Between Input and Output: The Importance of Modelling Transients in Meal Preparation Tasks

Michaela Kümpel[1,*], Vanessa Hassouna[1], Alina Hawkin[1] and Michael Beetz[1]

[1]*Institute for Artificial Intelligence, University of Bremen, Am Fallturm 1, 28359 Bremen, Germany*

Abstract

We are moving closer to autonomous robots preparing meals. While restaurant robots in static environments already are successfully performing single actions like making pizza, the goal is to enable robots to perform changing actions, in various environments and with any available object. Towards this goal, a methodology for creating actionable knowledge graphs that can be used to parameterise general action plans has been proposed. However, for extended failure handling towards fully automated action execution, we argue that transients need to be considered. A transient can be described as a transitory object in a task that is not the same as the input object anymore but not yet the output object of the task. For example, when pouring ingredients into a bowl to make the dough, the added ingredients form a mass of ingredients (here: a transient) that only becomes dough through mixing them. This work shows how transients can be modelled and how robots can integrate and possibly benefit from this modelling.

Keywords

Knowledge Representation, Meal Preparation Tasks, Transients, Agent Application

1. Introduction

Enabling robots to autonomously perform meal preparation tasks in changing environments, on varying objects and with any given tool is one of the goals of cognitive robotics research. Due to the variety of environments, actions, and objects, it is also very hard. Recent research has shown how general action plans can be used to tackle this challenge in the examples of setting the table [1] or performing task variations of cutting actions [2]. It has been shown how such general action plans can be parameterised using actionable knowledge graphs that link objects to action and environment information, thus making the contained knowledge actionable for agents [3]. What is more, the inclusion of object affordances and dispositions [4, 5], as well as image schema-based reasoning, has shown to help robots understand the conditions of the environment for an improved failure handling [6, 7].

However, if we consider robotic agents making pancakes, an action that has been the focus of many research projects in the past (such as in [8, 9]), we argue that for a fully automated task execution and extended failure handling, *transients* need to be considered. A *transient* is a

Woodstock'22: Symposium on the irreproducible science, June 07–11, 2022, Woodstock, NY

*Corresponding author.

michaela.kuempel@uni-bremen.de (M. Kümpel); hassouna@uni-bremen.de (V. Hassouna); hawkin@uni-bremen.de (A. Hawkin); beetz@cs.uni-bremen.de (M. Beetz)

0000-0002-0408-3953 (M. Kümpel); 0000-0003-1335-5698 (V. Hassouna); 0000-0003-1826-9983 (A. Hawkin); 0000-0002-7888-7444 (M. Beetz)

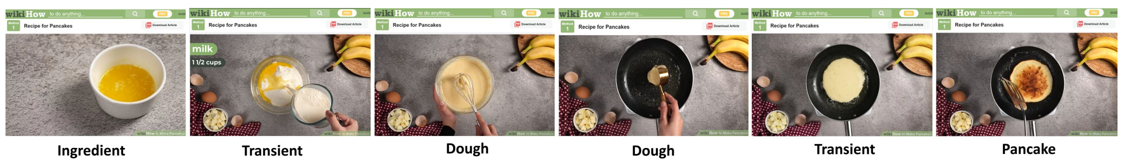

Figure 1: The action of making pancakes broken down into the object states that occur during task execution.

transitory object that exists during task execution. It is only apparent in the transitional state where it is not an input object anymore, but not an output object either.

Figure 1 shows the motivation behind modelling transients for the example of making pancakes using pictures from WikiHow instructions on how to prepare pancakes[1]. Here, we first have a number of separate **ingredients**. The ingredients are poured into a bowl and thus transform into a **transient** that is neither ingredient nor dough just yet. This transient is mixed until it transforms to **dough**. Some part of the dough is taken out of the bowl and poured into a pan, again forming a **transient** that is neither dough nor pancake. The transient is flipped and finally turns into a **pancake**.

Considering transients in action execution has the potential to enable robots to apply rules and thus reason about action execution and improve failure handling. For example, a robot that knows that the action of mixing dough includes input objects (ingredients), a transient object (mass of ingredients), and an output object (dough), can assign rules for action execution, such as setting time limits for mixing dough based on the temperature of the butter.

This work describes the logic of transients, how they might be modelled and the benefits for robots if transients are considered. We therefore show how the logic of transients can be integrated in a robot action plan.

2. Related Work

Many of the needed knowledge to perform tasks can be acquired using top-level ontologies such as the DOLCE+DnS Ultralite (DUL) foundational framework and its definitions of descriptions and situations [10] in combination with the Socio-physical Model of Activities (SOMA) [5], which models relations of actions, objects and agents at a given time and space, designed to provide robots with environment and activity knowledge so that they can relate an object to the task at hand through its dispositions and affordances. SOMA additionally allows for the integration of image schematic relations like SOURCE_PATH_GOAL (SPG), thereby supporting robots in reasoning about the functional relationships of objects and enabling a more dynamic action selection [7]. Actions in SOMA can be broken down into tasks, which again can be broken down into body movements of agents as shown in [3, 2]. While SOMA includes upper terms for processes, it has not been a research focus.

In contrast to this, the basic formal ontology (bio) defines occurrents and continuants over a period of time [11], which has been used to formally model processes that are defined as

[1]The WikiHow article on how to prepare pancakes is accessible at https://www.wikihow.com/Make-Pancakes

occurrences in bfo [12]. Here, a process relates to an execution plan and can be broken down into steps.

Both approaches based on SOMA and bfo have modelled input objects and output objects for tasks [4] and processes [12], but have not considered transients.

3. Logic of Transients

Let us continue to consider the action "making a pancake". For this, we first have to break down the action into its subtasks. Making a pancake can be broken down into the tasks of **grasping** the ingredient container, **pouring** the ingredient into the bowl and **placing** the empty container (these steps are repeated for all ingredients), **grasping** the whisk, **mixing** the dough, **placing** the whisk to then **grasping** a spoon, **scooping** some part of dough out of the bowl, **pouring** that dough part and **placing** the spoon to finally **grasping** a spatula, **flipping** the half-baked pancake, **transporting** the pancake to a plate and **placing** the spatula. If we closely look at these tasks, we can see that transients actually do not necessarily belong to a single task (which might have been expected) but can cover more than one task. In particular, when **pouring** a second ingredient into the bowl, the ingredients already form a transient. When the pouring task is completed, the transient still exists. Only after the **mixing** task is performed is the transient transformed into the dough.

If we consider a cutting action, however, only the cutting task includes a transient (the moment the knife touches the food object, it turns into a transient until the knife touches the supporting surface and the food object is cut into two pieces).

Looking at these two examples, it seems that the availability of input and output objects define transients. In particular, for the cutting task we can state that the input is a food object and the output are two food parts. For a pouring task, input and output objects are equal, unless more than one object is poured or the properties of the pouring destination (e.g. the heat of the pan) will lead to a transformation of the object.

Thus, the logic behind transients can be described formally as in Equation 1, 2, 3 and 4, which are stated in accordance to the SOMA ontology. In Equation 1 we consider tasks that have an input object and a result object, where the input object is different to the output object in its form (e.g. transitioning from a food object to food parts) or its quantitative measure (e.g. one object to two objects), then we can state that the input object transforms into a transient during task execution since the task triggers a process, in which the transient participates in. The process is stopped when the task is completed.

$$\begin{gathered}\forall\, i, t, o \,:\, Object,\ a \,:\, Task,\ p \,:\, Process \in X \,: \\ (has\_input\_object(a, i) \land has\_result\_object(a, o) \land i \neq o) \\ \rightarrow transient(t) \land triggers\_process(a, p) \land has\_participant(p, t) \land stops\_process(p, o)\end{gathered} \tag{1}$$

In Equation 2, we consider tasks that have the same input object and result object (like pouring). As discussed above, these tasks only start a process when performed multiple times and with different objects for every execution (e.g., if I pour water twice, the result object is still water and not transient). We can then state that the objects transform into a transient when performing the task for a second time, and the second object is added.

$$\begin{aligned}
&\forall\, i_1, i_2, t, o_1, o_2\ :\ Object,\ a\ :\ Task,\ p\ :\ Process \in X\ :\\
&(has\_input\_object(a, i_1) \wedge has\_result\_object(a, o_1) \wedge i_1 = o_1\\
&has\_input\_object(a, i_2) \wedge has\_result\_object(a, o_2) \wedge i_2 = o_2 \wedge o_1 \neq o_2\\
&\rightarrow transient(t) \wedge triggers\_process(o_2, p) \wedge has\_participant(p, t)
\end{aligned} \tag{2}$$

Another parameter that can trigger a process is object properties such as the pan's heat or the oven. In Equation 3, we, therefore, consider tasks that have the same input object and result object (like pouring) where the temperature as an object property leads to a transient. To add this additional restriction, we state that a task includes another object (the pan or oven) which is acted upon. This object has a temperature, and if the temperature is above a certain value, a baking process is started. In the same manner, a freezing process will start when performing a placing task and the object acted on has a temperature below a certain value.

$$\begin{aligned}
&\forall\, i, t, o, g\ :\ Object,\ h\ :\ Temperature\ a\ :\ Task,\ p\ :\ Process \in X\ :\\
&(has\_input\_object(a, i) \wedge has\_result\_object(a, o) \wedge i = o\\
&object\_acted\_on(a, g) \wedge has\_temperature(g, h) \wedge h > 80^\circ\\
&\rightarrow transient(t) \wedge triggers\_process(h, p) \wedge has\_participant(p, t)
\end{aligned} \tag{3}$$

Similarly, we can now state that consecutively performed tasks will stop the process started in Equation 2 or 3. For this, in Equation 4, we consider tasks that again have differing input and result objects and add the restriction that a transient already exists. Then, we can deduce that the transient is the task's input object, and the task's output object ends the process.

$$\begin{aligned}
&\forall\, i, o\ :\ Object,\ a\ :\ Task,\ p\ :Process \in X\ :\\
&(has\_input\_object(a, i) \wedge has\_result\_object(a, o) \wedge i \neq o \wedge (\exists\, t\ :\ transient(t) \wedge i = t)\\
&\rightarrow stops\_process(o, p)
\end{aligned} \tag{4}$$

4. Modelling Transients

With the described logic of transients, we can model transients as objects that participate in processes, as exemplarily depicted in Figure 2 for two tasks.

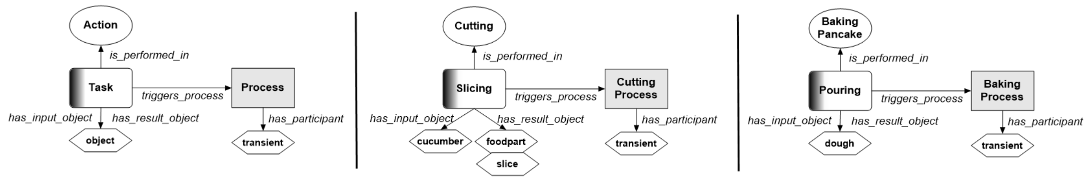

Figure 2: Model excerpt of Actions, Tasks, Processes and involved objects.

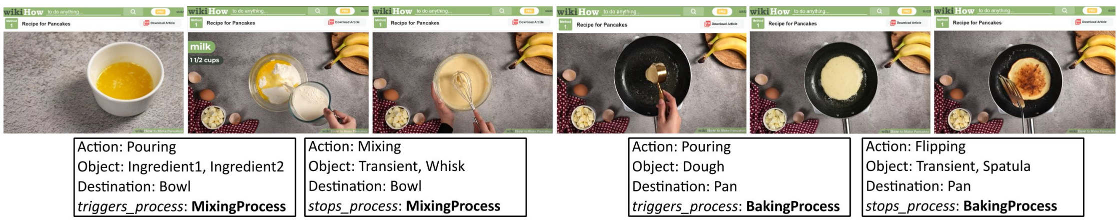
Action: Pouring
Object: Ingredient1, Ingredient2
Destination: Bowl
triggers_process: **MixingProcess**
Action: Mixing
Object: Transient, Whisk
Destination: Bowl
stops_process: **MixingProcess**
Action: Pouring
Object: Dough
Destination: Pan
triggers_process: **BakingProcess**
Action: Flipping
Object: Transient, Spatula
Destination: Pan
stops_process: **BakingProcess**
milk
1 1/2 cups
Recipe for Pancakes

Figure 3: Actions involved in making pancakes and their relation to processes.

For modelling transients, we propose to:

- include a relation such as *triggers_process* that links a task to a process
- integrate processes for every task that meets the criteria formulated in Equation 1, 2, 3 and 4
- integrates transients as participants of processes
- include a relation such as *stops_process* that links result objects to processes

As mentioned before, in DUL an action executes a task and can have a physical object as a participant. From SOMA we know that cutting is a task performed on an object. In recent work, input and result objects of tasks were proposed [4]. Regarding processes, in DUL, a *Process* $\sqsubseteq$ *Event* $\in$ *Entity*. Other work proposed input and output objects of processes [12]. In bfo, a *Process* $\sqsubseteq$ *Occurent* $\in$ *Enity*. Thus, transients can similarly be modelled for both top-level ontologies.

Following the proposed modelling approach, we enable agents to infer triggered processes and thus transient objects, as depicted in Figure 3. As described above, pouring the different ingredients into a bowl triggers a mixing process, which involves a transient. Similarly, pouring the dough into the hot pan triggers a baking process.

5. Benefits of Incorporating Transients into NEEMs

Narrative Enabled Episodic Memories (NEEMs) [13] [14] play a crucial role in helping robots learn from experience [15] [16]. By integrating knowledge about transients into NEEMs, a detailed record of transitional states and objects during task execution can be provided. This integration allows robots to analyse their actions, understand where transients occurred, and determine the success factors for completing tasks.

We propose to utilise Narrative Enabled Episodic Memories (NEEMs) to improve the robot's performance by tracking the transient state's duration and related conditions. If the robot is mixing [17] certain components for the first time, it is yet unknown for how long the mixing action should last to transition the mass from the transient to the goal state. The robot would have to potentially interrupt the mixing process, lift the tool in order not to obstruct the camera and perceive the current transient's state so that it can be determined if the mixing action needs to be prolonged or if the desired state has been reached. If the robot can generate NEEMs during this process, a knowledge database [14] can be filled with the obtained results. With this, whenever the robot is tasked to mix the same components, it would gain a much better estimate of how long the mixing process should last. This would result in less time to check the transient's state, potentially reducing spillage during the checking-of-current-mixing-state action. Accumulating this knowledge over time would also allow a robot to learn about the different properties of ingredients (such as the influence of the temperature of butter for mixing duration) and the success of different mixing techniques. The next time ingredients which have been mixed before need combining, it would be possible to estimate the mixing time more accurately in advance. Another benefit of generating NEEMs that consider transient states is that should an action end in that state; it would be possible to try and analyse why this might have happened. Maybe the robot dropped the whisk and could not pick it up again, or the consistency of the transient became too difficult for the robot to mix, or spillage occurred, and the experiment was aborted. NEEMs provide the potential to analyse such occurrences in hindsight, allowing the planning system to try and avoid them in the future.

6. Enabling Agents To Reason About Transients

Transients exist during transitional states (here: processes) in task execution, representing the intermediate stages between input and output objects. We utilize the Python version of the Cognitive Robot Abstract Machine (CRAM) [18], known as PyCRAM, to effectively enable robotic agents to reason about these transients. This adaptation assists robots in planning and executing complex, sequential tasks efficiently. PyCRAM encapsulates the logic and transitions within such tasks, significantly enhancing its ability to manage symbolic plans that account for transients. This adaptability is essential for robots to perform intricate tasks, allowing them to decompose tasks into smaller, manageable steps and adapt to fluctuating conditions. Central to PyCRAM are action designators that convert symbolic task descriptions into specific ROS action goals for robots. This structure enables robots to carry out high-level actions with an awareness of context and flexibility, which is crucial for handling complex tasks effectively.

Although PyCRAM offers a promising solution for understanding transients, there is still work to be done to fine-tune the framework to ensure it meets the demands of real-world applications. Integrating transients into PyCRAM requires careful consideration of action designators, task planning, and robotic reasoning, especially considering processes that span several tasks or involve multiple stages and transitional states. For instance, in a mixing task, a transient occurs when ingredients are combined but have yet to form a consistent mixture. Action designators that model triggered processes must include additional information on how tasks relate to processes and how transients transition from one state to another. Processes can then encompass information like the expected duration of a process, additional rules such as the influence of temperature on consistency, and desired results. One practical approach to integrating transients into PyCRAM is establishing rules that explain how transients are initiated, monitored, and concluded.

In our past work - ”Steps Towards Generalized Manipulation Action Plans - Tackling Mixing Task” [17], action designators break down the mixing task into various stages, including **preconditions**, **mix motions**, and **postconditions**. Each of these stages can contain transients. The mix motions are categorised into **circular**, **ellipse**, and **orbital** movements.

The action designator must describe the mixing process's **initial conditions**, **intermediate transitions**, and **outcomes** to incorporate transients. For example, a ”spiral outwards” motion leading into the main mixing phase represents a transitional state where the robot approaches the final mixing pattern. This phase's transient could be a partial combination of ingredients that requires further mixing to achieve the desired consistency. To further integrate transients into PyCRAM for real-world applications, it's essential to establish a robust method for coding these transitions into the framework. This enhancement involves refining the action designators to include detailed descriptions of task stages, outlining how transients are initiated, evolve, and conclude.

For example, consider the mixing task, which can be divided into several key stages or transient states:
Initial Mixing: Starts the mixing at a slow speed to blend ingredients without spillage.
Main Mixing: Increases speed to ensure thorough batter mixing.
Consistency Checking: Reduces speed and checks if the batter has achieved the desired consistency.
Finalizing Mix: Completes the mixing process and prepares to conclude the task.

The concept centres around the introduction of the TransientState class with Conditions, as depicted in Listing 1:

Listing 1: Transient Python Class

```
class TransientState:
#Example call {TransientState(”Consistency Checking”, [], [],
   \{speed: ”low”, duration: ”2 minutes”, check: ”visual”\})
    def __init__(self, name, entry_conditions, exit_conditions,
        process_info):
        self.name = name
        self.entry_conditions = entry_conditions
        # Conditions to enter this state
        self.exit_conditions = exit_conditions
        # Conditions to leave this state
        self.process_info = process_info
        # Information like duration, influence factors

        [...]
```

This object-oriented strategy utilizes the TransientState class to manage various task stages, with each state capturing essential details such as duration and external influences. The ActionDesignator class, illustrated in Listing 2, has been expanded from its initial design to direct the sequence of states required to complete the entire task:

Listing 2: Action Designator

```
class ActionDesignator:
    def __init__(self, task_name, states: List[TransientState],
        [...]):
        self.task_name = task_name
        self.states = states
        # List of states representing the transients
        [...]
    def perform_task(self):
        for the state in self.states:
            self.enter_state(state)
            self.execute_state(state)
            self.exit_state(state)
            [...]
     [...]
```

The workflow in PyCRAM involves iterating over each state defined in the `states` list and managing the lifecycle of each state using three methods: `enter_state`, `execute_state`, and `exit_state`, with the following functionalities:

- `enter_state(state)`: Prepares the system to enter a given state, potentially verifying and establishing entry conditions.
- `execute_state(state)`: Manages the actual operations defined for the state, such as directing robotic actions, monitoring real-time data, and adjusting parameters based on `process_info`, within the TransientState class.
- `exit_state(state)`: Concludes the state's operations, ensures all exit conditions are met, and readies the system for the transition to the next stage or task completion.

7. Conclusion and Future Work

This work highlights the critical role of modelling transients in meal preparation tasks, using the example of pancake making to elucidate the transient states between task start and completion. By defining the logic of transients and proposing methods for their integration into robotic reasoning through PyCRAM, we aim to enhance robotic performance in complex, sequential task execution. We believe that the consideration of transients in the modelling of meal preparation tasks is crucial for agents that are able to reason about objects, object states, and failures during task execution. Including transients provides robots with a subtle understanding of task processes, allowing for more adaptive and subtle responses to dynamic task conditions. This is particularly important in tasks involving state or composition transformations, such as cooking, where intermediate states significantly influence the final outcome.

Future work will involve refining the integration of transients into meal preparation ontologies and enhancing the robotic action designators to more effectively incorporate and manage these transient states.

The potential for improving robotic interaction with dynamic environments through a better understanding of transients presents an exciting frontier for cognitive robotics and practical applications in everyday life.

References

[1] G. Kazhoyan, S. Stelter, F. K. Kenfack, S. Koralewski, M. Beetz, The robot household marathon experiment, in: 2021 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2021, pp. 9382–9388.

[2] M. Kümpel, V. Hassouna, J.-P. Töberg, P. Cimiano, M. Beetz, Cut, chop, slice or dice: Parameterising general action plans using knowledge graphs, in: International Conference on Intelligent Robots and Systems (IROS 2024), 2024. Submitted.

[3] M. Kümpel, Actionable Knowledge Graphs - How Daily Activity Applications can Benefit from Embodied Web Knowledge, Ph.D. thesis, University of Bremen, 2024. doi:10.26092/elib/2936.

[4] M. Kümpel, J.-P. Töberg, V. Hassouna, P. Cimiano, M. Beetz, Towards a knowledge engineering methodology for flexible robot manipulation in everyday tasks, in: Workshop on Actionable Knowledge Representation and Reasoning for Robots (AKR3) at European Semantic Web Conference (ESWC), 2024.

[5] D. Beßler, R. Porzel, M. Pomarlan, A. Vyas, S. Höffner, M. Beetz, R. Malaka, J. Bateman, Foundations of the socio-physical model of activities (soma) for autonomous robotic agents, arXiv preprint arXiv:2011.11972 (2020).

[6] K. Dhanabalachandran, V. Hassouna, M. M. Hedblom, M. Küempel, N. Leusmann, M. Beetz, Cutting events: Towards autonomous plan adaption by robotic agents through image-schematic event segmentation, in: Proceedings of the 11th Knowledge Capture Conference, 2021, pp. 25–32.

[7] M. M. Hedblom, M. Pomarlan, R. Porzel, R. Malaka, M. Beetz, Dynamic action selection using image schema-based reasoning for robots, in: Joint Ontology Workshops, 2021. URL: https://api.semanticscholar.org/CorpusID:240005320.

[8] M. Beetz, U. Klank, I. Kresse, A. Maldonado, L. Mösenlechner, D. Pangercic, T. Rühr, M. Tenorth, Robotic roommates making pancakes, in: 2011 11th IEEE-RAS International Conference on Humanoid Robots, IEEE, 2011, pp. 529–536.

[9] D. Danno, S. Hauser, F. Iida, Robotic cooking through pose extraction from human natural cooking using openpose, in: International Conference on Intelligent Autonomous Systems, Springer, 2021, pp. 288–298.

[10] A. Gangemi, N. Guarino, C. Masolo, A. Oltramari, L. Schneider, Sweetening ontologies with dolce, in: International conference on knowledge engineering and knowledge management, Springer, 2002, pp. 166–181.

[11] V. Mascardi, V. Cordì, P. Rosso, et al., A comparison of upper ontologies., in: Woa, volume 2007, 2007, pp. 55–64.

[12] D. Dooley, M. Weber, L. Ibanescu, M. Lange, L. Chan, L. Soldatova, C. Yang, R. Warren, C. Shimizu, H. K. McGinty, et al., Food process ontology requirements, Semantic Web (2022) 1–32.

[13] J. Winkler, M. Tenorth, A. K. Bozcuoglu, M. Beetz, Cramm–memories for robots performing everyday manipulation activities, Advances in Cognitive Systems 3 (2014) 47–66. URL: https://www.semanticscholar.org/paper/CRAMm-Memories-for-Robots-Performing-Everyday-Winkler-Tenorth/9c8c2448a033da67f6bea2a4d89fe5c77cfb3b00.

[14] M. Beetz, D. Beßler, A. Haidu, M. Pomarlan, A. K. Bozcuoglu, G. Bartels, Knowrob 2.0 – a 2nd generation knowledge processing framework for cognition-enabled robotic agents, in: International Conference on Robotics and Automation (ICRA), Brisbane, Australia, 2018. URL: https://ai.uni-bremen.de/papers/beetz18knowrob.pdf.

[15] S. Koralewski, G. Kazhoyan, M. Beetz, Self-specialization of general robot plans based on experience, IEEE Robotics and Automation Letters 4 (2019) 3766–3773. doi:10.1109/LRA.2019.2928771.

[16] G. Kazhoyan, A. Hawkin, S. Koralewski, A. Haidu, M. Beetz, Learning motion parameterizations of mobile pick and place actions from observing humans in virtual environments, in: 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2020, pp. 9736–9743. doi:10.1109/IROS45743.2020.9341458.

[17] V. Hassouna, H. Alina, M. Beetz, Steps towards generalized manipulation action plans - tackling mixing task, in: Workshop on Actionable Knowledge Representation and Reasoning for Robots (AKR3) at European Semantic Web Conference (ESWC), 2024.

[18] M. Beetz, G. Kazhoyan, D. Vernon, The cram cognitive architecture for robot manipulation in everyday activities, 2023. URL: https://arxiv.org/pdf/2304.14119.pdf. arXiv:2304.14119.