2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 1-5, 2023. Detroit, USA

SkiROS2: A skill-based Robot Control Platform for ROS

Matthias Mayr[1], Francesco Rovida[2] and Volker Krueger[1]

2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) | 978-1-6654-9190-7/23/$31.00 ©2023 IEEE | DOI: 10.1109/IROS55552.2023.10342216

***Abstract*— The need for autonomous robot systems in both the service and the industrial domain is larger than ever. In the latter, the transition to small batches or even "batch size 1" in production created a need for robot control system architectures that can provide the required flexibility. Such architectures must not only have a sufficient knowledge integration framework. It must also support autonomous mission execution and allow for interchangeability and interoperability between different tasks and robot systems. We introduce *SkiROS2*, a skill-based robot control platform on top of ROS. SkiROS2 proposes a layered, hybrid control structure for automated task planning, and reactive execution, supported by a knowledge base for reasoning about the world state and entities. The scheduling formulation builds on the extended behavior tree model that merges task-level planning and execution. This allows for a high degree of modularity and a fast reaction to changes in the environment. The skill formulation based on pre-, hold- and post-conditions allows to organize robot programs and to compose diverse skills reaching from perception to low-level control and the incorporation of external tools. We relate SkiROS2 to the field and outline three example use cases that cover task planning, reasoning, multisensory input, integration in a manufacturing execution system and reinforcement learning.**

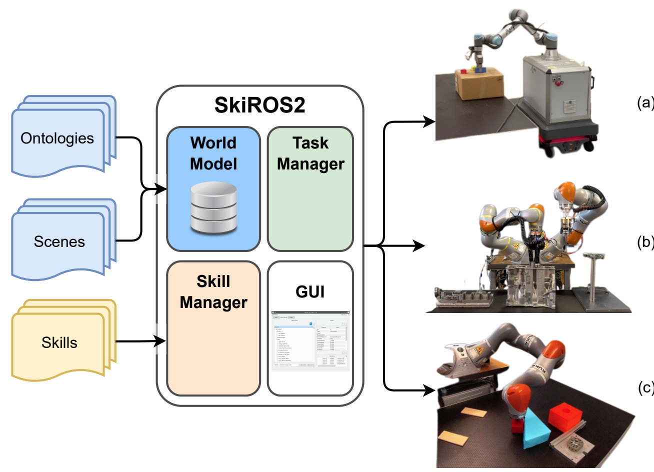

Fig. 1. SkiROS2 can be used with different ontologies, scene descriptions and skills to solve a variety of tasks with different robot platforms. (a): Task-level planning with the mobile manipulation of objects. (b): Precise and sensitive piston insertion. (c): Learning to push by combining planning, reasoning and reinforcement learning.

I. Introduction

Modern intelligent robots require an increasing system complexity in order to perform the increasingly complex tasks demanded from them. Especially in view of greater autonomy, this complexity needs to be matched by the system architecture used to program and control the robots. With more and more integrated systems that coordinate different partial solutions, there is a need for interoperability and a common framework for communication, control and task planning.

In the industrial robotics field this can be seen in the Industry 4.0 movement that advocates such a transition, but several aspects of current practice present barriers to it. Many robot control systems currently rely heavily on "implicit" knowledge representation. Typically this is implemented by using if-else statements in the actual code. This often inhibits the growth of a control platform, as well as the interchangeability and interoperability between different tasks or robot systems, as this knowledge is hidden and often only known to the programmer herself/himself. Furthermore, vendor lock-in into the robot programming platform of a specific manufacturer is widespread.

These barriers have also been identified in the past and early platforms such as *ClaraTy* [1] or *LAAS* [2] provided first architectures. For knowledge integration frameworks, the system around the *Rosetta* ontology [3], [4] and *Knowrob* [5], [6] created a strong foundation. However, the former is not publicly available and the latter targets the different needs of service robotics.

In this context, we introduce *SkiROS2*, a skill-based robot control framework for ROS. It utilizes knowledge representation in a Resource Description Framework (RDF) graph that supports an open and explicit formulation of knowledge. The skill model is based on *pre-conditions* that are checked before a skill is executed, *hold-conditions* that also need to be satisfied while the skill is running, and *post-conditions* that are checked after the skill execution. SkiROS2 supports reasoning to infer skill parameters, and it allows the implementation and integration of custom reasoners, such as a spatial reasoner. Built-in task planning allows it to utilize robot capabilities to automatically construct a planning domain and problem description without manual input from a domain expert. Furthermore, it is capable of orchestrating multiple robot systems. SkiROS2 is open source, written in Python and based on behavior trees (BTs). It is the successor platform of SkiROS1 [7], [8] that was written in C++ and did not use BTs.

In this paper, we discuss the requirements for our system

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by Knut and Alice Wallenberg Foundation.

[1]Department of Computer Science, Faculty of Engineering (LTH), Lund University, SE 221 00 Lund, Sweden. E-mail: <firstname>.<lastname>@cs.lth.se.

[2] RiACT ApS, A.C. Meyers Vaenge 15, 2450, Copenhagen, Denmark. Email: f.rovida@riact.eu

architecture and assess the state-of-the-art in the field and we show how our solution SkiROS2 fulfills these requirements. We introduce its individual modules such as the skill manager, the task manager and the world model.

We think that this approach establishes a modern system architecture for intelligent autonomous robot systems that allows to design and organize robot skills for advanced tasks.

II. Background and Related Work

This section introduces related work on the axes of robot control strategies, cognitive approaches as well as knowledge integration. Finally, a brief comparison with other platforms is provided.

A. Robot Control Strategies

Three control strategies dominate the research community: deliberative, reactive, and hybrid control [9].

1) Deliberative Systems: This architecture follows the "sense, plan, act" paradigm. An observation of the world is translated into a symbolic representation [9]. Reasoning is then utilized to plan a sequence of actions. While the actions are executed, new observations are not taken into account. This typically leads to slower reaction times, since the sequence of sensing and planning can often be time-consuming.

2) Reactive Systems: Reactive systems utilize a concurrently running finite state machines (FSM). These FSMs directly connect the input to the output. An important distinction is that they do not build up an internal representation. Their extension, behavior-based systems, also allows to assign priorities to behaviors and therefore inhibit output of low-priority ones.

3) Layered Hybrid Systems: In order to utilize the advantages of both aforementioned systems, reactivity and planning, layered hybrid systems were formulated. Typically, a higher planning level determines a mid- or long-term strategy, and a lower level is able to react directly to observations. Most often a synchronization layer is added to bridge between high-level reasoning and low-level control. Most modern robot control systems follow this approach.

B. Cognitive Systems

Besides having an architecture that allows tasks to be performed, another important question is what cognitive abilities are necessary and how these abilities are structured. Cognition can be defined as the process of a system to perceive the environment, act to pursue goals, anticipate the outcome of events, adapt the actions to changing circumstances and learn from experiences.

The *cognitivist approach* performs processing to obtain a symbolic knowledge representation [9], [10]. This abstracted symbolic representation of the world allows to reason about it. The representation is typically designed and interpreted by humans, which means that it can be understood and complemented with knowledge from other sources.

In contrast to that, the *emergent paradigm* means an inductive organization from observations [10]. Machine learning techniques and the automatic induction of ontologies are part of this field. It comes at the cost of being dependent on experiences.

Hybrid approaches aim to combine both approaches, explicit knowledge representation and learning from experiences [10].

For our work, the cognitivist approach is the most relevant. However, a recent line of research extends it to a hybrid approach by incorporating learning from experiences into the system [11], [12], [13].

C. Knowledge Integration

An important aspect of autonomous systems is the type of knowledge integration. It decides on the support for reasoning and inference, the possibility of integrating encyclopedic knowledge, and the expressiveness. The latter should be weighted against the efficiency of the reasoning algorithms [9].

The available formalisms in the field of robotics include model logic, temporal logic, and predicate logic such as Prolog and Golog. Description logics are widely used to define ontologies. In particular the Web Ontology Language (OWL) that is based on the Resource Description Framework (RDF) gained a lot of attention. A recent review and comparison of ontology-based approaches is presented in [14]. Projects such as *Knowrob* [5], [6] created a strong foundation in the field of service robotics, while, for example, *Rosetta* [3] is explicitly designed for the industrial domain.

D. Robot Control Platforms

We relate our work to other frameworks that are available as open-source software or provide clear guidelines for their task planning, task execution, and knowledge integration. Table I presents them along with criteria such as the application area, knowledge modeling and task planning. Most of the related frameworks explicitly target the industrial domain [3], [4], [15], [16] and [17]. Others, such as *Knowrob + CRAM* [5], [6] or *CAST* [18] address the different needs of service robotics instead. Task-level planning is implemented by most of the frameworks. COSTAR [15] implements visual neural network-based planning [19] while [20] offers an interface that can be used by planners. For the scheduling we see that only few implement reactive methods such as BTs. As a middleware, most frameworks use the Robot Operating System (ROS), which is advantageous for creating interoperability and using network effects. Many, but not all platforms are available as open-source software for other researchers to study, use, and improve. The comparison with existing frameworks shows that SkiROS2 has a unique combination attributes that enables it to be a modern autonomous robot control platform.

E. Behavior Trees

A behavior tree (BT) is a formalism for the representation and execution of procedures. BTs have emerged in the gaming industry, but are also becoming widespread in robotics [22]. A BT is a directec acyclic and rooted

TABLE I

A COMPARISON OF DIFFERENT ROBOT CONTROL PLATFORMS. THE EXISTENCE OR LACK OF A FEATURE IS SHOWN WITH ✓ AND ✗. THE "-" INDICATES A LACK OF INFORMATION.

Project Name/Group	Application	Knowledge Modeling	Task Planning	Scheduling	Middleware	Open Source
SkiROS2	Industrial	OWL-DL	PDDL	eBT	ROS	✓
COSTAR [15]	Industrial	-	Visual	BT	ROS	✓
GTax [16]	Industrial	SysMl	PDDL	SysMl	-	✗
Balakirsky et al. [17]	Industrial	OWL-DL	PDDL	CRCL	ROS	✗
Stenmark et al. [3], [4]	Industrial	OWL-DL	PDDL	State charts	-	✗
Knowrob/CRAM/EASE [5], [6]	Service	Prolog, OWL-DL	CPL	CPL	ROS	✓
CAST [18]	Service	Proxies	MAPL	MAPL	BALT	partially
LAAS [2]	General	-	lxTeT	Open-PRS	Bip/GenoM	partially
ClaraTy [1]	General	-	Corba	TDL	-	✓
SmartMDSD [21]	General	-	SmartTdl	SmartTdl	SmartSoft	✓
FlexBE [20]	General	✗	(Synthesis Interface)	FSM	ROS	✓

graph consisting of nodes and edges [23]. The root node is used to periodically inject an enabling signal called *tick* that traverses through the tree according to the conditions, state of skills and their connectors called *processors*. These processors allow to link child nodes in different procedural ways. Examples are a sequence (logical AND) or a selector (logical OR). When the *tick* reaches a leaf node, it executes one cycle of the action or condition. Actions can modify the system configuration and return one of the three signals *success*, *running* or *failure*. Condition checks are atomic and can only return *success* or *failure*. To pass information between different nodes, a common approach is to use a set of shared variables on a *blackboard*. For a full formalization of BTs in the context of robotics, we refer to [23].

The classical BT formulation is complemented by a formalism to define *extended behavior trees (eBTs)* in [24]. In eBTs, scripted and planned procedures are merged into a unified representation, so that an eBT describes both the execution and its effects on the world state. This is achieved by combining the flexibility of BTs with hierarchical task network (HTN) planning. In contrast to classical BT, the pre- and post-condition nodes are embedded into the eBT to use them for task-level planning. To achieve real modularity, eBT allows procedural abstraction by allowing different implementations for the same type of action. Additional preconditions then allow to choose the right implementation to use at run-time. For example, select a different opening strategy depending if you need to open a manual door or an automatic one that opens with a button. Furthermore, the eBT formalism allows to optimize the execution of the planned sequence [24].

III. DESIGN CONSIDERATIONS

On the basis of common applications in the field and our own experience, we identified the relevant requirements and design considerations. This section presents and discusses them in the context of robot control systems.

A. Control Strategy

Out of the main branches of robot control structures introduced in Section II-A, the layered hybrid approach is the most applicable to the target domain: longer-term task goals require planning and reasoning while the execution should be able to react quickly to observations from the real world. Therefore, SkiROS2 uses a deliberate planning level and a lower-level implementation with BTs [23].

B. Multi-Robot Orchestration

Challenging modern Industry 4.0 tasks are rarely content with a single robot. In addition, because of the importance of collaboration with humans, a robot control platform needs to be able to orchestrate several actuators simultaneously. In a multi-robot setup it is important that all actuators maintain a coherent world state to prevent failures and synchronization problems. SkiROS2 allows to start an arbitrary number skill managers for different robot systems that can share a single world model (WM), thus being able to simultaneously orchestrate a fleet of robots that have a common understanding of the world.

C. Knowledge Representation

To avoid an implicit representation of knowledge, for example in the form of "if, else" statements in code, it is important to have means to *explicitly* formulate and organize knowledge. Autonomous robot systems can also benefit from knowledge integration for abstract reasoning and especially in the industrial robotics domain, structured knowledge is often available. There are many logic formalisms and the choice needs to balance *expressivity* and *efficiency*. Furthermore, we find the usability and availability of ontologies, the set of concepts and their relation in a target domain, to be important. Therefore, SkiROS2 uses the OWL and a set of established ontologies such as the *Core Ontology for Robotics and Automation (CORA)*.

D. Manufacturing Execution Integration

A platform for industrial robot skill execution needs to be able to interact with higher-level systems such as a manufacturing execution system (MES). Although the manual or automated start of individual skills is important, the concept of a skill-based platform really excels when higher-level goals can be sent to robot control systems [25]. A robot control system then needs to plan and execute autonomously. Furthermore, the execution state and modification of a shared understanding of the world such as a WM or a digital twin (DT) must be reported back [25].

E. Stakeholders

Robot control systems have different stakeholders that need to be addressed. Besides special needs for a vertical integration described in the previous section, users also have different expectations and needs. Most systems differentiate between developers and end users. Developers are assumed to have an in-depth understanding of the matter and need tools to support their processes. With a robot control platform, it can be assumed that developers design and implement skills. They can also understand and form ontologies. On the other hand, end users need lower entry hurdles. In the robotics context, this usually means addressing them with a graphical user interface (GUI) that abstracts away the underlying processes.

F. Middleware

Finally, the middleware as a communication system and application programming interface (API) are an important choice for every larger software project. The robotics domain with its different actuators, sensors and software solutions is particularly diverse and support for a wide variety of robotic systems can easily surpass the capabilities of companies and research groups.

Although there are other robotic middleware systems such as *OROCOS* [26], the Robot Operating System (ROS) [27] became a commonly used and increasingly popular middleware solution for robotic systems and their programs. The ROS libraries are a good basis for integrated robot systems since they provide a standard communication and programming interface. Therefore, based on ROS, a control system such as SkiROS2 gets immediate access to a wide range of tools, drivers and trained users.

IV. Architecture of SkiROS2

This section outlines the architecture of the system shown in Fig. 2. The *skill manager* and the *world model (WM)* form the core. One or more task managers can be used to accept high-level goals. During the prototyping phase, a GUI supports the users.

A. Skill Model

In this section we introduce the SkiROS2 skill model with its components and types of skills. We define a skill as a parametric procedure that changes the world from some initial state to some new state [28]. This definition deliberately leaves room for a wide range of skills from deep learning-based object localization to lower-level motor control. The skill execution flow is shown in Fig. 3. In SkiROS2 skills can have different complexities: we refer here to primitive (atomic) skills and compound skills. The latter consist of any amount of primitive or compound skills.

Skill Description: The skill description defines the actions of a skill on a semantic level. A skill description

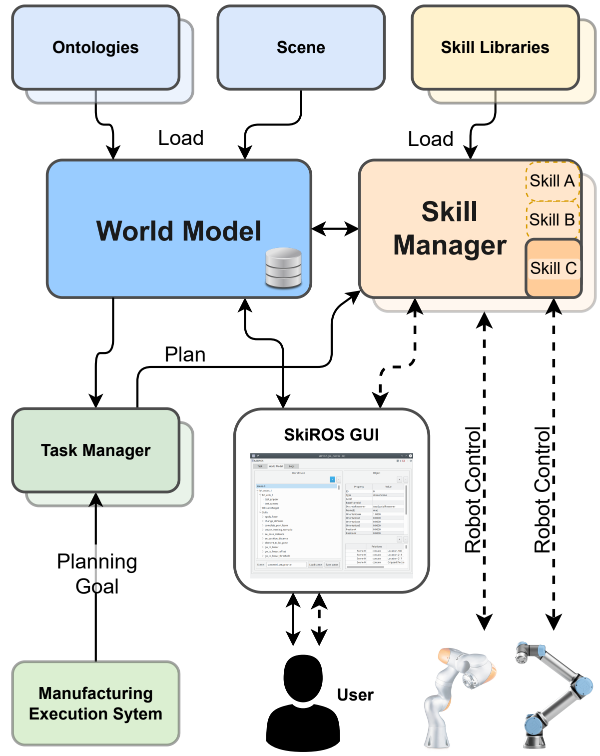

Fig. 2. An outline of the SkiROS2 architecture. The world model stores the knowledge about the relations, environment and the skills. The skill manager loads and executes the skills. Dashed lines show control flows and solid lines information flows. Shaded blocks indicate possible multiple instances.

includes its zero or more parameters and zero or more pre-, hold- and post-conditions. An example is shown in Lst. 1. Both primitive and compound skills always implemented exactly one skill description. However, an implementation is allowed to complement or further specify a skill description. This paradigm is useful to have multiple skill implementations for different hardware or specific scenarios. As an example for this, we can take a gripper actuation skill. A simple general skill description can be universal and have a boolean parameter "OpeningState" as well as a "Gripper" parameter that refers to a concept in the WM such as `Element("rparts:GripperEffector")`. Different gripper hardware needs different implementations of this description. To enable this, a skill implementation for a specific gripper can then modify the skill description of the parameter "Gripper" to handle only a single type of gripper that would be a subtype of the concept "rparts:GripperEffector", such as "scalable:RobotiqGripper". When executing a compound skill that utilizes this gripper actuation skill description, SkiROS2 will automatically select the matching implementation.

Skill Parameters: The skill parameters define the input and output of a skill. Furthermore, they can also be used for reasoning together with pre- and post-conditions. Parameters come in three flavors: 1) required, 2) optional, and 3) inferred. In contrast to the optional parameters, the required parameters must be set to execute a skill. The inferred

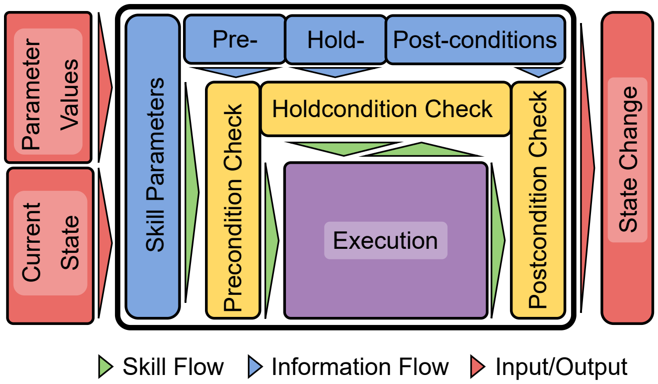

Fig. 3. The conceptual model of a skill in SkiROS2. Pre-, and hold-conditions ensure that the skill is only executed in the correct world state. Post-conditions check if the desired changes have been achieved.

parameters do not need to be set, but can automatically be reasoned about with the pre- and post-conditions. As an example, the *pick* skill in Lst. 1 has a required parameter *Arm*, could have an optional parameter *arm maximum velocity* and has an inferred parameter *Gripper*. If the parameter *Arm* is known, we can utilize the relations in the WM to automatically know which gripper is attached to it. The optional parameter for the maximum arm velocity can be used to overwrite a default value.

Skill parameters have either a fundamental data type such as `string`, `int`, `float`, `bool`, `list`, `dict` or are WM element in the form `Element("schema")`. The *schema* refers to a concept that is defined in the ontologies, and during execution it is grounded to a specific instance of that concept. This way its associated attributes and relations can be utilized by the skills. In the *pick* example above, the parameters *Gripper* and *Arm* would be WM element types `Element("rparts:GripperEffector")` and `Element("scalable:Arm")`.

Pre-, Hold- and Post-conditions: Preconditions describe the necessary world state under which a skill is to be executed. Furthermore, they can be used to infer parameters of a skill such as the parameters *Container* and *Arm* in the example in Lst. 1. Preconditions are complemented by *hold-conditions* that need to hold during the entire execution and are checked at every tick. Finally, the post-conditions specify the expected effects of a skill if the execution succeeded. Therefore, they are checked after the execution of the nodes has succeeded, and the skill will fail if the post-conditions do not hold.

All types of conditions have one of four types: 1) verification of relations in the WM such as "Arm has a Gripper"; 2) check the existence of an element property, e.g. the gripper has an attribute that specifies the finger length; 3) compare the value of a property, for example having gripper finger longer than $0.1\,\mathrm{m}$; or finally 4) check for abstract relations in the ontology to verify that two elements are allowed to have a specified relation according to the ontology. In contrast to SkiROS1 [7], primitives can also have pre-, post- and hold-conditions, which increases robustness and allows to infer

Listing 1. The skill description of a pick skill. It defines the parameters, pre-, hold- and post-conditions and can be automatically used for task-level planning.

```
class Pick(SkillDescription):
  def createDescription(self):
    # ======= Parameters =========
    self.addParam("Container",Element("skiros:Location"),ParamTypes.Inferred)
    self.addParam("Gripper", Element("rparts:GripperEffector"), ParamTypes.Inferred)
    self.addParam("Object", Element("skiros:Product"), ParamTypes.Required)
    self.addParam("Arm", Element("rparts:ArmDevice"), ParamTypes.Required)
    # ======= PreConditions =========
    self.addPreCondition(self.getPropCond("EmptyHanded", "skiros:ContainerState", "Gripper", "=", "Empty", True))
    self.addPreCondition(self.getRelationCond("ObjectInContainer", "skiros:contain", "Container", "Object", True))
    # ======= HoldConditions =========
    self.addHoldCondition(self.getRelationCond("RobotAtLocation", "skiros:at", "Robot", "Container", True))
    # ======= PostConditions =========
    self.addPostCondition(self.getPropCond("EmptyHanded", "skiros:ContainerState", "Gripper", "=", "Empty", False))
    self.addPostCondition(self.getRelationCond("Holding", "skiros:contain", "Gripper", "Object", True))
```

Listing 2. The code skeleton for a primitive skill. The functions allow to define the initialization, execution and reaction to a preemption. In line 3 it is stated which skill description (such as the one in Lst. 1) the primitive implements.

```
1 class my_primitive(PrimitiveBase):
2   def createDescription(self):
3     self.setDescription(MyPrimitive(), self.__class__.__name__)
4   def onInit(self):
5     """ Called once when loading skills - it is not loaded on False """
6     return True
7   def onPreempt(self):
8     """ Called when skill is requested to stop. """
9     return self.success("Preempted")
10  def onStart(self):
11    """ Called just before 1st execute """
12    return True
13  def execute(self):
14    """ Main execution function. Returns: self.fail, self.step or self.success """
15    return self.success("Executed")
16  def onEnd(self):
17    """ Called just after last execute OR preemption """
18    return True
```

parameters.

Primitive Skills: Primitive skills, or short primitives, are the atomic actions in the SkiROS2 skill model. They typically implement behaviors that actively change the world, such as opening a gripper or moving a robot arm. Lst. 2 shows the primitive code skeleton that offers Python functions for the skill initialization, startup, execution, preemption and cleanup . Primitives always have a skill description such as the one in Lst. 1 that defines them on a semantic level and specifies the input and output parameters. When a skill is running, an `execute` function of the skill is called whenever the primitive is ticked as part of a BT and it must return one of the three signals *success*, *running* or *failure*.

Compound Skills: As an important extension to SkiROS1, this version formulates compound skills. They allow one to connect arbitrary amounts of other compound and primitive skills to define more complex behaviors. An

Listing 3. An example for a mockup pick skill that implements the example description in Lst 1. In line 7 the processor of the BT is set. In line 9 the parameter "Duration" of the skill "wait" is set to a concrete value and in line 10 parameter remappings are done.

```
1 class pick_fake(SkillBase):
2   def createDescription(self):
3     self.setDescription(Pick(), self.__class__.__name__)
4
5   def expand(self, skill):
6     """ In this function the BT is defined """
7     skill.setProcessor(SerialStar())
8     skill(
9       self.skill("Wait", "wait", specify={"Duration": 1.0}),
10      self.skill("WmMoveObject", "wm_move_object",
11        remap={"StartLocation": "Container", "TargetLocation": "Gripper"}),
12    )
```

example is given in Lst. 3. Compound skills are modeled with BTs and their processors. This enables developers to fully utilize the modularity and reactiveness of the BTs by reusing existing skills and to formulate reactions to changes in the world. SkiROS2 formulates the following processors:

- *Serial*: Sequential execution (logical AND)
- *SerialStar*: Sequential execution with memory
- *Selector*: Fallback (logical OR)
- *SelectorStar*: Fallback with memory
- *Parallel First Stop*: Process children in parallel until one succeeds or fails
- *Parallel First Fail*: Parallel run children until all succeed

Furthermore, there are decorators such as *NoSuccess* or *NoFail* that modify the return signals.

When including other skills in a compound skill, it is possible to explicitly set the parameters of the included skill if they are of any of the fundamental types. It is also possible to remap the parameters that are WM elements. For example, the "pick" skill in Lst. 1 has the parameter "Container". In the example in Lst. 3, the primitive to update the world model ("wm_move_object") expects an input parameter "StartLocation". It is possible to map the references for these parameters, so they can be used in both skills. This can also be used to resolve naming conflicts.

B. Skill Manager

Every robot has its own skill manager. The skill manager loads a set of specified skills from the skill libraries. It initializes them and complements the WM with semantic information such as the representation of the skill parameters and the skill conditions. After the initialization phase, the skill manager offers services to start, stop and debug skills, as well as monitor their execution. Since the skill manager is also executing the skills and accepts skill plans from the task manager, it is a core component of the platform. Whenever a skill or skill sequence is started, the skill manager creates a new task with a unique ID. Within a task, the parameters of the executed skills are shared on a blackboard. This allows to exchange information such as calculated poses or even camera images between the skills.

C. World Model

The world model server stores the ontologies and the instances. As such, it loads the relevant ontologies that specify the known concepts (schemas) at the start. Furthermore, it can load a specific scene that contains the instances, also called vocabulary of the concepts that are specified in the ontologies. A typical scene includes the semantic description of the specific robot system, objects in the environment or known locations. The scene is complemented by the skill manager with information about the available skills.

The information in the WM can be utilized for reasoning and for the parameterization of skills. It also exposes an API that is used inside of the skills to read from and write to the WM. Furthermore, custom reasoners can be plugged in. As an example for such reasoners, *skiros2_std_lib* implements a spatial reasoner that can transform coordinates and calculate Allen intervals algebra [29].

D. Task Manager

In Section III we outlined the importance of the vertical integration into systems that provide tasks such as MES. SkiROS2 addresses this by providing a sophisticated task planning integration that builds on the Planning Domain Definition Language (PDDL) [30]. It utilizes the *temporal fast downward planner*[1] [31] to guarantee the finding of an optimal skill sequence. In contrast to other solution, SkiROS2 has a fully automated generation of the problem and planning domain based on the current state of the WM. This includes the vocabulary of the scenes like the available robot as well as all the skills with their pre- and post-conditions. This generation drastically simplifies usability, as it is not necessary to maintain a separate planning domain.

The resulting plan can be sent to the skill manager and can become an executed task. It is then automatically converted into an executable eBT [24] and the skill manager will expand the branches of the eBT automatically.

E. ROS Integration and User Interface

SkiROS2 is well integrated into Robot Operating System (ROS) [27]. While SkiROS2 utilizes ROS as a middleware itself, it also features many convenient integrations that ease the use and implementation of skills. The WM and its elements are fully integrated with the ROS transformation system *tf*. As such, WM elements can be linked to frames that exist outside SkiROS2 and elements can be published as coordinate frames. On the skill level the *skiros2_std_lib* provides a skill primitive that can easily turn any ROS *action* into a skill.

The SkiROS2 graphical user interface (GUI) offers a drastically lowered entry hurdle for non-experts. Like many other ROS GUIs, it is written with ROS' *rqt* and can be used alongside them. In the different tabs, users get an overview of the loaded skills. Skill parameters can be changed and skills can be started and stopped. Additionally, the WM view in the GUI provides full access to the content of the WM.

[1] http://gki.informatik.uni-freiburg.de/tools/tfd/

The vocabulary including all the properties and relations can be inspected as well as fully modified. New relations and properties can be added through easy-to-use dialogues. An integration with the visualization tool *RViz* allows to modify the pose of a WM element.

V. Use-Cases

In this section, we discuss a selection of different use cases in which *SkiROS2* is used as a robot control system for partially-structured tasks.

A. Pick-place with a Mobile Robot

In the pick-and-place scenario, a mobile robot (e.g. *heron* in Fig. 1a) is tasked to place an item at a new location that can either be at the same or at another workstation. In this scenario it is required that the robot can drive between workstations, actuate the arm as well as the gripper, and use the camera to compute the exact pose of the object. The skills can switch the control mode of the arm between compliant control for contact-rich tasks and position control for planned trajectories. The used skills have full integration with planning and it is sufficient to specify a goal in PDDL such as `skiros:contain skiros:locationB skiros:objectA`. As described in Section IV-D, triggering the planning automatically generates the domain description and problem instance. A simplified example skill sequence is:

```
drive(workstationA)
pick(objectA)
drive(workstationB)
place(locationB)
```

Lst. 1 shows a simplified description of a *pick* skill. The specified preconditions can be used to infer parameters at planning time, but also at run-time. If this example skill is called manually with *SkiROS2* GUI, it is sufficient to specify the object to pick and the arm to use, and the rest of the parameters are automatically inferred using the WM.

B. Dual-arm Piston Insertion

The task in this use case is to perform a tight insertion of a piston into a real engine block. This insertion requires dual-arm manipulation as well as specialized tools since an assistive ring must be held for insertion. The skills used in this task heavily utilize the WM, but are at the same time able to learn parts of the procedure, such as lifting poses, from kinesthetic teaching. Additionally, the pose of objects, such as the engine block, can be estimated using the camera mounted on the robot arm [32]. The motion skills implement a combination of a motion generator with BTs [29] and have extensive checks on the robot state such as applied forces and torques. The extensive use of those conditions allows aborting the execution if they are exceeded.

While the skills for this use-case do not have the necessary pre- and post-conditions to use them for task planning, the concept of these conditions is used to regularly check if the system is in the desired state. Furthermore, these skills are written so that they are preemptable [33]. As an example for a preemption procedure we can consider a state in which an object is in the gripper. Aborting the current action and switching to a different task requires to place the object at an appropriate location first.

C. Reinforcement Learning of Industrial Robot Tasks

In a third use-case, *SkiROS2* is used to learn industrial tasks with reinforcement learning (RL) [11], [12], [34], [35], [13]. Such tasks include pushing an object on a table (shown in Fig. 1c) or learning a peg-insertion strategy. The skills used in these scenarios are written by domain experts and can be utilized by the task planner. However, there is no automated reasoning module for some of the skills that can be utilized to fully parameterize them for the task at hand. In this learning approach, the skill parameters in a skill sequence that cannot be parameterized are automatically identified. Then additional information about these learnable parameters, such as their type and upper and lower limits, is obtained from the WM and used to automatically create a learning scenario description. In the next step, the learning procedure starts either on a real robot system or in simulation, and the RL framework calculates the rewards depending on the performance. When leveraging on learning in simulation, up to thousands of executions can be run to identify a robust and well-performing set of parameters. Finally, the best configurations are presented to the operator and a final set can be selected for production [11], [12]. If the operator has an educated guess for good parameter values or experiences from similar tasks, this learning approach can also incorporate this information as priors to accelerate learning and increase safety [34]. With [13], an extension of [11], [12] is proposed to learn behaviors for a variety of task variations and allow zero-shot execution even for unseen task configurations.

VI. Conclusions

Modern autonomous robotic systems require solutions for the fundamental requirements: knowledge organization, control structuring, multi-robot orchestration and integration with external systems. We outlined the requirements and introduced the skill-based robot control platform SkiROS2 and its core components. Its unique combination of features makes it suitable for intelligent autonomous robot control. To show this, we outlined three example use cases that cover a wide variety of relevant challenges in robotics: Integration of diverse robotic systems, task-level planning and integration of vision and learning from users. As part of these SkiROS2 has also been shown to allow the combination of deductive methods such as reasoning with inductive methods such as learning to improve the performance of executions by interacting with the environment.

The *SkiROS2* platform is fully open source and comes with documentation, tutorials and examples. Currently it integrates with ROS 1, but a ROS 2 version and extensions such as *EzSkiROS* [36] are in development. The code is available at: `https://github.com/RVMI/skiros2`

Acknowledgement

The authors thank Bjarne Grossmann, David Wuthier, Faseeh Ahmad, Simon Kristoffersson Lind and Momina Rizwan for their contributions.

References

[1] R. Volpe, I. Nesnas, T. Estlin, D. Mutz, R. Petras, and H. Das, "The claraty architecture for robotic autonomy," in *2001 IEEE Aerospace Conference Proceedings (Cat. No. 01TH8542)*, vol. 1. IEEE, 2001, pp. 1–121.

[2] S. Bensalem, M. Gallien, F. Ingrand, I. Kahloul, and N. Thanh-Hung, "Designing autonomous robots," *IEEE Robotics & Automation Magazine*, vol. 16, no. 1, pp. 67–77, 2009.

[3] M. Stenmark and J. Malec, "Knowledge-Based Industrial Robotics," *Frontiers in Artificial Intelligence and Applications*, pp. 265–274, 2013.

[4] ——, "Knowledge-based instruction of manipulation tasks for industrial robotics," *Robotics and Computer-Integrated Manufacturing*, vol. 33, pp. 56–67, Jun. 2015.

[5] M. Tenorth and M. Beetz, "KNOWROB — knowledge processing for autonomous personal robots," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct. 2009, pp. 4261–4266.

[6] ——, "KnowRob: A knowledge processing infrastructure for cognition-enabled robots," *The International Journal of Robotics Research*, vol. 32, no. 5, pp. 566–590, Apr. 2013.

[7] F. Rovida, M. Crosby, D. Holz, A. S. Polydoros, B. Großmann, R. P. Petrick, and V. Krüger, "Skiros—a skill-based robot control platform on top of ros," *Robot Operating System (ROS) The Complete Reference (Volume 2)*, pp. 121–160, 2017.

[8] F. Rovida and V. Krüger, "Design and development of a software architecture for autonomous mobile manipulators in industrial environments," in *2015 IEEE International Conference on Industrial Technology (ICIT)*, 2015, pp. 3288–3295.

[9] D. Kortenkamp, R. Simmons, and D. Brugali, "Robotic systems architectures and programming," *Springer handbook of robotics*, pp. 283–306, 2016.

[10] D. Vernon, C. Von Hofsten, and L. Fadiga, *A roadmap for cognitive development in humanoid robots*. Springer Science & Business Media, 2011, vol. 11.

[11] M. Mayr, F. Ahmad, K. Chatzilygeroudis, L. Nardi, and V. Krueger, "Combining planning, reasoning and reinforcement learning to solve industrial robot tasks," *IROS 2022 Workshop on Workshop on Trends and Advances in Machine Learning and Automated Reasoning for Intelligent Robots and Systems*, 2022.

[12] ——, "Skill-based multi-objective reinforcement learning of industrial robot tasks with planning and knowledge integration," in *2022 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. IEEE, 2022, pp. 1995–2002.

[13] F. Ahmad, M. Mayr, and V. Krueger, "Learning to adapt the parameters of behavior trees and motion generators (btmgs) to task variations," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2023.

[14] A. Olivares-Alarcos, D. Beßler, A. Khamis, P. Goncalves, M. K. Habib, J. Bermejo-Alonso, M. Barreto, M. Diab, J. Rosell, J. Quintas *et al.*, "A review and comparison of ontology-based approaches to robot autonomy," *The Knowledge Engineering Review*, vol. 34, p. e29, 2019.

[15] C. Paxton, A. Hundt, F. Jonathan, K. Guerin, and G. D. Hager, "Costar: Instructing collaborative robots with behavior trees and vision," in *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017, pp. 564–571.

[16] J. O. Huckaby, "Knowledge transfer in robot manipulation tasks," Ph.D. dissertation, Georgia Institute of Technology, 2014.

[17] S. Balakirsky, Z. Kootbally, T. Kramer, A. Pietromartire, C. Schlenoff, and S. Gupta, "Knowledge driven robotics for kitting applications," *Robotics and Autonomous Systems*, vol. 61, no. 11, pp. 1205–1214, 2013.

[18] J. L. Wyatt, A. Aydemir, M. Brenner, M. Hanheide, N. Hawes, P. Jensfelt, M. Kristan, G.-J. M. Kruijff, P. Lison, A. Pronobis *et al.*, "Self-understanding and self-extension: A systems and representational approach," *IEEE Transactions on Autonomous Mental Development*, vol. 2, no. 4, pp. 282–303, 2010.

[19] C. Paxton, Y. Barnoy, K. Katyal, R. Arora, and G. D. Hager, "Visual robot task planning," in *2019 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 8832–8838.

[20] P. Schillinger, S. Kohlbrecher, and O. Von Stryk, "Human-robot collaborative high-level control with application to rescue robotics," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 2796–2802.

[21] D. Stampfer, A. Lotz, M. Lutz, and C. Schlegel, "The smartmdsd toolchain: An integrated mdsd workflow and integrated development environment (ide) for robotics software," *Journal of Software Engineering for Robotics (JOSER)*, vol. 7, no. 1, pp. 3–19, 2016.

[22] M. Iovino, E. Scukins, J. Styrud, P. Ögren, and C. Smith, "A survey of behavior trees in robotics and ai," *Robotics and Autonomous Systems*, vol. 154, p. 104096, 2022.

[23] M. Colledanchise and P. Ögren, *Behavior Trees in Robotics and AI: An Introduction*. Chapman & Hall/CRC Press, 2017.

[24] F. Rovida, B. Grossmann, and V. Krüger, "Extended behavior trees for quick definition of flexible robotic tasks," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 6793–6800.

[25] V. Krueger, A. Chazoule, M. Crosby, A. Lasnier, M. R. Pedersen, F. Rovida, L. Nalpantidis, R. Petrick, C. Toscano, and G. Veiga, "A vertical and cyber–physical integration of cognitive robots in manufacturing," *Proceedings of the IEEE*, vol. 104, no. 5, pp. 1114–1127, 2016.

[26] H. Bruyninckx, "Open robot control software: the orocos project," in *2001 IEEE International Conference on Robotics and Automation (ICRA)*, vol. 3. IEEE, 2001, pp. 2523–2528.

[27] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "Ros: an open-source robot operating system," in *2009 IEEE International Conference on Robotics and Automation (ICRA) Workshop on Open Source Robotics*, Kobe, Japan, May 2009.

[28] S. Bøgh, O. S. Nielsen, M. R. Pedersen, V. Krüger, and O. Madsen, "Does your robot have skills?" in *Proceedings of the 43rd international symposium on robotics*. VDE Verlag GMBH, 2012.

[29] F. Rovida, D. Wuthier, B. Grossmann, M. Fumagalli, and V. Krüger, "Motion generators combined with behavior trees: A novel approach to skill modelling," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 5964–5971.

[30] C. Aeronautiques, A. Howe, C. Knoblock, I. D. McDermott, A. Ram, M. Veloso, D. Weld, D. W. SRI, A. Barrett, D. Christianson *et al.*, "Pddl the planning domain definition language," *Technical Report, Tech. Rep.*, 1998.

[31] P. Eyerich, R. Mattmüller, and G. Röger, "Using the context-enhanced additive heuristic for temporal and numeric planning," in *Nineteenth International Conference on Automated Planning and Scheduling*, 2009.

[32] B. Grossmann, F. Rovida, and V. Kruger, "Continuous close-range 3d object pose estimation," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 2861–2867.

[33] D. Wuthier, F. Rovida, M. Fumagalli, and V. Krüger, "Productive multitasking for industrial robots," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 12 654–12 661.

[34] M. Mayr, C. Hvarfner, K. Chatzilygeroudis, L. Nardi, and V. Krueger, "Learning skill-based industrial robot tasks with user priors," in *2022 IEEE 18th International Conference on Automation Science and Engineering (CASE)*. IEEE, 2022, pp. 1485–1492.

[35] F. Ahmad, M. Mayr, E. A. Topp, J. Malec, and V. Krueger, "Generalizing behavior trees and motion-generator (btmg) policy representation for robotic tasks over scenario parameters," *2022 IJCAI Planning and Reinforcement Learning Workshop*, 2022.

[36] M. Rizwan, R. Caldas, C. Reichenbach, and M. Mayr, "Ezskiros: A case study on embedded robotics dsls to catch bugs early," in *2023 IEEE/ACM 5th International Workshop on Robotics Software Engineering (RoSE)*. IEEE, 2023, pp. 61–68.