SkiROS2

A skill-based Robot Control Platform for ROS

Matthias Mayr, Lund University & WASP
matthias.mayr@cs.lth.se
A platform for Intelligent and Autonomous Robots
What is a skill?

“What Skills [...] are parametric procedures that modify the world model (world state), bringing it from an initial state to a final state according to their pre- and post-conditions.”

1 https://github.com/RVMI/skiros2/wiki/Overview-3:-Skill-model
Skill Model
Skill Model

Skill Description

- Semantic level
- Parameters
- Pre-, hold- and post-conditions

```python
class Drive(SkillDescription):
    def createDescription(self):
        # ======Params========
        self.addParam("Robot", Element("cora:Robot"), ParamTypes.Required)
        self.addParam("TargetLocation", Element("skiros:location"), ParamTypes.Required)
        self.addParam("Velocity", 0.5, ParamTypes.Optional)
        self.addParam("StartLocation", Element("skiros:Location"), ParamTypes.Inferred)
        # ======PreConditions=====
        self.addPreCondition(self.getRelationCond("RobotAt", "skiros:at", "Robot", "StartLocation", True))
        # ======PostConditions=====
        self.addPostCondition(self.getRelationCond("NoRobotAt", "skiros:at", "Robot", "StartLocation", False))
        self.addPostCondition(self.getRelationCond("RobotAt", "skiros:at", "Robot", "TargetLocation", True))
```

Skill Implementation

- Implements one description
- Different implementations of one description
- Can modify the description

```python
class drive_fade(SkillBase):
    def createDescription(self):
        self.setDescription(Drive(), self.__class__.__name__)

    def expand(self, skill):
        skill.setProcessor(SerialStar())
        skill[
            self.skill["wait", "wait", specify={"Duration": 1.0}],
            self.skill["wmsSetRelation", "wms_set_relation", remap={'Src': 'Robot', 'Dst': "StartLocation"},
                            specify={'Relation': 'skiros:at', 'RelationState': False}],
            self.skill["wmsSetRelation", "wms_set_relation", remap={'Src': 'Robot', 'Dst': "TargetLocation"},
                            specify={'Relation': 'skiros:at', 'RelationState': True}]
        ]
```
Skill Description

- Parameters
  1. Required
  2. Optional
  3. Inferred

- Conditions
  1. Preconditions
  2. Holdconditions
  3. Postconditions

- Condition Types
  1. Relation Condition
  2. Property Existence
  3. Property Value

```python
class Drive(SkillDescription):
    def createDescription(self):
        # ======Params========
        self.addParam("Robot", Element("cora:Robot"), ParamTypes.Required)
        self.addParam("TargetLocation", Element("skiros:Location"), ParamTypes.Required)
        self.addParam("Velocity", 0.5, ParamTypes.Optional)
        self.addParam("StartLocation", Element("skiros:Location"), ParamTypes.Inferred)
        # ======PreConditions==========
        self.addPreCondition(self.getRelationCond("RobotAt", "skiros:at", "Robot", "StartLocation", True))
        # ======PostConditions========
        self.addPostCondition(self.getRelationCond("NoRobotAt", "skiros:at", "Robot", "StartLocation", False))
        self.addPostCondition(self.getRelationCond("RobotAt", "skiros:at", "Robot", "TargetLocation", True))
```
Skill Implementations: **Primitive Skills**

- Semantically atomic actions
- Typically directly interact with an API
- Examples:
  - Gripper actuation
  - Arm manipulation
  - Sensor input

**Code Skeleton:**

- Implement one skill description
- Python functions for start, execution, ...
- Return “running”, “success” and “failure”

```python
class myPrimitive(PrimitiveBase):
    def createDescription(self):
        """Set the primitive type""
        self.setDescription(MyPrimitive())

    def onInit(self):
        """Called once when loading the primitive. If return False, the primitive is not loaded""
        return True

    def onPreempt(self):
        """Called when skill is requested to stop.""
        pass

    def onStart(self):
        """Called just before first execute"
        return True

    def onEnd(self):
        """Called just after last execute"
        pass

    def execute(self):
        """Main execution function"
        return self.success("Done")
```
Skill Implementations: **Compound Skills**

- Combine several compound skills and primitives
- *Extended Behavior trees*
- Processors
  - Serial (AND)
  - Selector (OR)
  - Parallel
  - …
- Automatic selection of implementations

### Compound Skill Implementation:

```python
class drive_fake(SkillBase):

class drive_platform(SkillBase):

class drive_platform(SkillBase):

def createDescription(self):
    self.setDescription(Drive), self.__class__.__name__

    def expand(self, skill):
        skill.setProcessor(SerialStar())
        skill(
            self.skill(SelectorStar())
            self.skill("MovePlatformDirect", "", specify={"Velocity": self.params["Velocity"][values]},
            self.skill("MovePlatformPlanning", "", specify={"Velocity": self.params["Velocity"][values]},
        self.skill("VerifyPlatformArrival", ""),
        self.skill("WmSetRelation", "wm_set_relation", remap={'Src': 'Robot', 'Dst': 'StartLocation'},
        specify={'Relation': 'skiros:at', 'RelationState': False}),
        self.skill("WmSetRelation", "wm_set_relation", remap={'Src': 'Robot', 'Dst': 'Targetlocation'},
        specify={'Relation': 'skiros:at', 'RelationState': True})
```
SkiROS2 Architecture
World Model

- Stores knowledge in an RDF graph
- Ontologies
  - Concepts
  - Properties
  - Relations
- Scene has concrete instances
- Enables reasoning and planning

```python
class Drive(SkillDescription):
    def createDescription(self):
        # ======Parameters======
        self.addParam("Robot", Element("cora:Robot"), ParamTypes.Required)
        self.addParam("TargetLocation", Element("skiros:Location"), ParamTypes.Required)
        self.addParam("Velocity", 0.5, ParamTypes.Optional)
        self.addParam("StartLocation", Element("skiros:Location"), ParamTypes.Inferred)
        # ======PreConditions=====
        self.addPreCondition(self.getRelationCond("RobotAt", "skiros:at", "Robot", "StartLocation", True))
        # ======PostConditions=====
        self.addPostCondition(self.getRelationCond("NoRobotAt", "skiros:at", "Robot", "StartLocation", False))
        self.addPostCondition(self.getRelationCond("RobotAt", "skiros:at", "Robot", "TargetLocation", True))
```

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>skiros:Container</td>
<td>rdfs:subclassOf</td>
<td>skiros:Location</td>
</tr>
<tr>
<td>skiros:DriverAddress</td>
<td>rdfs:subPropertyOf</td>
<td>skiros:DeviceProperty</td>
</tr>
<tr>
<td>skiros:Scene-0</td>
<td>skiros:contains</td>
<td>skiros:Location-1</td>
</tr>
<tr>
<td>skiros:Robot-2</td>
<td>skiros:at</td>
<td>skiros:Location-1</td>
</tr>
</tbody>
</table>
Skill Manager

- Loads skills from skill libraries
- Populates the world model with skill information
- Executes skills
  - Creates a task
  - Skills share a blackboard
  - Grounds skills
  - Automatically selects skills
Task Manager for Task-Level Plans

- Receives planning goal such as
  \[(\text{skiros:at skiros:Robot-2 skiros:Location-3})\]
- Automatically creates a PDDL planning domain
  - Based on the knowledge in the world model
- Uses a PDDL planner (tfd)
- Execution in the skill manager
Integration and GUI

● Turn ROS Actions into skills

● GUI
  ○ Start, tick & stop skills
  ○ Debug skill execution
  ○ View and modify the world model

● Python API
  ○ World model access
  ○ Skill manager

● tf-frames and RViz integration
  ○ Couple frames to world model entities
  ○ Publish tf frames
Example Use Cases
SkiROS2 - Summary

- Flexible robot control platform
- Targeted for semi-structured environments
- Knowledge integration and reasoning
- Automatic task-level planning
- Behavior trees
- Reinforcement learning
- ROS 2 support

What will you do with it?

Documentation:
https://github.com/RVMI/SkiROS2/wiki

Paper:
SkiROS2: A skill-based robot control platform for ROS
https://arxiv.org/abs/2306.17030

{Code}:
https://github.com/RVMI/SkiROS2