2016 IEEE International Conference on Automation Science and Engineering (CASE)
Fort Worth, TX, USA, August 21-24, 2016

Skill Parametrization Approaches and Skill Architecture for Human-Robot Interaction

Franz Steinmetz[1] and Roman Weitschat[2]

Abstract— **There is an ongoing shift in industries from mass production to low-batch-production with highly individualized goods. This increases the programming effort required for the producing machines and robots, which is currently carried out by robot experts. For keeping the production economical, new programming approaches are required, allowing shop-floor workers to instruct robots. One approach is to develop robotic skills, which are pre-programmed software modules that only need to be parametrized by the shop-floor user. In this paper, a new software architecture for robot skills is presented, which aims at robustness and human-robot interaction. In addition, four basic demands on the skill parametrization are described that fasten up the process and increase intuitiveness for the user. We give several examples and implement a screwing skill and a pick & place skill, which are demonstrated in two case studies.**

I. Introduction

The shift from mass production to high customization in industrial manufacturing leads to a decreasing number of batch sizes. Up to now, customized machines or robots have been used for the production of a high number of workpieces. The set-up times for a robot work-cell are in a good ratio compared to the production period for large production volume, whereas a certain proportion of set-up time is caused by programming the robot. For lower batch sizes, this kind of manufacturing is economically not feasible. Therefore, shop-floor workers are employed when product periods and number of manufactured products are low and an automation is not profitable. Robots are an important component for increasing the productivity and efficiency of a manufacturing process. Hence, the programming of a robot must be simplified such that non-experts are able to program it. In addition, the reprogramming time of a robotic system needs to be reduced significantly.

One approach is simplifying the programming expenditure by using robot skills [1]. A robot skill is a kind of software module, which is readily implemented by a robot expert and only needs to be parametrized by a user. One example is a screwing skill that only requires the parameters "screw type" and "target screw pose". The main idea is to compose robot skills with all information, behavioral logic, recover strategies, controllers, etc., which are needed to execute a certain task. The non-expert can sequence these single robot skills to obtain a complex autonomous or interactive robot task with a bounded set of parameters.

[1]Franz Steinmetz franz.steinmetz@dlr.de
[2]Roman Weitschat roman.weitschat@dlr.de
Both authors are with Robotics and Mechatronics Center, Deutsches Zentrum für Luft- und Raumfahrt (DLR, German Aerospace Center), Oberpfaffenhofen-Wessling, Germany

While [1] focuses on fully autonomous robots, e. g. for tending machines, our research targets at human-robot interaction. Within the next years, robots will not be sophisticated enough to replace all workers participating in a production process. We rather foresee a more intense collaboration between a human, being very flexible, and a robot, being fast and accurate. Collaboration leads to a significant reduction of production time, while keeping the human in the loop.

In order to make the robot skill approach applicable in an industrial scenario, we contribute to this research in two ways:

1) First, we developed a software architecture for skills to make human-robot interaction more robust and allows for process control.
2) Second, we describe strategies and approaches for a fast and intuitive parameterization process.

In the following Section II, research related to this paper is introduced. Then, Section III presents the developed software architecture for skills and Section IV describes our four demands on skill parametrization. The hardware and software implementation is given in Section V. We demonstrate the implemented skills in Section VI and finally draw a conclusion in Section VII.

II. Related work

A method to skill programming, similar to the one of our paper, is introduced in [2], where *Behavior Trees* are used for modeling and combining *capabilities* in a framework called *CoSTAR*. Another example is the *rob@work* robot assistant, applying symbolic reasoning for the execution of tasks that are instructed using a man-machine-interface [3]. The paradigm behind these approaches is called *task-level programming*.

Archibald and Petriu did pioneering work in an approach closely related to robot skills [4]. Their skills can be parametrized and have both pre- and postconditions. In addition, an iconic programming language for sequencing is implemented. The approach is later picked up by Bøgh et al. in [5], establishing the terminology and model of robot skills (similar to Fig. 1). They further refine this skill model [1] and show how skills can generically be used on different hardware [6].

According to [1], a skill contains a combination of *skill primitives*. A skill primitive is typically a sensory input or a single robot motion, described using the *Task Frame Formalism* (TFF) [7]. There exist different approaches on how these primitives are combined. One concept is the usage of nested generic components as presented in [8]. These

978-1-5090-2409-4/16/$31.00 ©2016 IEEE

action components also define end conditions, similar to our action blocks. Yet, there exists no exit handler allowing to leave the component in a defined manner. Our software architecture uses stop conditions. This is similar to [9], where stop conditions are used for skill primitives and are connected to *arcs* defining the logical flow. However, no concurrency is allowed in [9], which restricts the parallel observation of different events.

The aforementioned approaches use graphical representations for the combination of components instead of textual ones. When utilizing textual programming languages, the number of software patterns is countless. The most popular ones haven been described by the so called *Gang of Four* (GoF) [10], e. g. *Observer*, *Model-View-Controller* or *Publisher-Subscriber*. Yet, the situation is different when using visual programming languages, often based on some kind of state machine concept like Harel statecharts [11]. Most of the common software patterns are not applicable here, therefore alternative approaches are required.

However, visual programming languages offer many advantages compared to traditional textual languages. They are often easy to learn and more intuitive to use. In addition, they support the creation of mental models of programs [12].

For the implementation and combination of skill primitives, we use our novel visual programming tool for defining the flow control, named RAFCON. It supports hierarchies and concurrencies in a state-machine like concept.

When employing a skill for a specific task as a non-expert, the pre-implemented skill has to be parameterized first. For this process, different approaches have been explored. In [4], a graphical user interface (GUI) was used for parameterization, which requires the use of keyboard and mouse. *Programming by demonstration* (PbD) is an alternative to this traditional approach and has been investigated for years [13]. There are a various number of options, such as *kinesthetic teaching* (for teaching trajectories [14], also in combination with force profiles [15]) or by showing and watching [16]. Pedersen et al. started to make use of these intuitive teaching approaches for the parametrization of skills. In [17], pointing gestures are used to determine desired objects. In [18], more gestures are added and a GUI for sequencing skills on a tablet is introduced.

III. Software architecture for skills

Robotic skills share a common composition as depicted in Fig. 1. First, precondition checks ensure a (world) state for which the skill is designed. At the end, postcondition checks validate the effect of the skill on that state [1]. The state change is achieved in the execution block, consisting of (device) primitives. In order to make skills robust, we developed a software framework for combining these primitives. This concept is described in the following.

For the display of the software components in this paper, an abstract graphical representation is used. This slightly differs from the actual state machine representation in RAFCON, but tries to carve out the concepts.

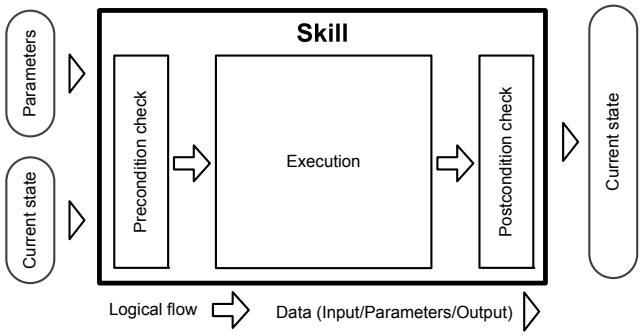


Fig. 1. Simple robot skill model, based on [1]. Preconditions must be fulfilled before the execution starts. The execution changes the world state. The success is checked using the postconditions.

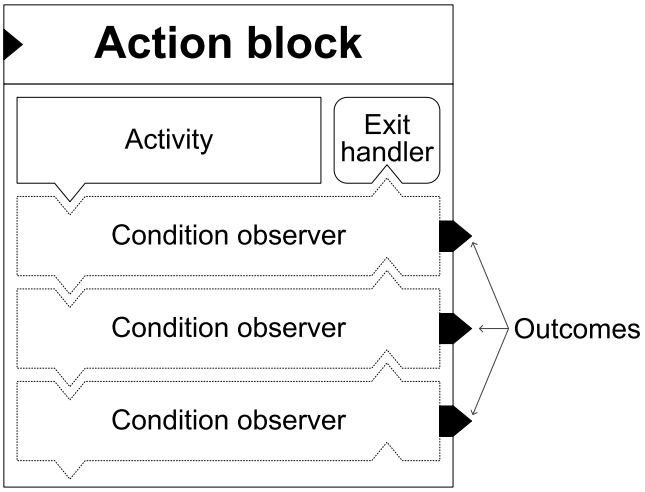


Fig. 2. Developed action block pattern for skill primitives. Parallel running observers can stop the execution of the activity and call the exit handler.

The central idea of our concept is a programming scheme named *action block*, which is illustrated in Fig. 2. An action block changes the system state and consists of an *activity* and an *exit handler*, as well as *condition observers*. The activity is executed, when the execution logic enters the action block. Each condition observer has an *outcome* and checks for a single success or failure case in parallel to the execution of the activity. If one of the conditions is fulfilled, the exit handler is executed and the action block is left on the outcome of the triggered condition. In other words, satisfied conditions cause the activity to preempt and trigger the preemption routine, namely the exit handler. This handler is especially intended for gracefully stopping the activity and therefore ensuring that e. g. resources are freed, motions stopped and the system state updated.

Outcomes can be connected with a *transition* to another action block or other primitives that do not alter the system state. A transition defines the logical flow of the execution.

It is possible to nest action blocks in a hierarchical manner, as shown in Fig. 3. The activity of the outer action block “Screw Operation” consists of two inner action blocks (“Move to”, “Screw”) and other primitives (“User dialog”). If

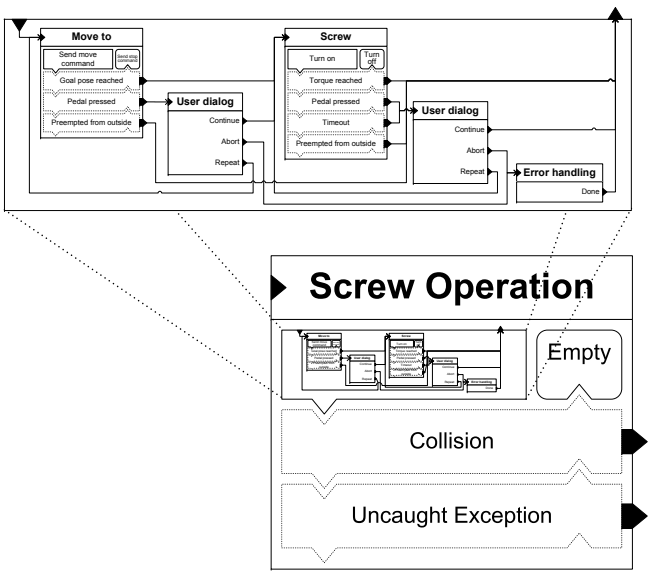


Fig. 3. Action blocks can be nested. The hierarchical structure allows for example common condition handling.

an outer action block preempts, the primitives are preempted from the inside out. First, the exit handler of the innermost running action block is executed, then the hierarchy (outer activity) is left and the exit handler of the next higher action block is called.

This architecture has clear advantages compared to a simple sequential/branching logic, as it allows for a proper process monitoring and control. Process monitoring is achieved by allowing concurrently running observers, each responsible for different sensors, variables or other inputs. Process control is gained in two ways. First of all, action blocks are always exited in a defined state. The system state when leaving an action block is the same as the system state before the start of the block, e. g. the screwer is off before/after the screwing action block. This is also critical for the safe interaction with the environment. In the case of collisions, the robot is supposed to stop, no matter what it is currently doing. When collisions are observed on the highest hierarchy level, the architecture stops all activities from the inside out. Process control and robustness is also attained by allowing the custom handling of different events: Events that trigger conditions can be caught on every hierarchy level and each condition observer has its own outcome. This grants to handle the event where it is most reasonable, e. g. a timeout is observed on a low level, such as the screwing action block, as the timeout is related to that block, whereas a collision might be caught on a higher level, as it is related to all movements. Concerning the unique outcomes, the timeout can be handled in a different way than the reach of certain torque threshold.

As many conditions as possible should of course be handled automatically using some kind of strategy for a maximum of autonomy. However, a skill cannot decide for every event what to do. Especially in cases of failure conditions (e. g. collisions), the user must be in the loop

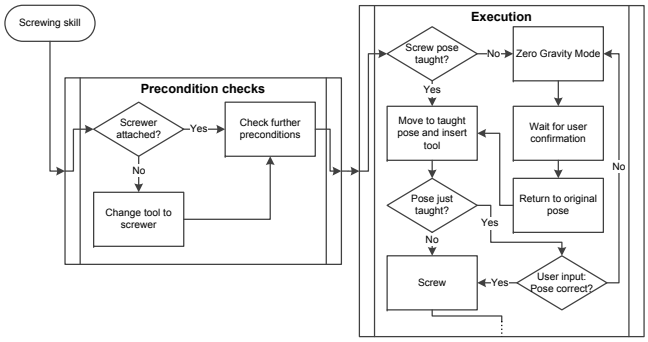


Fig. 4. Simplified part of a screwing skill. The parametrization of the screw pose is entangled in the skill.

and asked for a choice between predefined solutions such as continue, repeat or abort (see again Fig. 3).

IV. Parametrization approach

Another topic usually neglected in the field of robot skills is the parametrization process, which is a key part of the skill. Proper parametrization integration can lead to short teach-in times and thus fast programming of the robot.

Therefore, we developed approaches for a successful parametrization strategy: (i) Entangling of parametrization in the skill, (ii) direct parameter application and verification, (iii) parameter reduction by automatic parameter derivation and (iv) use of different parameter sets depending on the user.

(i) Parameters should be taught at the instance at which they are needed. Therefore, in the teach-in phase, a skill is in a semi-autonomous mode. It is executed until it requires a parameter from the user. At this point, the execution pauses and the parameter is taught by the user; afterwards the skill continues. If more parameters are required, the procedure repeats. Applied to a simple screwing skill (see Fig. 4), the skill should first automatically change the tool to the screwer. Then the skill oughts to pause and request the user to teach the pose of the screw in gravity compensation mode. By this, the user directly sees the environmental state, in which the parameter is needed. In the case of the given example, this is mandatory, as the user requires the correct tool to define the pose of the screw.

(ii) Entangling the parametrization in the skill is related to the approach of direct parameter application and verification. If possible, after teaching a parameter, the robot should immediately use and apply it. Subsequently, the user is asked, whether the execution with the parameter was successful. If this is confirmed, the skill continues, otherwise the teach-in of the parameter is repeated. This approach can be applied to the previously mentioned example as demonstrated in Fig. 4: When the user confirms the screw pose, the robot tries to insert the bit at the specified location. The user confirms or declines the success of the trial with a dialog window. Using this approach, the user receives feedback directly and can control the correctness of the specified parameter.

(iii) A reduction of the number of parameters further speeds up parametrization. The premise should be to require as few parameters as possible. This can be achieved by automatic derivation of parameters from a given one. For a sophisticated screwing skill, the type of the screw, the fastening torque, the rotation speed, the timeout or more parameters are needed for the execution. However, by introducing some static knowledge about screws into the skill, those parameters can be reduced to only the type of the screw. Using the static knowledge, e. g. with a lookup table in a data base relating screw types to fastening torques, the desired torque can be looked up automatically.

(iv) Further reduction of parameters can be achieved by using different parameter sets, depending on the user. The shop-floor worker only needs to specify parameters being mandatory for the execution. Those obligatory parameters belong to the *primary parameter set*. The teach-in of these parameters has to follow the upper mentioned approaches. Parameters of the *secondary parameter set* have default values, which are safe and not critical. Experienced workers may alter these parameters such as maximum speed or parameters that were automatically derived (see (iii)) in order to further tune a skill. Hereby, a GUI for parametrization is sufficient. The third set of parameters is what [18] calls *internal parameters*. These are fixed and specified by the skill programmer. Examples are impedance values, controller gains, virtual walls, maximum acceleration and so on.

V. Implementation

This section introduces both the hardware and software used for the case studies of Section VI.

A. Hardware

We are using a LWR 4+ [19] as robotic platform. The tool has to be changed manually. Currently, we are using three different tools: a suction gripper (for chocolate bars), a simple hook (for Kanban boxes) and a screwer with force-torque sensor. A foot pedal serves as further input device.

B. Software

For the software implementation, the aforementioned graphical flow control tool RAFCON is used. This eases the software development and helps understanding the structure of a skill, as textual code is abstracted in named blocks, similarly to a state-machine approach. In addition, the currently executed block is highlighted, which is helpful for debugging purposes.

The implemented skills are intended for a small collaborative assembly task. In our scenario, the robot takes a specific box with screws from a shelf and puts it on the table. The shop-floor worker takes some of the screws and starts to insert them loosely in a workpiece. After returning the box, the robot tightens the inserted screws.

This requires two common skills, *pick & place* for handing over the box and *screwing*. Pick & place is often split in separate skills *pick* and in *place*. Here it is merged in one skill, as in our scenarios, there is typically no other skill required in between the two. Keeping them together reduces the effort of the user for sequencing and has advantages for the parametrization process.

1) Screwing skill: The task of the screwing skill is supporting the shop-floor worker in the time consuming action of tightening a screw, which was previously inserted in existing screw threads. The focus in the screwing skill is more on the software architecture, which was implemented according to Section III. The actual screwing part is an action block named “Screwing”. In the activity, the screwer is simply turned on and in the exit handler switched off again. Three condition observers are used: the applied torque is monitored to stop when the screw is fully inserted, the time is counted to stop if the action takes too long (timeout) and the foot pedal is observed, to stop in case the user wants to pause the action by pressing a pedal. The torque observer uses the force-torque sensor to monitor the fastening torque of the screw. A trigger of this observer is desired, as it means the screw is fully inserted. If triggered, the execution logic goes to the next step (move bit out of screw). The other two observers currently both lead to a user dialog, with the options to continue (go to next step), repeat (reset timer and start screwing again) or to abort (stop with an error). A simplified version of this logic is shown in Fig. 3.

2) Pick & Place skill: The pick & place skill illustrates a further approach for reducing the number of required parameters (see Section IV). As a generic skill for picking and placing different objects in different fixtures with different tools requires a vast variety of strategies (pick from above, pull out, ...), we retrieve this information from the user, parameterizing the skill via kinesthetic teaching. Internally, the skill requires a pre-pick-pose, a pick-pose, a post-pick-pose, a pre-place-pose, a place-pose and a post-place-pose, thus six different parameters.

To reduce this number, instead of storing single poses specified by the user (as it is done in [20]), three trajectories are recorded: First, the user guides the tool to the pick-pose and confirms, which activates the tool (e. g. activate the vacuum gripper). Then, the user moves the tool to the place-pose and confirms again, deactivating the tool. Finally, the user releases the tool from the object and confirms a last time. From the three trajectories, the six poses are inferred automatically. The trajectories and interesting poses are schematically shown in Fig. 5. For example, the pick-pose is the last pose of the first trajectory and the pre-pick-pose is the last pose within the first trajectory, having a minimum distance to the pick-pose of 5 cm. This principle is demonstrated in Section VI.

One may argue that there are far more sophisticated approaches for this type of trajectory learning. There are for example Learning from Demonstration (LfD) methods using Hidden Markov Models (HMM) and Dynamic Movement Primitives (DMP) being able to learn from multiple demonstrations and even segment demonstrations into sub-skills [21]. While being versatile and generic, these methods are way more complex than the approach used in our skill. As we know beforehand that a pick & place skill is being

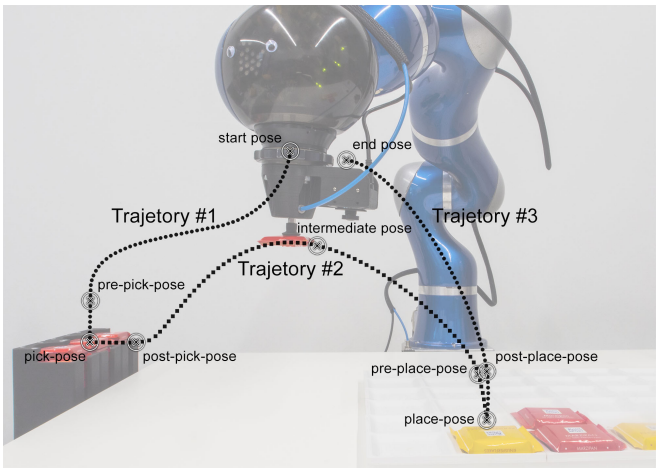


Fig. 5. From three trajectories, recorded from user demonstration, the required poses are automatically retrieved.

demonstrated, we can apply that knowledge. In this case, it is the sufficiency of a fixed number of via-points at certain locations. For the case given, common interpolators are adequate, reducing the complexity required for path planning and working robustly. In addition, trajectories generated from LfD approaches are often smoothed compared to the demonstration(s), which is not desired in accurate pick and place movements. Furthermore, we are only interested in small subparts of the trajectory and do not require the movement to be copied entirely.

VI. Case Studies

Two skills were implemented in order to test the software architecture and to verify the improvements in the parametrization process.

A. Screwing

The screwing skill is kept relatively simple. The user is required to teach only a single parameter, the screw pose. After defining this pose using kinesthetic teaching, the screwing skill autonomously moves the bit in front of the screw. The robot is controlled by a Cartesian torque controller with a PD-behavior, which implies an absolute error to the desired pose. Therefore, positioning accuracy is limited and special methods could be implemented to detect the screw under this uncertainty. However, in our case we have chosen to insert the screwing bit by hand in gravity compensation mode, which is faster than sensing the screw with the torque sensors of the robot and is a straightforward approach in a human-robot interaction context. The efficiency is still given, as the actions requiring most of the time (reaching the screw and the actual screwing) are still automated. Subsequent to the confirmation, the skill starts screwing. The attached video demonstrates the skill (Experiment 2).

Typically, the skill finishes successfully when the screw is fully inserted. Hereby, the torque observer stops when measuring a certain threshold limit. This causes the exit handler to be executed, stopping the screwer. The screw bit is moved out of the screw and the skill ends.

Two more scenarios cause a stop of the screwer by the exit handler. First, in some situations a timeout occurs, e. g. when the screw slips. Second, the user presses the foot pedal. In both cases, the user is then asked for the desired next step. The three options “continue”, “repeat” and “abort” can be selected with a mouse/keyboard or by pressing the according foot pedal having three possible inputs. This allows full control by the user, who can easily pause the execution and quickly react to failure cases.

This trivial example demonstrates the advantages of action blocks. It is sufficient to declare one exit handler for all circumstances. Nevertheless, each case can in consequence be treated individually. Moreover, the user was required to teach only a single parameter, the screw pose.

B. Pick & Place

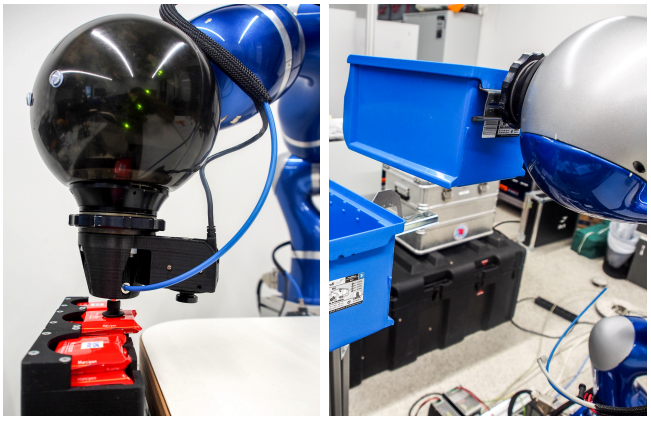
The pick & and place skill was tested in two different scenarios. In the first, the suction gripper was used to slide a chocolate bar out of a fixture and put it onto the table. In the second, the hook was used to lift a Kanban box out of a hanger and put it also on the table. The user can choose between three chocolate bar stacks and two Kanban boxes, respectively. The place-pose can be chosen arbitrarily. Both scenarios are shown in the attached video (Experiment 1).

This required varying pick and place strategies. The chocolate bar is picked from above and must then be moved sideways out of the fixture (see Fig. 6a). Placing is again from top. For picking the box, the hook must be inserted from below and then lifted upwards (see Fig. 6b). Placing is from the top and then further down.

As explained in Section V-B.2, the poses required for the pick and the place strategies are determined from the user demonstration. This works reliably in both scenarios. In consequence to the object being “grabbed” when confirming the pick-pose, the user is forced to move the tool appropriately to get the object out of the fixture. If this was not the case, the user might move the tool upwards (in case of the chocolate bar), as he is instructed just to move to the place-pose. Also the orientation of the tool can be controlled easier with an attached object. This is important for the box, as it would be emptied if tilted too much.

The extracted poses are usually sufficient to generate a proper trajectory. In some cases, a *post-post-place-pose* is helpful, to safely move away from the object. One or more intermediate poses between the pick- and the place-pose can be used to avoid obstacles. Those additional poses do not cause any changes for the user, but only in the parameter extraction routine.

In the user’s point of view, only three parameters (poses) are required. Hereby, one is relieved from defining a specific object, pick and place strategies, controller parameters, etc., but all this is inferred from the demonstrations or included in the skill.



(a) (b)

Fig. 6. Picking strategies vary depending on the tool, object and environment. In (a), the fixture requires the chocolate to be moved out sideways. In contrast in (b), the hook must be inserted accurately from below.

VII. Conclusions

In this paper, we presented a software architecture for combining skill primitives, named action blocks. As the action block pattern allows for process monitoring and control, skills can be implemented more robustly. Furthermore, the user has full control over the execution and all events are handled in a defined way.

In addition, we described how the parametrization process should be implemented in a skill. The entangling of this process with the skill, combined with a reduction of skill parameters, makes the usage of skills very intuitive and fast. The pick & place experiment gave an example how the number of parameters can be reduced while keeping the skill generic and suitable for different scenarios.

As we are at an early phase of skill development, there is still room to make our skills more robust, i. e. handling more (failure) cases autonomously. This also requires a further integration of sensors and a world model.

Acknowledgement

This work has been funded by the Helmholtz-Gemeinschaft Germany as part of the project RACELab.

References

[1] M. R. Pedersen, L. Nalpantidis, R. S. Andersen, C. Schou, S. Bøgh, V. Krüger, and O. Madsen, "Robot skills for manufacturing: From concept to industrial deployment," *Robotics and Computer-Integrated Manufacturing*, 2015.

[2] K. Guerin, C. Lea, C. Paxton, and G. Hager, "A framework for end-user instruction of a robot assistant for manufacturing," in *Robotics and Automation (ICRA), IEEE International Conference on*, 2015, pp. 6167–6174.

[3] E. Helms, R. D. Schraft, and M. Hägele, "rob@work: Robot assistant in industrial environments," in *Robot and Human Interactive Communication. Proceedings. 11th IEEE International Workshop on*, 2002, pp. 399–404.

[4] C. Archibald and E. Petriu, "Skills-oriented robot programming," in *Proceedings of the International Conference on Intelligent Autonomous Systems IAS-3*, 1993, pp. 104–15.

[5] S. Bøgh, O. S. Nielsen, M. R. Pedersen, V. Krüger, and O. Madsen, "Does your robot have skills?" in *The 43rd Intl. Symp. on Robotics (ISR2012)*, 2012.

[6] M. R. Pedersen, L. Nalpantidis, A. Bobick, and V. Krüger, "On the integration of hardware-abstracted robot skills for use in industrial scenarios," in *2nd International Workshop on Cognitive Robotics Systems: Replicating Human Actions and Activities*, 2013, pp. 1166–1171.

[7] H. Bruyninckx and J. D. Schutter, "Specification of force-controlled actions in the "Task Frame Formalism" - A synthesis," *Robotics and Automation, IEEE Transactions on*, vol. 12, no. 4, pp. 581–589, Aug. 1996.

[8] U. Thomas, G. Hirzinger, B. Rumpe, C. Schulze, and A. Wortmann, "A new skill based robot programming language using UML/P Statecharts," in *Robotics and Automation (ICRA), IEEE International Conference on*, 2013, pp. 461–466.

[9] U. Thomas, B. Finkemeyer, T. Kroger, and F. Wahl, "Error-tolerant execution of complex robot tasks based on skill primitives," in *Robotics and Automation. Proceedings. ICRA '03. IEEE International Conference on*, vol. 3, 2003, pp. 3069–3075.

[10] J. Vlissides, R. Helm, R. Johnson, and E. Gamma, "Design patterns: Elements of reusable object-oriented software," *Reading: Addison-Wesley*, vol. 49, no. 120, p. 11, 1995.

[11] D. Harel, "Statecharts: A visual formalism for complex systems," *Science of computer programming*, vol. 8, no. 3, pp. 231–274, 1987.

[12] R. Navarro Prieto and Jose J. Cañas, "Are visual programming languages better? The role of imagery in program comprehension," *International Journal of Human-Computer Studies*, vol. 54, no. 6, pp. 799–829, June 2001.

[13] A. Billard, S. Calinon, R. Dillmann, and S. Schaal, "Robot programming by demonstration," in *Springer handbook of robotics*. Springer, 2008, pp. 1371–1394.

[14] S. Calinon, F. Guenter, and A. Billard, "On learning, representing, and generalizing a task in a humanoid robot," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 37, no. 2, pp. 286–298, 2007.

[15] F. Steinmetz, A. Montebelli, and V. Kyrki, "Simultaneous Kinesthetic Teaching of Positional and Force Requirements for Sequential In-Contact Tasks," in *Humanoid Robots (Humanoids), 15th IEEE-RAS International Conference on*, 2015.

[16] Y. Kuniyoshi, M. Inaba, and H. Inoue, "Learning by watching: Extracting reusable task knowledge from visual observation of human performance," *Robotics and Automation, IEEE Transactions on*, vol. 10, no. 6, pp. 799–822, 1994.

[17] M. R. Pedersen, C. Høilund, and V. Krüger, "Using human gestures and generic skills to instruct a mobile robot arm in a feeder filling scenario," in *Mechatronics and Automation (ICMA), International Conference on*, 2012, pp. 243–248.

[18] M. Pedersen, D. Herzog, and V. Kruger, "Intuitive skill-level programming of industrial handling tasks on a mobile manipulator," in *Intelligent Robots and Systems (IROS), IEEE/RSJ International Conference on*, 2014, pp. 4523–4530.

[19] C. Loughlin, A. Albu Schäffer, S. Haddadin, C. Ott, A. Stemmer, T. Wimböck, and G. Hirzinger, "The DLR lightweight robot: design and control concepts for robots in human environments," *Industrial Robot: an international journal*, vol. 34, no. 5, pp. 376–385, 2007.

[20] B. Akgun, M. Cakmak, Jae Wook Yoo, and A. Thomaz, "Trajectories and keyframes for kinesthetic teaching: A human-robot interaction perspective," in *Human-Robot Interaction (HRI), 7th ACM/IEEE International Conference on*, 2012, pp. 391–398.

[21] S. Niekum, S. Osentoski, G. Konidaris, and A. G. Barto, "Learning and generalization of complex tasks from unstructured demonstrations," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, 2012, pp. 5239–5246.