The International Journal of Advanced Manufacturing Technology (2022) 120:6269–6285
https://doi.org/10.1007/s00170-022-09071-w

ORIGINAL ARTICLE

An extendable framework for intelligent and easily configurable skills-based industrial robot applications

Lisa Heuss[1] · Clemens Gonnermann[1] · Gunther Reinhart[1]

Received: 10 December 2021 / Accepted: 17 March 2022 / Published online: 7 April 2022

Abstract
Modern, flexible, and easy-to-use robotic technologies have the potential to support companies to increase their productivity within today's dynamic and volatile production. In this context, we introduce a skills-based software framework that makes it possible to configure the functional capabilities of industrial robots flexibly. In addition, we have structured the software framework into three consecutive expansion stages. In this way, it is possible to expand the robot's reasoning capabilities step by step so that the robot is enabled to be instructed at higher abstraction levels and to process increasingly complex tasks. The contribution of our work is the further development of previous approaches and ideas from the research field of skills-based industrial robotic frameworks by considering new and previously unaddressed design issues within the structure of our software framework. We demonstrate the application of the framework using the example of an industrial robot for assembling a diverse range of LEGO products. The example of use consists of three consecutive scenarios. To begin with, the robot assembles different predefined product variants. Subsequently, we extend the robot application in a step-by-step manner to allow the robot to execute more and more complex tasks until it can finally plan individual tasks autonomously. On the one side, our approach shows how to enable companies with little robotic experience to start developing robotic applications and thereby gain further expertise. On the other side, by using this approach the effort and time for developing industrial robot applications will be reduced in the long term.

Keywords Industrial robot · Skills · Framework · Task planning and control · Reconfiguration · Autonomy

1 Introduction

Today's industry faces the challenges of increasing product variety, cost pressure and market dynamics [1]. To remain competitive, the constant flexibilization of manufacturer's production processes is essential. In this context, modern robotic technologies are seen as a key enabler and will help companies to further increase their productivity [2]. However, considering the current state of the art, industrial robots are still lacking in the flexibility required to be quickly and easily reconfigured to process diverse tasks [3]. The development and programming of robotic applications require expert knowledge and are time-consuming [4].

Service-oriented solutions, which can be equated to the skills-based approaches addressed in this paper, promise to address these challenges [4]. The robot's capabilities are first encapsulated in so-called skills, which can then be applied in different situations by simple parameterization. Subsequently, individual skills can be combined in a modular fashion to process complex tasks.

Building on this idea, we derived our long-term vision for the development and operation of future robotic applications as follows [5–7]: Individual developers of robotic software deploy their solutions as robot skills via a central market place. In it, users can browse available skills. Different solutions can be combined in a modular way to individualize the robot's skills to fulfill the requirements of diverse applications. Then, suitable skills can be easily downloaded to the robot. During operation, the robot can be intuitively instructed to complete various tasks. If required, the robot plans its task autonomously and handles variations and uncertainties in the task itself and/or its surroundings. If the task range of the robot changes over time, the skills of the robot are adapted dynamically and on a situation-specific basis.

✉ Lisa Heuss
lisa.heuss@iwb.tum.de

1 Institute for Machine Tools and Industrial Management, Technical University of Munich, Boltzmannstraße 15, 85748 Garching near Munich, Germany

This paper focuses on the necessary software architecture of a robot within the outlined vision. We present the ***REpac framework***, a skills-based software framework for industrial robots that makes it possible to easily ***RE***configure the robot's functionalities and provides autonomous task ***p***lanning ***a***nd ***c***ontrol based on skills. We build on previous research and address the new combination of three previously unconsidered aspects in the design of our framework:

1. Within the REpac framework, the robot's skills should be freely configurable. Afterward, it should be possible that a user instructs the robot to process diverse tasks by composing its skills, or the robot even plans its task autonomously.
2. The REpac framework should provide modular usable components to equip the robot with different reasoning capabilities depending on the requirements of an application scenario. In this way, the robot can be instructed at different abstraction levels and process tasks of diverse complexity.
3. The REpac framework should be structured in successive expansion stages to support a user in learning how to build intelligent robotic applications. Thus, the user can start with a simple application in which the robot processes a predefined task and expands it step by step until the robot plans its tasks autonomously.

Aspects 1 and 2 intend to significantly reduce the effort and time required for developing and operating industrial robotic applications. Aspect 3 aims to support companies with little robotics expertise in acquiring the necessary competencies.

The remainder of the paper is as follows: Sect. 2 reviews related research. Sect. 3 presents the REpac framework. Sect. 4 describes an example of use for the REpac framework. Based on this, Sect. 5 discusses the results. Finally, Sect. 6 concludes the paper.

2 Related works

2.1 Robot operating system (ROS) and related tools

ROS is widely used in the robotics community. As a platform for the development of robotic applications, it provides a wide range of tools for common tasks (e.g. motion planning or visualization) and thus supports the exchange of knowledge about these topics [8]. To support the development and organization of robot behaviors, there are, for example, the ROS packages SMACH based on state machines [9] or py_trees_ros based on behavior trees [10] in the ROS ecosystem. In addition, the ROS package ROSPlan provides automated task planning [11]. However, these tools cover only a part of the functionalities required by a skills-based software framework for industrial robots as considered in this paper.

2.2 Skills-based industrial robotic frameworks

With reference to our vision for future skills-based robots, several conceptual aspects need to be taken into account when designing the framework. Firstly, we consider the definition and modeling of skills and tasks. In the state of the art for industrial skills-based systems, an overriding structuring into at least three abstraction levels can typically be identified [12–19, 21–25, 28]. Primitive skills at the lowest level are the actions directly performed by the robot system. Composite skills are reoccurring higher-level behaviors and are described by an organized collection of primitive or lower-level composite skills. In turn, tasks are specified by an organized sequence of skills. As [15] already noted, the exact distinction of abstraction levels is often difficult due to the underlying idea of flexible combining and composing of the functionalities offered by a system. In related research, several authors, for example [12–15], have already studied the definition and conceptual modeling of skills and/or the classification of production skills in more detail. We build on these aforementioned results.

Secondly, a well-defined system structure is needed to organize the implementation of skills and control them based on a given task. In [16], the authors present a skills execution management system based on state machines. New skills can be added in a modular manner. Through the integration of sensors, a robot perceives its environment and adapts the execution of its skills to the individual situation (e.g. detects variations in object's poses). The programming and control frameworks of [17] or [18] allow the easy use of developed programs on robots of different vendors and/or with diverse kinematics. The approach of [19] supports a seamless transition between the simulated and the physical robot system. Commercially, [20] already offers a programming framework that can easily be used with different robot systems and equipment. Common to all these approaches is the idea of building a system structure and/or implementing skills that are robust and thus reusable in diverse settings, resulting in a reduced development effort. This is in line with our vision stated in the introduction.

In our work, we progress from the idea that skills are distributed via a central market place. Users can browse available skills and download a set of suitable skills for their robot application. Following, in the third step we consider the selection, composition and configuration of suitable skills for an application. In [21], a model-based engineering tool to support users in this process was developed. The authors of [22] present a knowledge-based architecture to also share skills and additional knowledge between different systems and to provide various knowledge-based services to simplify the creation of robot applications. As can be seen from these examples, diverse approaches for sharing and choosing skills from a central market place already exist. Hence, we do not consider it any further. However, to meet the requirements of today's constantly changing and evolving production, we envisage the possibility of freely combining different skills in the robot architecture as a key requirement.

Lastly, we review existing approaches for instructing the robot in the task to be performed. Task-level programming represents the modeling of tasks based on the available skills of the robot and is also used by most of the works mentioned in the two previous paragraphs. For task-level programming, a wide range of concepts have been researched. A two-part procedure is promising as, for example, presented by [13, 23, 24] and described below based on these works. The task is first specified as a structured flow of skills, and then, the skills are parameterized. In the second step, the skills parameterization, the user is supported by intuitive interfaces or guidelines, or can teach the robot in an interactive way, for example, by demonstration or gestures. As programming based on this approach is done at a quite high abstraction level, it is especially suited to non-robotics experts. Nevertheless, each action to be performed by the robot as well as the detailed organization of all actions to complete a task needs to be exactly specified. This is still time-consuming for larger tasks. An alternative to task-level programming is automated task planning based on skills. This was, for example, explored in the research papers of [13, 25–28]. Here, the user only specifies the goal state to be achieved and does not need to specify all the actions to be performed. These are planned by the robot itself based on its available skills. For this, the robot maintains an internal model of its world containing its own state as well as that of its operating environment. In the following, we look at the above papers again on an individual basis with a focus on the robot system architecture. In [13] and [25], the underlying robot architecture is not described. [26, 27] consider mainly kitting applications, focus on the knowledge modeling and related technologies and see their robot architecture more as an example of how these approaches can be used. Within the skills-based control platform of [28], an internal world model for the robot is first specified and then based on that developers implement skills with which defined actions can be executed and thus the state of the robot or its environment changed. The well-defined interfaces of this architecture ease the software integration in a shared development team. However, we think that the binding of skills to a central world model reduces their reusability in a different context and thus restricts the possibility of freely combining skills from different sources. Therefore, for automated task planning, it can be summarized that it allows a further increase in the abstraction level of the task instruction. However, it also results in an increased effort to set up, for example, central knowledge management. To conclude, both presented approaches, task-level programming and automated task planning, simplify the task instruction of industrial robots, but also have their individual advantages and disadvantages. We think that for different applications one or the other approach is beneficial or even applicable, and thus, it is desirable to have the option of selecting the most suitable procedure as required. We want to address this aspect in more detail in the design of our framework.

2.3 Conclusions and need for research

From the literature review, we derive three main conclusions for our work:

C1 The effort required to develop various robot applications can be reduced considerably in the long term by implementing reusable skills and combining them flexibly in different robot applications.

C2 The level of abstraction for task instruction can be increased by enabling the robot to reason independently or even plan specific tasks. In addition, a robust skill design allows the robot to adapt the task execution based on the individual situation. In total, this way, the effort and required expertise for instructing and maintaining robotic applications during operation can be reduced.

C3 However, the comparison of existing works on task instruction also shows that, for example, the effort and expertise required to develop a robot application that is programmed at the task level is less than if the robot plans its tasks itself. Therefore, it is desirable to equip the robot with more or less reasoning capabilities depending on the application area of the robot and the resulting requirements.

Based on these considerations, we highlight the current challenges and related need for research that we address in this paper:

R1 To the best of our knowledge, the current skills-based development frameworks that support automated task planning [13, 25–28] do not explicitly consider the idea to deploy and share robot skills via a central market place and freely combine these on an application-specific basis. To fully exploit the potentials stated in C1 and C2, we see a need for research to investigate to what extent it is possible to select and flexibly combine diverse skills from a central market place in the robot architecture and use these also with automated task planning.

R2 Based on our considerations C1–C3, we think it is desirable to freely combine the skills available in a central market place in any skills-based robot application, equip the robot with reasoning capabilities as required, and in this way instruct it either using task-level programming or at a higher abstraction level that enables autonomous task planning. In the works of [12, 13, 25, 28], both methods for task instruction are utilized. However, [12, 13, 25] do not present their underlying robot system architectures. The approach of [28] does not foresee the use of the skills without the specified world model and thus has the restrictions mentioned above on the free composition of the robot's skills. Therefore, we see a need for research to design a suitable robot architecture for this purpose.

R3 Although intelligent systems offer high added value in operation, their development is still subject to a certain complexity (see C2 and C3). So, from our point of view, it is advisable to start with a simple robot application, for example, processing predefined tasks, and then enhance this subsequently with reasoning capabilities until the robot plans its tasks autonomously. In this way, the knowledge required for building intelligent robot applications can be acquired step by step. Thus far, this has not been considered in any skills-based robotic framework that we know and therefore should be addressed in future research.

3 REpac framework

This paper presents an extendable framework for autonomous task planning and control for easily reconfigurable skills-based industrial robots, the REpac framework. Based on the identified need for research, we follow three objectives within the design of the REpac framework:

1. *Reusability and flexible composition of skills:* The robot's functionalities should be flexibly configurable. Developers should design their software solutions as robust skills reusable in various situations on diverse robot systems and provide these via a central market place. Based on the available skills, the skills of a robot should be freely composable within the REpac framework as required for an application.
2. *Modular configuration of the robot's reasoning capabilities:* To allow users to adapt their robot to their needs, the REpac framework should further provide different components for task control, knowledge management, and task planning that can be integrated in a modular manner as required. In this way, it should be possible to equip the robot with various reasoning capabilities so that it can be instructed at different abstraction levels depending on the requirements of the individual application.
3. *Learning how to build intelligent robotic applications through step-by-step expansion:* Within the design of the REpac framework, the system architecture should be structured into three expansion stages. First, the robot should process different tasks specified by the user based on its available skills. Second, the robot architecture should be extendable by a shared knowledge management system for skills that can be used for information exchange or decision making during the execution of a task. Third, it should be possible to integrate automated task planning easily. Thus, the user can start with a simple skills-based robot application which she/he further develops. This step-by-step expansion will support the user in learning how to use the different components of the REpac framework and how to build intelligent robot applications.

The presented REpac framework builds on our previous research [5–7, 29]. Below, we define the modeling of tasks and skills. Based on this, we present the modules of the framework and their interaction with reference to the three mentioned expansion stages. Next, we firstly describe the structure of the skills and the underlying development procedure. Secondly, we present the possibilities of using the common knowledge management for skills and thirdly the integration steps for automated task planning.

3.1 Modeling of tasks and skills

In the presented REpac framework, the modeling of tasks and subordinate skills is derived from the state of the art, mainly [15] and [30]. We have also previously published our definition of skills in [6]. The modeling of tasks and skills is divided into four abstraction levels. In doing so, *tasks* for the robot are specified on one of two abstraction levels:

– A *mission* describes a goal state to be achieved by the robot.
– An *action plan* describes an organized sequence of actions to be executed by the robot's skills. If the task is specified as a mission, the plan contains the necessary actions to complete the mission.

Action plans are subsequently executed using the robot's skills. Skills are in turn hierarchically modeled on different levels of abstraction:

– *Composite skills* combine subordinate skills in a structured flow to complete complex behaviors.
– *Primitive skills* are the lowest level actions, which are performed directly by the robotic system.

3.2 Overview of the modules and expansion stages

The REpac framework provides functionalities for task planning and control of skills-based industrial robots and can be used for a wide range of applications. Figure 1 gives an overview of the framework and its modules. By selecting different skills from the skills database and combining them within the framework, the robot can be configured for a specific application. The skills and related data are marked in blue. Building on this, the REpac framework offers three expansion stages. The modules required for these expansion stages are colored in different gray levels. In the following subsections, the expansion stages are used as a basis for presenting the framework. The whole REpac framework is built based on ROS.

3.2.1 Expansion stage 1

The first expansion stage consists of a control platform for robot skills. Based on its available skills, the robot can complete various tasks. This first expansion stage more or less corresponds to the state of the framework as we presented it in [6].

The *skills* are the central building blocks for determining the functionalities offered by the robot. Thus, they define in which applications the robot can be used and which tasks it can perform. As it is expected that future robots will be used for a wide range of tasks at short notice, not all possible skills and related knowledge can be permanently stored on the robot. For this reason, robotic skills are deployed via a central skills database. From there, robot skills can be downloaded onto the robot on demand. Until now, this has been performed manually. In the future, however, it would also be conceivable for the robot itself to reload the necessary skills on a situation-specific basis. The REpac framework allows free combination of diverse skills as required by the current application.

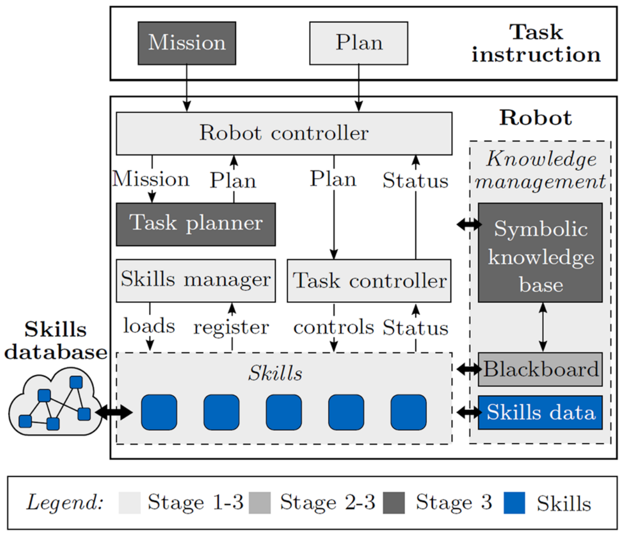


Fig. 1 Overview of the REpac framework

Upon start-up of the robot system, the *skills manager* is responsible for correctly loading all skills currently installed on the robot. After a successful start-up, all skills register themselves with the skills manager and in this way allow it to check whether all skills have been loaded correctly. Afterward, the robot is ready to receive tasks. Within the first expansion stage, *task instruction* is done at the level of an action plan based on the available skills. The *robot controller* accepts the tasks specified by the user and superordinately coordinates the processing of the tasks during operation. It forwards the plan to the task controller. Based on the defined action plan, the *task controller* controls the corresponding skills and supervises task execution using the skill's status messages. Furthermore, it provides status feedback to the robot controller.

3.2.2 Expansion stage 2

In the second expansion stage, the *skills blackboard* extends the REpac framework and offers a common memory and communication platform for the skills. The blackboard centrally stores information about the objects in the robot's environment. All skills can access and edit the data on the blackboard during their execution. In this way, an information exchange between the skills is realized. Furthermore, the data on the blackboard can be used for reasoning purposes. Now, that all information about the objects relevant for the robot is available centrally, and in a structured manner, skills can also be called directly with reference to specific objects relevant for their execution. The skill itself knows which further information about the objects it needs for its execution and queries these from the blackboard. This simplifies the task instruction. The blackboard can be saved and reloaded for later applications.

3.2.3 Expansion stage 3

In the third expansion stage, automated task planning is integrated into the REpac framework. In [7], we presented our first concept to integrate automated task planning into skills-based industrial robots and a draft for the robot architecture. In this paper, we describe the elaborated concept, its integration into the overall framework and the concrete implementation details.

This third expansion stage introduces two new modules, the *task planner* and the *symbolic knowledge base*. Using automated task planning, task instruction can be further simplified for the user. Tasks are then specified at the mission level by defining the goal state that is to be achieved by the robot. The task planner derives an action plan to complete the mission based on the actions the robot can perform based on its current skills set. In doing so, the task planner utilizes the symbolic knowledge base, which always represents the current state of the robot's world that is relevant for this purpose. It is updated by the robot's skills and is automatically synchronized with the skills blackboard. The exact distinction and connection between the skills blackboard and the symbolic knowledge base will be described later in the paper. Next, the robot controller forwards the action plan to the task controller to be executed by the skills.

3.3 Structure and development of skills

The idea behind our work is that developers can provide their individual robot skills independently of each other. Afterward, different skills can be combined in the REpac framework as demanded by the current application case. For the development of skills, this means firstly that they must be reusable in various tasks/situations and/or on different robot systems as previously demonstrated by [16–18]. Secondly, they must be designed as modular, independent and self-contained building blocks. Thirdly, they must provide a self-description about their functionality and characteristics similar to that requested by [15]. Finally, they must be externally controllable as well as monitorable and offer a standardized interface for this purpose [6]. These characteristics are essential for all skills. Thus, all skills are built based on a unified control structure and interface. Based on this, the individual functionality of a skill is implemented and completed by a description of its characteristics. We first describe the unified control structure of skills, and afterward, how to describe and implement skills based on this.

3.3.1 Unified control structure and interface

We presented the uniform control structure and interface to implement skills in our framework previously in [6, 29]. It is built based on an adapted version of the PackML interface state model as it is described in the PackML unit/machine implementation guide [31] that itself is based on the ANSI/ISA TR88.00.02-2015. Figure 2 illustrates the slightly more developed approach.

The control structure defines at a high level, and based on the adapted version of the PackML interface state model, the states a skill can adopt. The PackML interface state model distinguishes between waiting and acting states. This was taken over for the control structure of the skills. The execution of a skill happens in the acting states. During the later implementation of skills, a developer defines the routines that will be executed when the related state is entered. In doing so, it is defined what will happen when starting, executing, completing, etc., a skill. The waiting states function

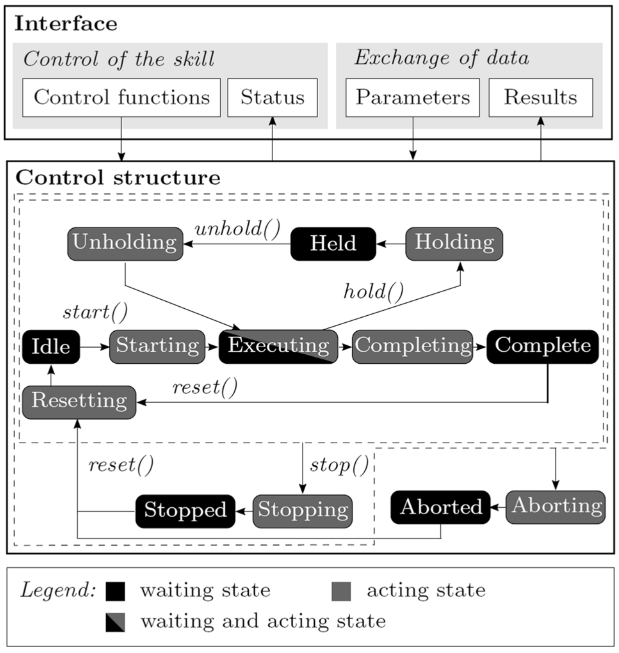


Fig. 2 Unified control structure and interface of skills

as stable intermediate states between the acting states. After the routine defined for an acting state has been processed, it transitions to the subsequent waiting state. Waiting states wait for a command in order to transition to their subsequent state. These commands are defined as the control functions of the skill to start, stop, un-/hold and reset it. Only the executing state is acting and waiting state at the same time. The stopping and aborting state can be reached from all states within the related dashed boxes.

The execution of a skill is externally controlled via the control interface based on the previously described control structure. The control functions are used for this purpose. The execution status of a skill corresponds to the state within the skill's control structure in which it is currently situated. In addition, a parameters list can be passed to the skill, when starting it. In turn, the skill returns its results after completion during resetting.

This unified control structure and interface is implemented by means of a skill meta-class. This skill meta-class forms the basis for all skills. Based on this, we introduce two distinct meta-classes for primitive and composite skills. These meta-classes inherit from the skill meta-class and extend it by further functionalities necessary for the development of concrete primitive and composite skills. In this way, developers are provided with a template for the development of skills that they can use for their custom skills. This is summarized in Fig. 3. For example, based on the primitive skill meta-class two skills for moving the robot to a pose or linear have been developed. These skills are further used to

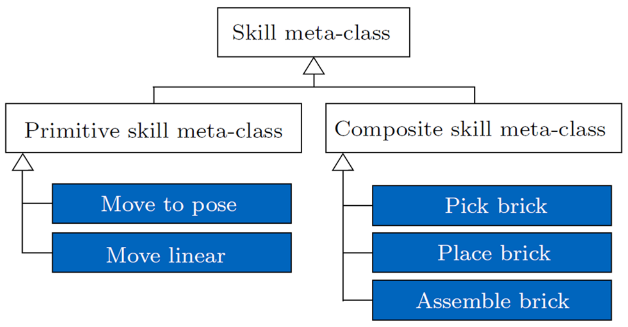


Fig. 3 Class structure of skills

Table 1 Exemplary description of the primitive skill **move to pose**. Individual parameters/results, such as the pose, are composed of several subordinate elements. This is simplified for better readability

Description of a primitive skill
Name
move to pose
General description
move the tool center point (TCP) of the robot to the given pose with the specified velocity
Skill parameters
pose, velocity
Skill results
-

develop a set of composite skills to pick, place and assemble LEGO bricks.

3.3.2 Description and implementation

The development of skills comprises two steps. First, the developer describes the functionalities and characteristics of the newly designed skill. These descriptions serve as information for other people who want to use the skill and are also required for the execution of it. Second, the developer must implement the procedures to be executed by the skill.

The state of the art presents different approaches for describing skills. In [15], skills are described from a process-oriented point of view as capabilities that are offered by a device or a combination of devices and are not bound to a specific hardware design or vendor. The authors of [13] describe skills as object-centered capabilities applied to physical entities that can be referred to. We think that it is best to combine these descriptive approaches depending on the selected abstraction level.

Primitive skills are the lower-level capabilities that are directly offered by the robot system. Thus, we describe these from a purely process-oriented point of view. As a basis for development, we use and extend the skills taxonomy of [15]. Within the description of a primitive skill, the developer specifies its name, provides a general description of the primitive skill and defines the parameters necessary for its execution as well as the results it returns after completion. The skill parameters and results are passed to or returned by the skill when calling it via the unified control interface presented in the previous section. By way of example, Table 1 shows the description of the primitive skill **move to pose**.

Based on the primitive skill's description, the developer implements its internal processes. For this, the developer has to specify what to do within the acting states of the skill's control structure. For this purpose, the skill meta-class provides a template with the related functions to be filled by the developer. Based on this and utilizing the previously defined parameters and results, the developer scripts the steps to be performed.

Composite skills are composed of subordinate skills and represent complex behaviors the robot uses to change the state either of itself or its environment. Thus, composite skills perform actions that are related to concrete objects in the world of the robot. In this context, it is easier to describe an action to be processed in relation to the objects involved. Thus, we describe composite skills from a process-oriented and object-centered point of view. In the first expansion stage of the framework, a purely process-oriented description of composite skills is still possible. If composite skills are to be used within the second and third expansion stage, an object-centered description of these skills is necessary. For the object-centered description of composite skills, it is possible to proceed as exemplified in Table 2 for the composite skill **pick brick**, which can be used by the robot to pick up a LEGO brick from its current location with the robot's gripper. Analogous to primitive skills, composite skills are identified by a name, provide a general description and have a set of parameters and results. In addition, the objects considered by the composite skill as well as parameters and results related to these objects are defined. When the skills blackboard is used in the second expansion stage, these object parameters/results are queried/updated by the skill from the skills blackboard. This means the skill parameters can be reduced to the objects involved and it is no longer necessary to pass all the underlying object parameters when calling the skill. As shown in the example of the skill **pick brick** in Table 2, this skill is simply parameterized with the brick that should be picked up with the robot's gripper at its current location. In this case the location represents an imaginary object. The object brick has assigned the parameters pose, which is composed subordinately of the elements describing the brick's position and orientation, and type. These object parameters are queried from the skills blackboard. If composite skills are to be used for automated task planning, their preconditions and effects need to be defined

Table 2 Exemplary description of the composite skill **pick brick**. Analogous to Table 1, the representation of the parameters/results has been partially simplified

Description of a composite skill		
Name		
pick brick		
General description		
pick the given brick from its location with the gripper		
Skill parameters		
brick, location, gripper		
Skill results		
-		
Objects		
brick	location	gripper
Object parameters		
pose, type	-	-
Object results		
pose	-	-
Preconditions		
brick-at-location	-	gripper-free
Effects		
brick-in-gripper	-	not(gripper-free)
not(brick-at-location)		

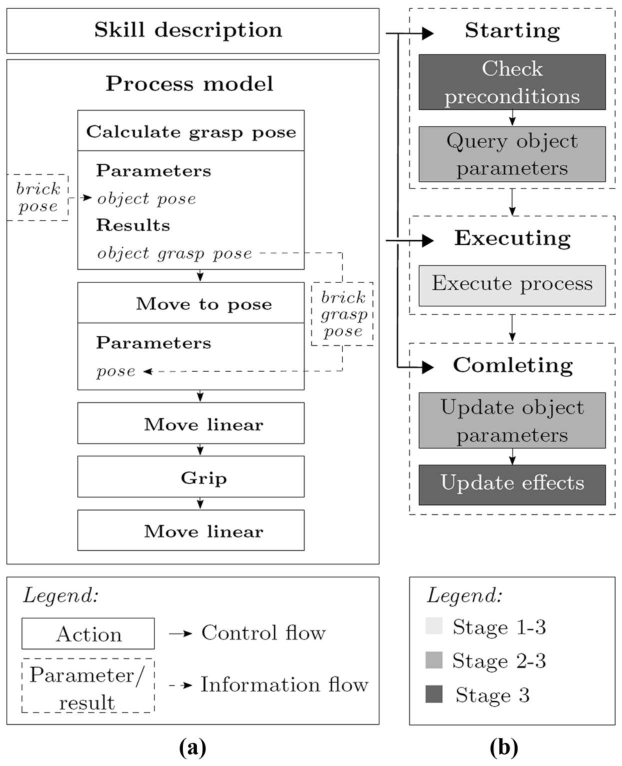


Fig. 4 (**a**) Components of a composite skill and modeling of the skill **pick brick** as a state machine. The presentation is simplified for clearer visualization and easier understanding. (**b**) Embedding of a composite skill into the unified control structure of skills

[13]. Thus, to enable the use of a composite skill in combination with task planning in the third expansion stage, we additionally describe its preconditions and effects in the skill's description. This will be explained in more detail in Sect. 3.5.

To model the process to be executed within a composite skill, a choice can be made between two approaches: finite state machines or scripting. Using the first approach, the process sequence of a composite skill is modeled by a finite state machine as shown for the skill **pick brick** on the left side in Fig. 4 in a typical and slightly simplified manner. The states represent the actions executed by subordinate skills and the transitions define the control flow of the actions to be proceeded. The previously defined skill and object parameters are used to parameterize the subordinate skills to execute the defined actions. Furthermore, results of the execution of a subordinate skill can be forwarded as input parameters to a succeeding skill. As primitive and composite skills are on different abstraction levels, parameters may have different names and meaning within their namespaces. For example, within the namespace of the primitive skill **move to pose** the pose describes the pose to which the TCP of the robot should move. Within the namespace of the composite skill **pick brick**, different poses for the brick itself and the grasp pose of the brick are defined. As a primitive skill may be used several times to execute different actions within a superordinate composite skill, a remapping of the parameters between their namespaces is necessary. We specify this on the level of the superordinate skill. Thus, subordinate skills can be easily and independently integrated into the procedure of a superordinate skill without any modification. For the underlying implementation of the state machine approach, we use SMACH [9]. As an alternative approach for programming composite skills, scripting based on python can also be used. For this purpose, a simple to use interface to call subordinate skills is provided. Both of the mentioned approaches and related functionalities for programming composite skills are offered by the composite skill meta-class. A developer can either select one of the two approaches or combine them. Furthermore, the meta class can be extended by further programming approaches (e.g. behavior trees as used by [32, 33]) in the future. The ability to use and combine different approaches for implementing skills provides more design freedom for the developers of these. This allows them to better choose a design approach based on the requirements of the process to be implemented or their own technical skills. Most existing works are limited to one approach.

Finally, Fig. 4b shows how composite skills are embedded into the unified control structure. The execution of a composite skill takes place in its executing state based on the defined process model. In the first expansion stage, all required skill and object parameters must be passed to the skill, when starting it. For the second and third expansion stage, the composite skill meta-class offers functions to interact with the knowledge management during the starting and completing routine. These will be introduced in the following two subsections.

3.4 Usage of the skills blackboard

In the second expansion stage, the skills blackboard is integrated into the robot architecture. The skills blackboard functions as common memory and platform for information exchange for skills. As composite skills are built in an object-centered manner, the skills blackboard also applies an object-centered structure to its data. It stores all objects of interest for the skills and their parameters. All skills can access the blackboard and use it as a platform for information exchange or reasoning purposes. When utilizing the blackboard, the skills need to be expanded by the interaction with the blackboard. Composite skills can then be used in an object-centered manner. Thus, it is no longer necessary to pass all single parameters to them, but they can be parameterized directly with the objects for which they should perform the action and the skill itself queries further necessary parameters of these objects from the skills blackboard. Figure 5 illustrates a simplified example of the use of the skills blackboard. In the example, the robot builds a given LEGO product. The related task is passed to and controlled by the task controller using the subordinate skills. The supplied bricks and related data are stored on the skills blackboard. The task controller calls the skills accordingly to the action plan. The two skills **pick brick** and **assemble brick** are called to pick up brick “brick1” from location “loc2” and assemble it at location “loc4”. The skills are easily called for the related objects and query the parameters required for

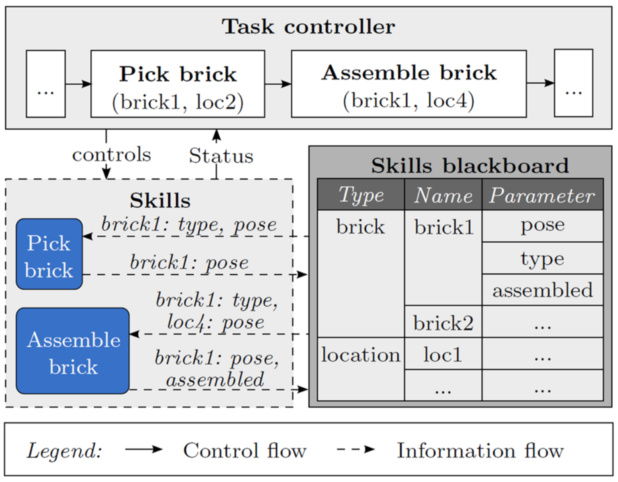


Fig. 5 Exemplary use of the skills blackboard in the second expansion stage

execution (e.g. concrete poses) from the blackboard. After completion, the skills update the data on the blackboard.

Below, we describe how the skills blackboard can be used. Use of the skills blackboard is mainly specified within the development of the skills. It is not mandatory to specify the blackboard content in advance. For use of the skills blackboard by a composite skill, the related meta-class provides the functionalities to easily integrate the usage of the skills blackboard into a skill. For this, the developer first needs to specify the data to be exchanged with the blackboard. These are the objects and their parameters/results that are necessary for the skill’s execution, as already shown in Table 1 for the skill’s description. As shown in Fig. 4b, based on these specifications the skills query the data about the objects necessary for their execution from the blackboard during their starting routine. Subsequently, they perform the requested action within their executing routine based on the specified process model. At this point, the blackboard data may be used for the parameterization of the process, reasoning or also for other purposes that a developer may think of. During their completing routine, the skills update all data on the blackboard, which has been changed during their execution. In conclusion, the data exchange with the skills blackboard is integrated in the starting and completing state of the skill using the functions of its meta-class and based on the skill’s description. Then, the skill can use these data for its process execution.

The meta-class for primitive skills does not offer special functions to interact with the skills blackboard. Nevertheless, it is possible to use the blackboard within a primitive skill. This can be defined within its implementation by directly using the services offered by the blackboard. But, we recommend to primarily interact with the skills blackboard via the composite skills.

When using the skills blackboard, it is important to use a uniform naming convention for objects and their parameters considered on the blackboard so that they are named consistently between different skills and on the blackboard. For future work, it might be interesting to integrate a translation mechanism into the blackboard to enable the usage of different naming conventions between different skills. Furthermore, the REpac framework allows storing a blackboard when shutting down the robot, as well as loading of an existing one when starting the robot system.

3.5 Integration of automated task planning

In the third expansion stage, automated task planning can be integrated into the robot architecture. Thus, a task can be specified as a mission by a goal state to be reached and the robot autonomously plans the action plan to complete it.

Before we describe the integration of automated planning into the REpac framework, we first give a short introduction to it based on [34]. A planning domain models the relevant object types, their properties and the possible executable actions. Based on this, a planning problem consists of a description of the actually existing objects, their initial state and the goal state to be achieved. Planning domain and problem can be specified by means of the planning domain definition language (PDDL). A planning algorithm takes the planning domain and problem as input and solves the planning problem by finding an action plan to proceed from the initial state to the goal state.

To illustrate this explanation, we again use the already considered example of a robot picking and placing LEGO bricks. For this example, the planning domain contains as object types the gripper of the robot, available bricks and their possible locations. In the planning domain, these objects can have the following properties: gripper-free, brick-at-location, brick-in-gripper. These properties can either be true or false. Based on these specifications, the executable actions are modeled in the planning domain. These correlate to the skills of the robot. The actions are parameterized by a set of objects and have a set of preconditions, which must be fulfilled to execute the action, as well as a set of effects that are caused by the execution of the action [34]. Table 2 illustrates the preconditions and effects of the skill **pick brick**. Based on this description of a planning domain, we specify a planning problem. We assume that there are three objects: the robot's gripper, one brick "brick2" and one location "loc3". In the initial state, the gripper of the robot is free and the brick "brick2" is at location "loc3". In the goal state, the brick "brick2" should be in the robot's gripper. The solution of this planning problem is an action plan containing of one action to pick up brick "brick2" from location "loc3".

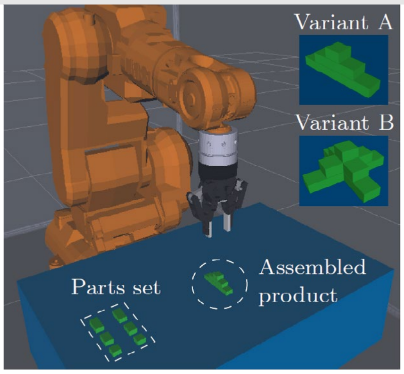
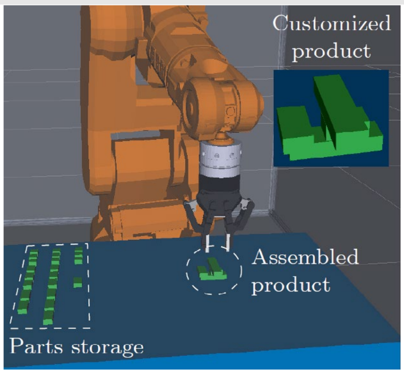
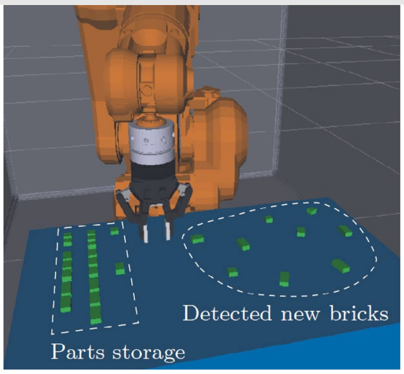
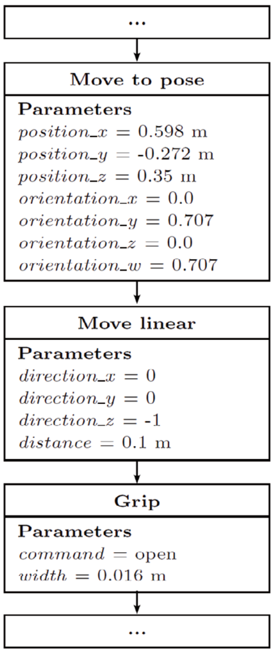
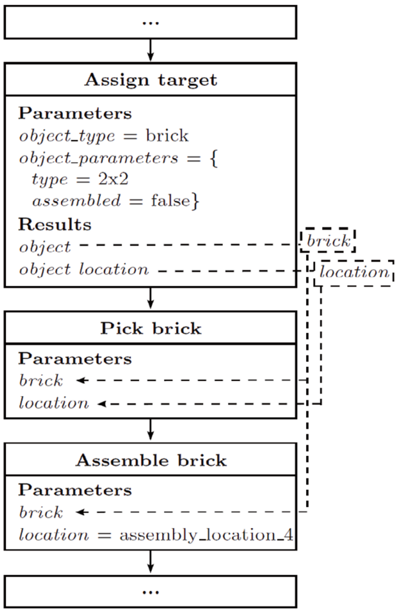
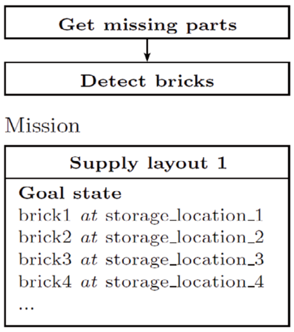
The REpac framework is extended by the task planner and the symbolic knowledge base for integration of automated task planning within the third expansion stage. The current planning domain is described by means of PDDL. Using the specifications from the planning domain, the symbolic knowledge base stores the current state of the robot. When a new mission is handed over by the user to the robot, the task planner generates a planning problem by using the information of the symbolic knowledge base for the initial state and the task specification within the mission for the goal state. In the next step, the task planner plans a suitable action plan based on the planning domain and the created planning problem. For the implementation of the task planner and the symbolic knowledge base, we used ROSPlan [11] and integrated it into the framework. After completion, the task planner returns the found action plan to the robot controller. The robot controller hands the generated action plan over to the task controller to execute it using the robot's skills. During execution, in addition to the blackboard, the skills also update the symbolic knowledge base. The skills have to be expanded by this interaction when they are used in combination with task planning. The composite skills meta-class provides an interface for this purpose. The blackboard and the symbolic knowledge base are interconnected in the framework. Both represent the objects known by and relevant for the robot. The symbolic knowledge base represents the objects at a higher abstraction level and describes their interconnections in a symbolic way (e.g. brick "brick2" at location "loc3"). Subordinate to this, the blackboard stores concrete data for the individual objects, which is necessary for the skill's execution (e.g. the concrete pose of brick "brick2"). Objects added to or deleted from the blackboard are also automatically updated in the symbolic knowledge base.

Below, we describe the necessary steps for extending an application by automated task planning: First, the user has to decide which skills should be used for task planning. Second, the PDDL planning domain needs to be described. The selected skills represent the actions that should be usable in the planning domain and determine the relevant object types and their properties. Third, the preconditions and effects assigned to the actions in the planning domain also have to be added to the description of the related skills as shown in Table 2. As illustrated in Fig. 4b, the skills interact with the symbolic knowledge base to check their preconditions during their starting routine and to update their effects upon completion. We are currently working on simplifying the creation of the planning domain by automatically generating the planning domain for specific application fields based on the selected robot skills.

4 Example of use

In this section, we present an example of use for the REpac framework. We use the simulation of a small robot cell that contains an ABB IRB140 robot and a ABB IRC5 compact control unit. The robot is equipped with a 2F-85 two finger gripper and a FT 300 force torque sensor from Robotiq. In front of the robot is a table which is its working area. A Microsoft Kinect camera is mounted above the table. As an example product we use LEGO. In the following, we present three application scenarios that build on each other. All scenarios are illustrated in Table 3. The application scenario is always introduced first. Based on it, we derive the requirements for the robot architecture. Then, we describe the implementation of the scenario using the REpac framework. Finally, we demonstrate how to instruct a task to the robot to use the application in operation.

Table 3 Summary of the scenarios of the example of use

Assembly of predefined products	Assembly of customized products	Autonomous refill of parts storage
Illustration		
		
Implementation		
Expansion stage 1	Expansion stage 2	Expansion stage 3
Task instruction		
Action plan 	Action plan 	Action plan 

Get missing parts
Detect bricks
Mission
Supply layout 1
Goal state
brick1 at storage_location_1
brick2 at storage_location_2
brick3 at storage_location_3
brick4 at storage_location_4
...

4.1 Assembly of predefined products

Scenario First, the robot is tasked with assembling different predefined product variants. The parts required for assembly are provided in defined sets.

Requirements For this application the robot needs capabilities for moving and gripping that are encapsulated into modular function blocks. It further needs to be able to combine these capabilities in different variations to complete the assembly of the different product variants.

Implementation We realized this scenario using the first expansion stage of the REpac framework. We implemented the required capabilities as robot skills. Thus, we developed the primitive skills **move to pose**, **move linear** and **grip** for moving the robot to defined poses or in a linear direction and opening/closing its gripper. As we use MoveIt [35] for motion planning, we additionally added some skills to interact with MoveIt and the digital model of the robot and its environment for this purpose. All implemented skills are stored in a designated folder. In this way, the implementation is completed. When starting the robot system, the skills manager automatically loads all stored skills. Based on the available skills, it is possible to specify different tasks as action plans and send them to the task controller for execution.

Task instruction We specify one action plan for each product variant. An extract of one action plan is shown in Table 3. The task is modeled using the previously implemented primitive skills which are parameterized to complete the current assembly (e.g. by the characteristic poses). Depending on the product variant to be assembled, the related action plan is passed to the robot.

4.2 Assembly of customized products

Scenario In the next scenario, we also want to use the robot for the assembly of customized products. At this point, it is no longer possible to provide the required parts as prepared sets. Thus, we supply the parts in a buffer storage to be assembled as required. The assembly should be processed based on an assembly plan defining the sequence and assembly locations of the related parts. For example, this information can be automatically derived from the CAD model of a product as shown by [36].

Requirements For this application, the robot now needs to internally store information about the currently available parts in its buffer storage as well as monitor which parts have been assembled. Based on this knowledge, it needs to select the parts to be assembled in the next step and parameterize the execution of the individual actions to complete the overall task.

Implementation For this purpose, we have extended the robot architecture by the skills blackboard and use the second expansion stage of the REpac framework. The robot stores all information about the relevant parts and their parameters as well as related storage and assembly locations on the skills blackboard. The management of the bricks supply is handled by a set of skills. For example, the skill **assign target** decides about the part to be assembled in the next step. Using the information on the blackboard, it selects an appropriate part based on its type and characterizing parameters (e.g. a brick of type 2x2 that has not been assembled). For the assembly of the requested products, we implemented two new composite skills **pick brick** and **assemble brick** for picking and assembling LEGO bricks. As can be seen in Table 2, the picking skill is parametrized by the brick to be picked up and its current location. The assembly skill is parametrized by the brick and its dedicated assembly location. Both skills are defined as finite state machines using the subordinate primitive skills. Within its execution, these composite skills request necessary information (e.g. poses) from the skills blackboard. The new skills are also added to the designated folder. Thus, they can be found and started by the skills manager at start-up.

Task instruction As in the previous scenario, we specify the task as an action plan. However, now we do not need to explicitly specify all poses within the task description. Instead, we can specify the task with reference to specific objects. This is exemplified in Table 3 by the excerpt of a possible action plan for this scenario. The skill **assign target** determines a suitable brick and its current location for the next assembly step and passes this information as input parameters to the skills **pick brick** and **assemble brick**. Assembly locations (e.g. assemble_location_4) are predefined during the creation of the action plan and stored with all their parameters on the skills blackboard. These two skills will pick up the designated brick from the storage and assemble it. This simplification within the task instruction makes it affordable to also use the robot system for product variants with smaller unit numbers or even for customized products.

4.3 Autonomous refill of parts storage

Scenario Lastly, the second scenario is expanded. After assembling some products, the robot is to refill its storage autonomously. For this purpose, the parts are randomly placed in its working area. Using the camera, the robot is to detect the provided parts and sort them into the predefined layout of its storage.

Requirements In this scenario it is not possible to specify the task to refill the parts storage as an action plan in advance. Depending on the previously assembled products, the number and type of parts to be refilled differs and, in addition, the parts are randomly placed. Thus, the robot should autonomously plan the necessary actions to sort all missing parts into its buffer store.

Implementation The third expansion stage of the REpac framework offers automated task planning and is used for this scenario. At a functional level, we need additional skills for determining the missing parts in the storage, detecting the bricks offered to the robot by the camera and placing these into the storage. To enable the robot to autonomously plan its tasks, we created a PDDL planning domain file and used this to configure the task planner of the robot. Furthermore, we expanded the robot's skills with preconditions and effects.

Task instruction During operation, the tasks to assemble different products are specified as in the previous application scenario. The task instruction to refill the parts storage comprises two parts as shown in Table 3. First, we call the action plan to determine the missing parts in the storage and to then detect these missing bricks among the randomly provided new bricks in the robot's working area. Second, we specify the desired layout of the parts in the storage as the goal state in the mission and send it to the robot. By means of automated planning, the robot plans the pick and place actions to complete the mission and executes them using its skills.

5 Discussion

Based on the experience from the example of use in the previous section and our resulting assessments, we first discuss in this section how our work serve the need for research that we identified in Sect. 2 and state the resulting advantages. Afterward, we identify current limitations and challenges that arise and provide recommendations for future research.

5.1 Reusability and flexible composition of skills

Utilizing the benefits of the skills-based approach, we could flexibly configure and extend the functionalities of our robot system. This is shown in Fig. 6. Whereas in the first scenario, we had to develop all skills, we continued to use an increasingly large part of the skills in the following scenarios. By reusing previously implemented skills within new application scenarios the development effort for these applications can be reduced. This confirms our observations from the review of the related works. Based on that and to the best of our knowledge, the REpac framework is the first skills-based industrial robotic framework that follows the idea to share robot skills via a central market place, freely combine these on an application-specific basis, and also use these with automated task planning. For the first and second application scenarios, we instructed the tasks at different abstraction levels using task-level programming. In contrast, the robot partly plans its task autonomously in the third scenario. We could reuse a large proportion of the skills in all three scenarios. Some of these skills had to be expanded to interact with the knowledge management or be usable with automated task planning. However, the effort for this is relatively small compared to the initial implementation of the skill's functionality. This demonstrates that the REpac framework provides the prerequisites to fulfill the stated idea. However, further research should be done to realize more diverse application scenarios within the REpac framework and investigate the potentials and limitations of this approach in more detail. To conclude, in the long term, we expect that the reuse of robots' skills and their composition in various application scenarios will lead to the great advantage of reducing the working effort for new applications significantly. As the number of implemented scenarios increases, so will the number of available skills. Thus, when developing new application scenarios, more and more skills can be reused, and fewer skills need to be newly created. This advantage is valid regardless of how the robot is instructed for different tasks and strengthened by reusing the same skills in diverse system settings.

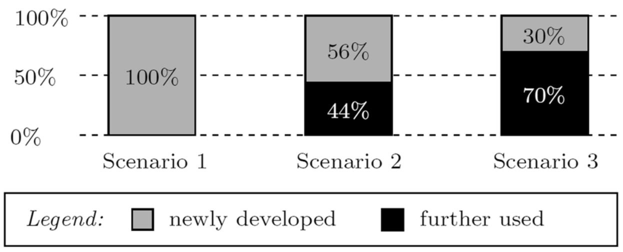


Fig. 6 Extension of robot functionalities across application scenarios. For each scenario, the diagram shows the percentage of skills that were newly developed for that scenario or continued to be used from the previous scenario

5.2 Modular configuration of the robot's reasoning capabilities

The REpac framework allows on the one side to configure the functional capabilities of a robot by selecting suitable skills and combining these within the robot architecture based on the requirements of the current application. On the other side, a user can further equip the robot with different reasoning capabilities offered through the REpac framework. Thus, the robot can be instructed at different abstraction levels using task-level programming or automated task planning. Looking back at the discussion of previous works in Sect. 2, the REpac framework is the first skills-based industrial robotic framework that considers combining these properties in one framework as far as we know. In this context, our contribution is the combination of existing approaches from this subject area in a new way. Following, we discuss the possibilities arising from the REpac framework structure in a qualitative way using Table 4. First, we look at the possibilities for the application characteristics. The successive expansion stages allow to start with task control based on skills and further integrate components for knowledge management and task planning

Table 4 Qualitative comparison of the expansion stages of the REpac framework

Expansion stage	1	2	3
Characteristics of the application			
Reasoning capabilities	◢		
Abstraction level for task instruction	◢		
Working effort and required expert knowledge			
Application development	◢		
Task instruction during operation	◣		
Recommended application field			
Product units	◣		
Variations in task	◢		

in a modular manner. In this way, the robot can be equipped with increasing reasoning capabilities across the expansion stages. Our three application scenarios also reflect this, using the three different expansion stages. When the reasoning capabilities of the robot increase, we can instruct tasks for the robot at higher abstraction levels during operation. Whereas we predefined all movement sequences in the first scenario, we only needed to specify the target poses for the LEGO bricks in the parts storage in the third scenario. Second, based on these considerations, we discuss the working effort and required expertise to implement robotic applications based on the three expansion stages. The REpac framework must be initially set up for the considered robot application; the necessary skills must be implemented or downloaded from a central repository and composed within the REpac framework for all three expansion stages. Within the second expansion stage, the skills need to be additionally extended to interact with the knowledge management in the form of the skills blackboard, and the skills blackboard must be initially set up if required. Furthermore, within the third expansion stage, the skills need to be expanded for use within task planning, and the PDDL planning domain must be described. If necessary, knowledge management must be initialized again at the beginning. These additional working steps allow a user to equip a robot with increasing reasoning capabilities. However, compared to the first expansion stage, these additional working steps also cause a higher working effort and require further expertise in knowledge management and automated task planning. In contrast, the effort and necessary expertise for task instruction decrease across the expansion stages within the operation of a robot application due to the possibility to specify tasks at higher abstraction levels. The advantage of these mentioned development options offered by the REpac framework is the possibility to better design a robotic application tailored to its individual requirements (see also Table 4). A user can compare additional effort and resulting benefits to decide what reasoning capabilities its robotic application requires and select the appropriate expansion stage. For example, the first expansion stage may be sufficient for applications with high product units and static environmental conditions. If product units decrease or variations in tasks increase, we recommend using the second or third expansion stage. Variations in tasks can result from diverse sources. For example, in the second scenario, the parts to be assembled are selected based on the current state of the parts storage. Whereas in the third scenario, parts are randomly placed, and the number of parts to be manipulated as well as the part's types differ within each task. Partly unknown and dynamic environmental conditions are further examples that result in a high task variety for the robot. We expect that the possibility to configure the robot architecture depending on the requirements of an application and thus only use the components required will reduce the development effort for each application. Furthermore, we assume to strengthen this advantage, as the resulting manifold possibilities of using the framework will increase its applicability for diverse application scenarios. In addition, the possibilities to reuse once developed skills will increase, which in turn has a positive effect on the development effort for new applications.

5.3 Learning how to build intelligent robotic applications through step-by-step expansion

The example of use demonstrates that the REpac framework allows a step-by-step realization of more advanced scenarios. By increasing the reasoning capabilities of the robot across the expansion stages, the robot can process more complex tasks while simultaneously task instruction during operation is simplified. In the first scenario, we developed a set of primitive skills and combined them in a modular fashion in different action plans to process the assembly of various predefined product variants. In the second scenario, we used the skills blackboard for joint knowledge management. In doing so, we could specify the action plans at a higher abstraction level with less parameterization effort. In the third scenario, we enabled the robot to plan its tasks autonomously partially. Thus, we could specify a task as a mission that describes the goal state to be achieved. The task control, knowledge management, and task planning modules are integrated independently across the expansion stages. Thus, a user can learn the required expert knowledge for using these modules step-by-step. Summing up, the design of the REpac framework supports a user in learning how to build intelligent robotic applications that is a novelty compared to the related works. This results in the advantage that companies with less expert knowledge can start building robotic applications and expand their expertise as required to realize increasingly diverse and complex application scenarios.

5.4 Limitations, challenges and recommendations for future research

Finally, we discuss our work's current limitations and challenges and derive recommendations for future research. Firstly, it is essential to consider that we are not aiming to develop fully autonomous robots. Based on our review of the related works in Sect. 2, we would recommend an approach like the one of [28] for this purpose. Considering the evolving industrial requirements, the REpac framework addresses an easy configuration of robotic applications based on skills combined with the possibility to equip the robot with intelligent capabilities. The estimation of the necessary development effort compared to the later benefit in operation is a challenge here (see also Sect. 5.2), and future research should consider this aspect in more detail. Secondly, information exchange between skills, such as via the blackboard, requires a common understanding of the related data. If skills are developed independently of each other and afterward combined to complete a specific task, this does not exist. For example, skill development sometimes required an iterative approach across application scenarios. Therefore, suitable solution approaches should be investigated, and initiatives for standardization should be strengthened in the future. As considered previously, integrating a system to translate different naming conventions might be a solution approach. Thirdly, manual creation of the PDDL planning domain is time-consuming and requires expert knowledge when using automated task planning. [26] or [28] present approaches to automating the generation of these knowledge models. Simplifying the usage of automated task planning techniques is essential to promote their use in production and thus should further be explored in future research. Lastly, we used LEGO as an example product in the example of use. We assume that the presented results can be transferred to more complex industrial scenarios in the real world. The framework itself and the implementation guidelines for skills can be used in the same way. Nevertheless, and as already stated in Sect. 5.1, we recommend realizing more diverse industrial application scenarios to investigate the benefits and limitations of our approach in more depth.

6 Conclusion and outlook

This paper presents the REpac framework, an extendable skills-based software framework for industrial robots. By designing various robot skills and combining them in the robot architecture, the robot's capabilities can be easily and flexibly configured as demanded by an individual application. Furthermore, the REpac framework offers modules for task control, knowledge management, and task planning, which can also be deployed in a modular manner as necessary. These modules are structured into three expansion stages to increase the robot's reasoning capabilities in a step-by-step manner. Thus, it is possible to start with a simple application such as assembling predefined product variants and extend this until the robot plans its tasks autonomously. We presented an example of use for the REpac framework that allowed us to demonstrate the following benefits: The reuse of robot skills in diverse application scenarios combined with the possibility to equip the robot with reasoning capabilities as required by the individual application will decrease the working effort for developing new applications in the long term. In addition, the structuring of the REpac framework in consecutive expansion stages allows companies with less expertise to start developing robotic applications and gain further expert knowledge during the realization of increasingly complex application scenarios. In total, we expect that these benefits will simplify the development and operation of industrial robotic applications, make their usage options more flexible and thus allow using industrial robots for new application areas.

Based on the identified recommendations for future research in the previous chapter, we plan to work on the following topics in the future: First, we want to explore concepts for the automated generation of the related PDDL planning models in the context of the REpac framework to simplify the use of automated task planning techniques in production. Second, we intend to evaluate the REpac framework in real-world applications to transfer our research approaches into industrial practice. In this context, the concrete design of the skills will play a crucial role. Finally, we aim to investigate the integration of the REpac framework into the higher-level production planning layer. For this purpose, we intend to use the REpac framework as an execution layer in a planning system for the automated setup of process monitoring in reconfigurable assembly systems [37].

Funding Open Access funding enabled and organized by Projekt DEAL. The research was developed with the support of the Bavarian Research Foundation in the research network FORobotics (AZ-1225-16) and of the Franco-German Alliance for Factory Supported by the Federal Ministry of Education and Research (BMBF) and the Free State of Bavaria under the Excellence Strategy of the Federal Government and the Laender, and the French alliance for the industry of the future, in the context of the German-French Academy for the Industry of the Future of Institut Mines-Telecom (IMT) and Technical University of Munich (TUM) in the research project RDS2-Production. We thank them for their funding and thank all participating project partners for their support in these projects.

Code availability The code that supports the findings of this article is available from the corresponding author upon reasonable request.

Declarations

Ethics approval Not applicable.

Consent to participate Not applicable

Consent for publication Not applicable

Competing interests The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Koren Y, Gu X, Guo W (2018) Reconfigurable manufacturing systems: Principles, design, and future trends. Front Mech Eng 13(2):121–136. https://doi.org/10.1007/s11465-018-0483-0
2. International Federation of Robotics (2018) The impact of robots on productivity, employment and jobs. https://ifr.org/papers. Accessed 6 Jul 2021
3. SPARC (2016) Robotics 2020 Multi-annual roadmap. https://www.eu-robotics.net/cms/upload/topic_groups/H2020_Robotics_Multi-Annual_Roadmap_ICT-2017B.pdf. Accessed 06 Jul 2021
4. Hägele M, Nilsson K, Pires JN, Bischoff R (2016) Industrial robotics. In: Siciliano B, Khatib O (eds) Springer Handbook of Robotics. Springer, Berlin, Heidelberg, pp 1385–1421
5. Heuss L, Lux-Gruenberg G, Hammerstingl V, Schnös F, Rinck P, Reinhart G, Zäh M (2018) Mobile autonome roboter in der smart factory: Dynamische planung und adaption mobiler roboter für die flexible produktion. wt Werkstattstechnik Online 108(9):574–579
6. Heuss L, Blank A, Dengler S, Zikeli GL, Reinhart G, Franke J (2019) Modular robot software framework for the intelligent and flexible composition of its skills. In: Ameri F, Stecke KE, Cieminski G, Kiritsis D (eds) Advances in Production Management Systems. Production Management for the Factory of the Future. APMS 2019. IFIP Advances in Information and Communication Technology, vol 566. Springer, Cham. https://doi.org/10.1007/978-3-030-30000-5_32
7. Heuss L, Reinhart G (2020) Integration of autonomous task planning into reconfigurable skill-based industrial robots. In: 2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), pp 1293–1296. https://doi.org/10.1109/ETFA46521.2020.9212005
8. Lentin J, Cacace J (2018) Mastering ROS for robotics programming: Design, build, and simulate complex robots using the Robot Operating System, 2nd edn. Packt Publishing Limited, Birmingham
9. Bohren J, Cousins S (2010) The smach high-level executive [ros news]. IEEE Robot Autom Mag 17(4):18–20. https://doi.org/10.1109/MRA.2010.938836
10. Stonier D, Staniaszek M., Usmani N (2021) py_trees_ros. http://wiki.ros.org/py_trees_ros. Accessed 6 Jul 2021
11. Cashmore M, Fox M, Long D, Magazzeni D, Ridder B, Carrera A, Palomeras N, Hurtos N, Carreras M (2015) Rosplan: Planning in the robot operating system. In: Proceedings of the 25th International Conference on Automated Planning and Scheduling, pp 333–341
12. Huckaby J, Christensen H (2012) A taxonomic framework for task modeling and knowledge transfer in manufacturing robotics. AAAI Workshop - Technical Report, pp 94–101
13. Pedersen MR, Nalpantidis L, Andersen RS, Schou C, Bøgh S, Krüger V, Madsen O (2016) Robot skills for manufacturing: From concept to industrial deployment. Robot Comput-Integr Manuf 37:282–291. https://doi.org/10.1016/j.rcim.2015.04.002
14. Backhaus J, Reinhart G (2017) Digital description of products, processes and resources for task-oriented programming of assembly systems. J Intell Manuf 28(8):1787–1800. https://doi.org/10.1007/s10845-015-1063-3
15. Hammerstingl V, Reinhart G (2018) Skills in assembly. https://mediatum.ub.tum.de/1428286. Accessed 6 Jul 2021
16. Herrero H, Moughlbay AA, Outón JL, Sallé D, de Ipiña KL (2017) Skill based robot programming: Assembly, vision and workspace monitoring skill interaction. Neurocomputing 255:61–70. https://doi.org/10.1016/j.neucom.2016.09.133
17. Halt L, Tenbrock P, Nägele F, Pott A (2018) On the implementation of transfareable assembly applications for industrial robots. In: ISR 2018, 50th International Symposium on Robotics, pp 1-7
18. Stenmark M, Malec J, Stolt A (2015) From high-level task descriptions to executable robot code. Adv Intell Syst Comput 323:189–202. https://doi.org/10.1007/978-3-319-11310-4_17
19. Sorensen LC, Mathiesen S, Waspe R, Schlette C (2020) Towards digital twins for industrial assembly - improving robot solutions by intuitive user guidance and robot programming. In: 2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), pp 1480–1484. https://doi.org/10.1109/ETFA46521.2020.9212072
20. drag and bot (2021) drag & bot website. https://www.dragandbot.com/de/. Accessed 6 Jul 2021
21. Wenger M, Eisenmenger W, Neugschwandtner G, Schneider B, Zoitl A (2016) A model based engineering tool for ros component compositioning, configuration and generation of deployment information. In: 2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA), pp 1–8. https://doi.org/10.1109/ETFA.2016.7733559
22. Stenmark M, Malec J (2015) Knowledge-based instruction of manipulation tasks for industrial robotics. Robot Comput-Integr Manuf 33:56–67. https://doi.org/10.1016/j.rcim.2014.07.004
23. Steinmetz F, Wollschlager A, Weitschat R (2018) Razer–a hri for visual task-level programming and intuitive skill parameterization. IEEE Robot Autom Lett 3(3):1362–1369. https://doi.org/10.1109/LRA.2018.2798300
24. Franka Emika (2021) Franka Emika website. https://www.franka.de/. Accessed 6 Jul 2021
25. Huckaby J, Vassos S, Christensen HI (2013) Planning with a task modeling framework in manufacturing robotics. In: 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp 5787–5794. https://doi.org/10.1109/IROS.2013.6697194
26. Kootbally Z, Schlenoff C, Lawler C, Kramer T, Gupta SK (2015) Towards robust assembly with knowledge representation for the planning domain definition language (PDDL). Robot Comput-Integr Manuf 33:42–55. https://doi.org/10.1016/j.rcim.2014.08.006
27. Kootbally Z, Schlenoff C, Antonishek B, Proctor F, Kramer T, Harrison W, Downs A, Gupta S (2018) Enabling robot agility in manufacturing kitting applications. Integr Comput-Aid Eng 25(2):193–212. https://doi.org/10.3233/ICA-180566
28. Rovida F, Crosby M, Holz D, Polydoros A, Großmann B, Petrick R, Krüger V (2017) Skiros—a skill-based robot control platform on top of ros. In: Koubaa A (ed) Robot Operating System (ROS). Studies in Computational Intelligence, vol 707. Springer, Cham. pp 121–160. https://doi.org/10.1007/978-3-319-54927-9_4
29. Berger J, Colceriu C, Blank A, Franke J, Haerdtlein C, Hellig T, Henrich D, Heuss L, Hiller M, Krae M, Leichtmann B, Lottermoser A, Lu S, Nitsch V, Reinhart G, Riedl M, Roder S, Schaefer K, Schilp J, Vogt L, Zaeh MF (2021) Abschlussbericht: FORobotics - mobile ad-hoc kooperierende Roboterteams. https://doi.org/10.24406/igcv-n-624794. http://publica.fraunhofer.de/dokumente/N-624794.html. Accessed 19 Sep 2021
30. Ingrand F, Ghallab M (2017) Deliberation for autonomous robots: a survey. Artif Intell 247:10–44. https://doi.org/10.1016/j.artint.2014.11.003

31. Nokleby C (2016) Omac packml unit machine implementation guide. https://www.omac.org/wp-content/uploads/2016/11/PackML_Unit_Machine_Implementation_Guide-V1-00.pdf. Accessed 16 Nov 2021
32. Paxton C, Hundt A, Jonathan F, Guerin K, Hager GD (2017) CoSTAR: Instructing collaborative robots with behavior trees and vision. In: 2017 IEEE International Conference on Robotics and Automation (ICRA), pp 564-571. https://doi.org/10.1109/ICRA.2017.7989070
33. Rovida F, Grossmann B, Krueger V (2017) Extended behavior trees for quick definition of flexible robotic tasks. In: 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp 6793-6800. https://doi.org/10.1109/IROS.2017.8206598
34. Ghallab M, Nau DS, Traverso P (2016) Automated planning and acting. Cambridge University Press, New York. https://doi.org/10.1017/CBO9781139583923
35. Sucan IA, Chitta S (2021) Moveit. https://moveit.ros.org/. Accessed 6 Jul 2021
36. Michniewicz J, Reinhart G, Boschert S (2016) CAD-based automated assembly planning for variable products in modular production systems. Proc CIRP 44:44–49. https://doi.org/10.1016/j.procir.2016.02.016
37. Gonnermann C, Reinhart G (2019) Automatized setup of process monitoring in cyber-physical systems. Proc CIRP 81:636–640. https://doi.org/10.1016/j.procir.2019.03.168