Humble*)
- Test phase (*t*$_{\text{gen}}$
- Property
  - *(Number of contained distinct message types)*
Scatterplot: Metrics x Time in Phase

Systematic Testing of a ROS Interface Specification Backend
Johannes Mey, Ariel Podlubne, René Schöne, Paul Gottschaldt, Diana Göhringer, Uwe Aßmann
April 15th 2024
Scatterplot: Standard Packages vs All Packages
Coverage

**Problem:** Coverage of elements in templates

1. Assign a number to each text fragment and create a lookup table

<table>
<thead>
<tr>
<th>#</th>
<th>Template File</th>
<th>Pos.</th>
<th>Stack</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>template1.mustache</td>
<td>(1,1)</td>
<td>msg</td>
<td>&quot;&quot;</td>
</tr>
<tr>
<td>1</td>
<td>template1.mustache</td>
<td>(1,13)</td>
<td>\n</td>
<td>&quot;\n&quot;</td>
</tr>
<tr>
<td>2</td>
<td>template1.mustache</td>
<td>(2,12)</td>
<td>#fields</td>
<td>&quot;\n&quot;</td>
</tr>
<tr>
<td>3</td>
<td>template1.mustache</td>
<td>(3,12)</td>
<td>#fields&gt;#simple</td>
<td>&quot;\n&quot;</td>
</tr>
</tbody>
</table>
| 4  | template1.mustache | (4,10)| #fields>#simple>#axis| "\n{{name}}_tready_in when s_counter"
| 5  | template1.mustache | (5,53)| #msg>#fields>#simple>#axis>#currentMsg| "_{{currentMessage}}" |
| 6  | template1.mustache | (5,91)| #msg>#fields>#simple>#axis| "{index_tdata}" else\n" |
| 7  | template1.mustache | (6,10)| #msg>#fields>#simple| "\n" |

2. Create copy of templates replacing all fragments with just the number

   [0][[\#message]][1][\#fields][2][\#simple][3][\#axis][4][\#currentMessage][5][/currentMessage][6][/axis][7]

3. Run the test suite, obtaining number sequences

4. Aggregate all numbers, thus finding missing fragment indices

5. Identify dead code using the lookup table
Conclusion

Summary
- ~30k tests
+ High confidence in FIRM quality
+ Test systems allow expert collaboration
+ Data about ROS message landscape
  - Blocking factor specification
  - No good minimal test set yet

Opportunities and Next Steps
- Apply fuzzing and Controllable Combinatorial Coverage to generate test set
- ROS Message → OMG IDL
- Applicable to any middleware backend