Variability Analysis for Robot Operating System Applications

André Santos[1], Alcino Cunha[24], **Nuno Macedo**[34], Sara Melo[2] and Ricardo Pereira[2]

[1]VORTEX CoLab, [2]Universidade do Minho, [3]Universidade do Porto, [4]INESC TEC, Portugal

Motivation: Variability in Robotics

- Modern robotic applications are rarely made from scratch
- Reusable third-party components are configured for a purpose and integrated into a single system
- Many configuration points: component configuration, component integration, mission-specific parameters, etc.
- Enabled by middlewares such as ROS
- How can developers manage this **variability**?

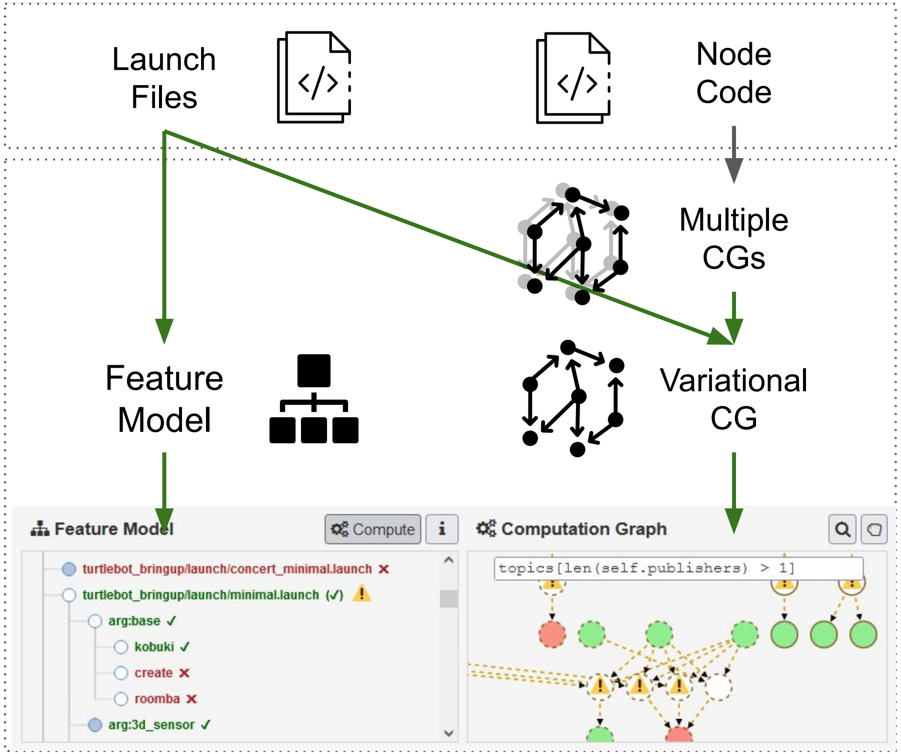
- >50 launch files
- 100s of possible configurations
- Which are valid?
- What is their impact?

Software Product Lines

- Popular approach in the software engineering community
- Family of products seen as a single artifact with variability points
- Requires *domain engineering* activities
  - **Domain analysis:** which features exist and what is their relationship?
  - **Domain design:** what is the variability-aware design of the system?
  - **Domain implementation:** how is variability resolved to obtain a product?
- Most popular approach is *feature-oriented*
  - A **feature model** determines existing features and valid configurations
  - Implementation of features may be *annotative* or *compositional*

Contributions

- Preliminary study to identify how **variability is implemented** in ROS apps
- Interpretation of **ROS applications as SPLs**
- Technique to **extract** feature models and variational architectures
- Tool to interact with these artifacts and aid in **product configuration**



ROS in a Nutshell

- ROS package: node source code + launch files
- ROS application: has an associated runtime computation graph
  - Running nodes
  - Communication channels
  - Parameters
- Determined by the selected set of *launch files* and their *arguments*

Study on Variability in ROS

- We consider a particular **product**/application a concrete CG
- A **feature** is configuration point of launch process that affects the CG
- We *do not* consider variability in node behaviour
- Several popular open-source ROS robots analyzed
- We found both compositional (preferred) and annotative strategies

Turtlebot2

Lizi

Husky

Strategies to Implement Variability in ROS

Compositional:

- multiple launch files for each functionality
- to launch an application, select a subset

```
<launch>
  <node name="n" type="A" .../>
</launch>
```

```
<launch>
  <node name="n" type="B" .../>
</launch>
```

Annotative:

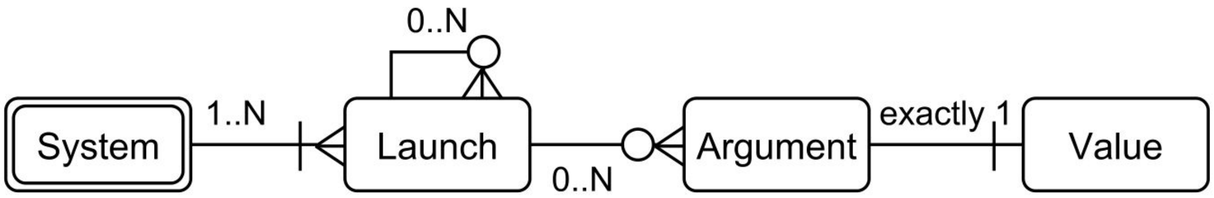
- conditional behaviour inside launch files
- arguments used in conditional blocks
- names of other launch files to include
- topic names in remaps

```
<launch>
  <arg name="x"/>
  <node name="n" type="A" ... if="$(arg x)"/>
  <node name="n" type="B" ... unless="$(arg x)"/>
</launch>
```

ROS Feature Models

- The selection of a **launch file** is a feature
- *Dependencies*: launch files including others
- *Conflicts*: launching nodes with same name

- Assigning a value to a **CG-affecting arguments** is a feature
- *Dependencies*: assigning an argument requires its launch file
- *Optional*: if there is a default value
- *XOR-group*: possible values of an argument



ROS SPL Example: Feature Model

simple.launch

```
<launch>
  <arg name=”tele” default=”True”/>
  <include file=“tele.launch” if=”$(arg tele)”/>
  <node name=”base” type=”base” .../>
</launch>
```

base.launch

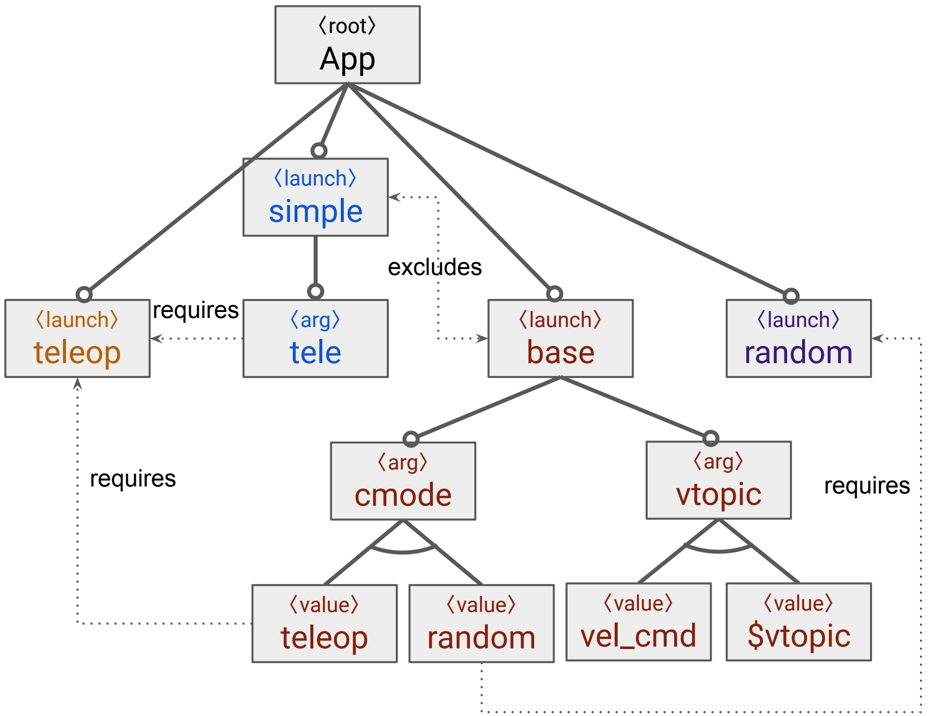
```
<launch>
  <arg name=”cmode” default=”teleop.launch”/>
  <arg name=”vtopic” default=”vel_cmd”/>
  <include file=“$(arg cmode)”/>
  <node name=”base” type=”base” ...>
    <remap from=”vel” to=”$(arg vtopic)”/>
  </node>
</launch>
```

teleop.launch

```
<launch>
  <node name=”tele” type=”teleop” .../>
</launch>
```

random.launch

```
<launch>
  <node name=”random” type=”random” .../>
</launch>
```



ROS Variational CGs

- Each element of the CG is assigned a **presence condition**
- Proposition whose variables are features from the feature model
  - if a node is launched by `a.launch`, condition is the respective feature
- Similar elements are merged and their conditions simplified
  - if a node is launched by `a.launch` and `b.launch`, condition is their disjunction
  - nodes always present eventually have presence condition *true*

ROS SPL Example: Variational CG

simple.launch

```
<launch>
  <arg name=”tele” default=”True”/>
  <include file=“tele.launch” if=”$(arg tele)”/>
  <node name=”base” type=”base” …/>
</launch>
```

base.launch

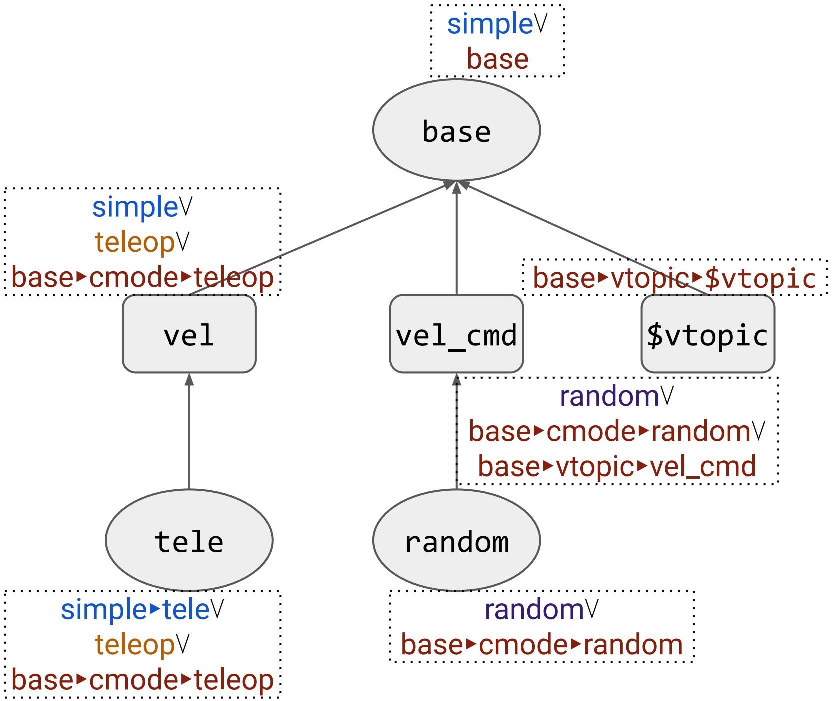
```
<launch>
  <arg name=”cmode” default=”teleop.launch”/>
  <arg name=”vtopic” default=”vel_cmd”/>
  <include file=“$(arg cmode)”/>
  <node name=”base” type=”base” …>
    <remap from=”vel” to=”$(arg vtopic)”/>
  </node>
</launch>
```

teleop.launch

```
<launch>
  <node name=”tele” type=”teleop” …/>
</launch>
```

random.launch

```
<launch>
  <node name=”random” type=”random” …/>
</launch>
```



ROS Feature Model Extraction

- Processes launch files
- Identifies CG-affecting arguments
  - conditionals, included files, remap names, node attributes
- Identifies possible values of CG-affecting arguments
  - identifies Boolean values
  - if path to file, identifies acceptable string values
  - other strings left as user-provided value
  - considers default values
- Identifies incompatible launch files
  - nodes with the same name
  - may be conditional on passed arguments

ROS Variational CG creation

- Builds on previous work extracting CG from ROS applications
  - static source code analysis of node code
- Elements whose presence could not be determined statically are optional
- These are now attached presence conditions, identified when extracting the Feature Model

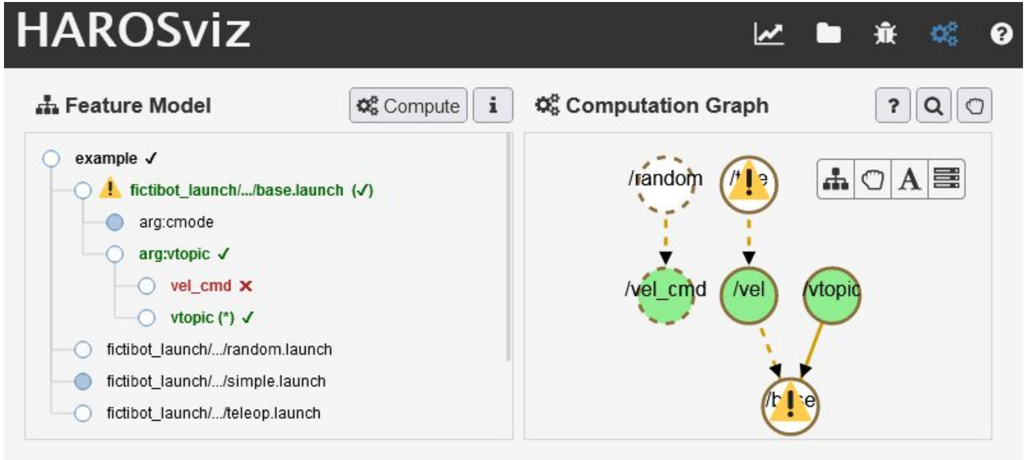
HAROS

- Analysis engine for ROS applications
- Static analysis over ROS source code
- Model extraction from code and launch files
- Plug-ins run analyses over code and extracted models
    - Pattern matching
    - Runtime verification
    - Automated test generation

https://github.com/git-afsantos/haros

HAROS SPL Plug-in

- SPL extraction technique in the backend
- Depicts the extracted feature model and variational CG
- *Multi-step configuration*: feature dependencies and CG iteratively resolved
- Once CG is fully resolved, provides the respective *launch command*




```
roslaunch example base.launch cmode:=tele vtopic:=vtopic
```

Evaluation: Launch Features

- Applied our SPL extraction technique to 4 realistic robots

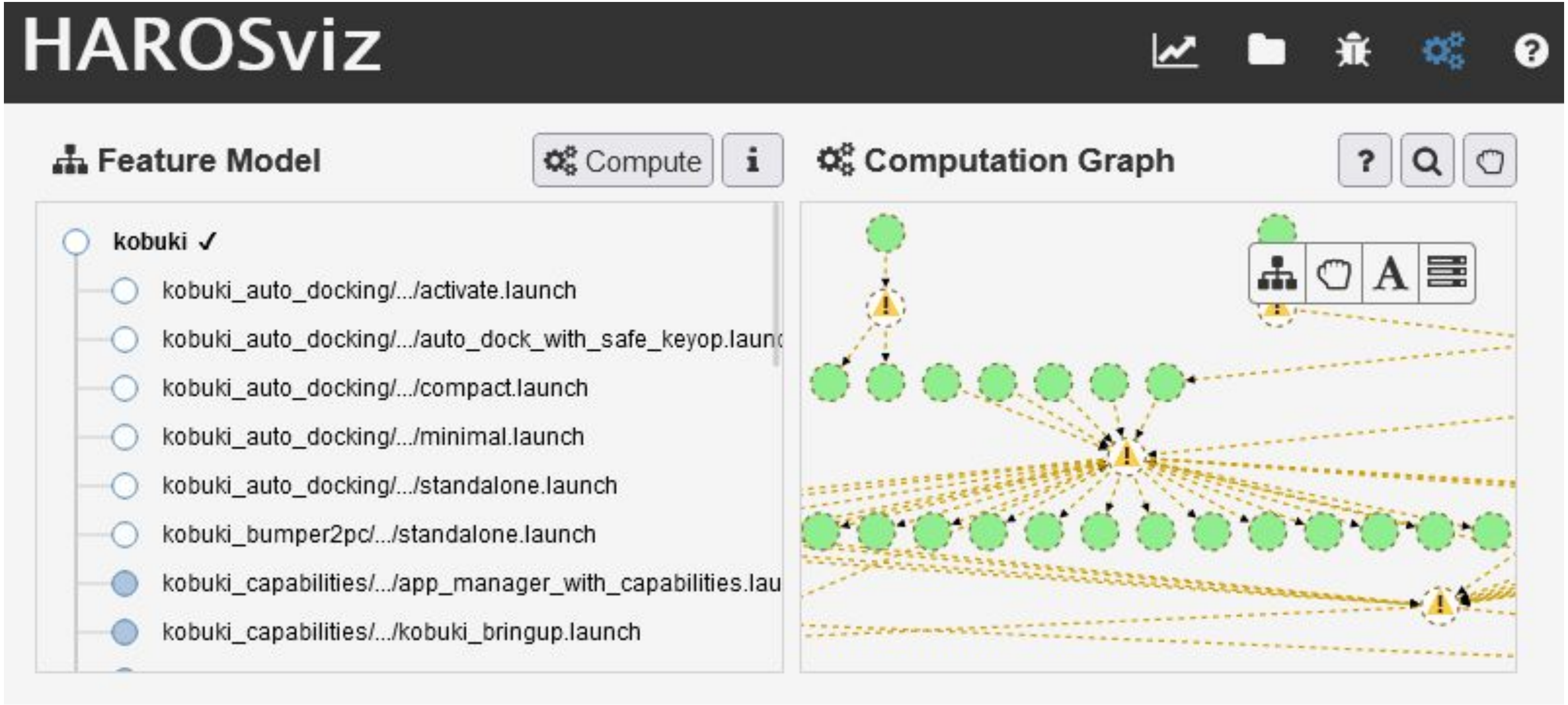
System	Launch Files	Always Compatible	Always Incompatible	Maybe Incompatible
Kobuki	21	4	17	0
TurtleBot2	53	15	36	2
Lizi	14	2	12	0
Husky	137	37	98	2

Evaluation: Argument Features

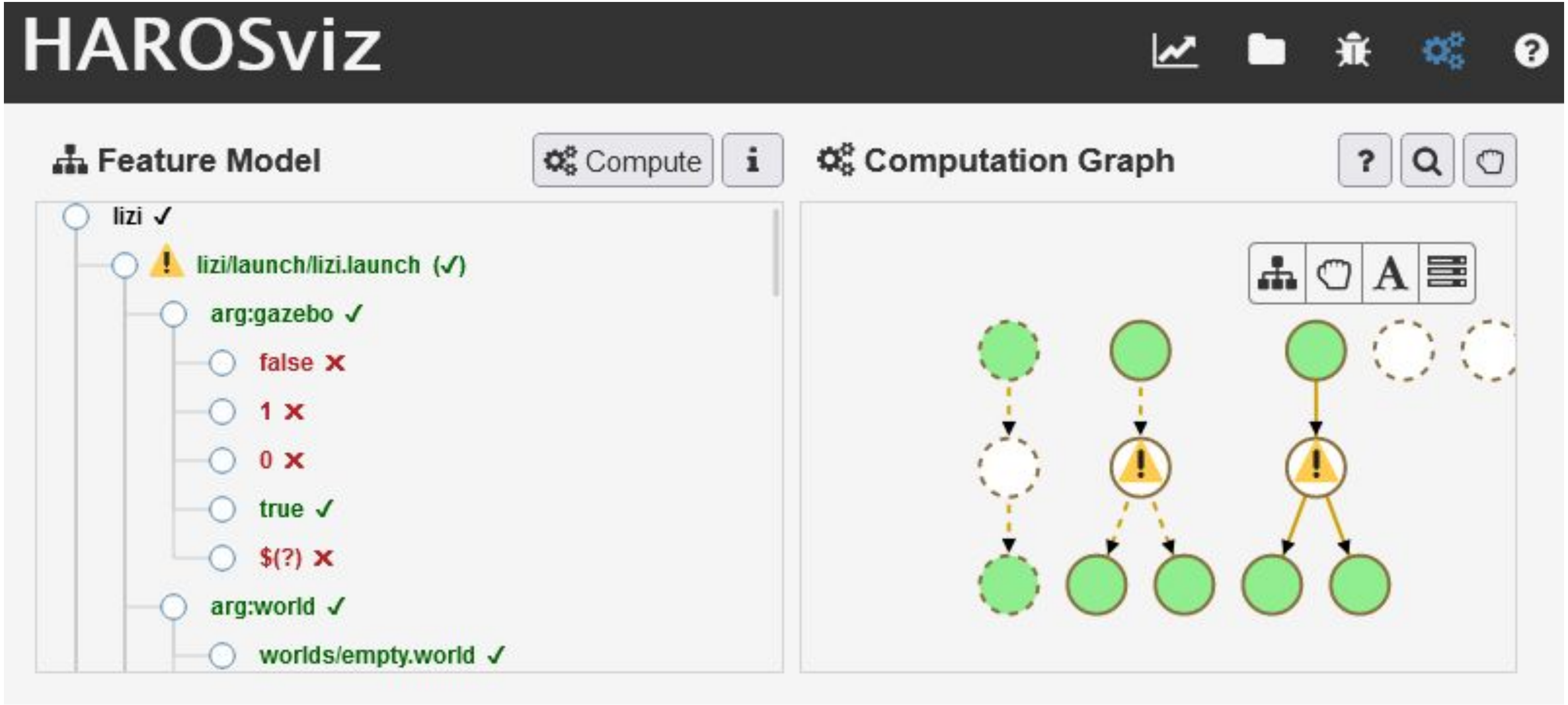
- Applied our SPL extraction technique to 4 realistic robots

System	Arguments	CG-affecting Arguments	Non-Bool CG-affecting	Non-Bool Computed	Conditional Blocks	Maximum Occurrences
Kobuki	18	0	0	0	0	0
TurtleBot2	195	23	19	5	6	2
Lizi	66	12	1	0	20	4
Husky	391	82	40	12	57	4

Evaluation: Kobuki



Evaluation: Lizi



Evaluation

- Does the execution time of the technique scale for realistic ROS systems?
    - We did not find any performance bottlenecks
- How effective is the technique in automatically extracting feature models?
    - Total precision, but not complete in detecting valid argument values
    - Tool allows manual definition of values during configuration
- How complex are the SPLs of realistic ROS systems?
    - Several conflicting launch files that the user may not be aware of
    - A lot of CG-affecting arguments without documentation,
    - An argument often affects multiple points

Conclusions

- Variability is pervasive in ROS, with different implementation strategies
- Interpretation of ROS applications as SPL: feature model + variational CG
- We propose a technique to extract such SPLs and integrate it in HAROS
- Evaluation shows features cause conflicts and affect CG in multiple points

- Continue to tackle limitations of the underlying CG extraction (ROS2, Python)
- Extend existing analysis for the variational context (analyse whole SPL)