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?

“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_fake(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("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})
        }
```
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("RobotAt", "skiros:at", "Robot", "TargetLocation", False))
```
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(self, 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 1st 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

```python
class drive_fake(SkillBase):
    def __init__(self):
        self.set_description("Drive")

class drive_platform(SkillBase):
    def __init__(self):
        self.set_description("Drive")

    def create_description(self):
        self.set_description("Drive")

    def expand(self, skill):
        skill.set_processor(SerialStar())
        skill = self.skill(SerialStar())
        skill = self.skill("MovePlatformDirect", "", specify={"Velocity": self.params["Velocity"]["values"]})
        skill = self.skill("MovePlatformPlanning", "", specify={"Velocity": self.params["Velocity"]["values"]})

        skill = self.skill("VerifyPlatformArrival", "")
        skill = self.skill("WmSetRelation", "wm_set_relation", remap={"Src": "Robot", "Dst": "StartLocation"}, specify={"Relation": "skiros:at", "RelationState": False})
        skill = 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))
```

Subject | Predicate | Object
--- | --- | ---
`skiros:Container` | `rdfs:subclassOf` | `skiros:Location`
`skiros:DriverAddress` | `rdfs:subPropertyOf` | `skiros:DeviceProperty`
`skiros:Scene-0` | `skiros:contains` | `skiros:Location-1`
`skiros:Robot-2` | `skiros:at` | `skiros:Location-1`
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

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

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