HanKA: Enriched Knowledge Used by an Adaptive Cooking Assistant

Nils Neumann(✉) and Sven Wachsmuth

Institute for Cognition and Robotics (CoR-Lab), Universitätsstr. 25, 33615 Bielefeld, Germany
{nneumann,swachsmu}@techfak.uni-bielefeld.de

Abstract. Cooking a meal is a challenging and recurring task that requires the consideration of various environmental influences and constraints as well as significant domain knowledge. One way to reduce the complexity is to follow recipes that provide an ordered set of tasks for the preparation of a dish. This concept was already transferred to cooking assistants which present the recipe to the cook while adding further assistance like device control or step-by-step visualization. Although recipes and assistants simplify the cooking process itself, other factors like the available devices or differences in cooking skills are ignored. Aside from that, current assistants are often limited to one recipe at a time, ignoring the regularly occurring requirement to prepare a meal of multiple components. Considering these challenges, we propose our adaptive cooking assistant *HanKA* that considers the individual user skill and environmental kitchen setup, while assisting the cook at the preparation of their freely combined and synchronized recipes. This is achieved through a modular approach consisting of the automatic detection and control of the available devices and user interfaces, the scheduling of multiple recipes based on the distributed knowledge representation, and a deviation management that considers the user experience. Hereby, we created an adaptive cooking assistant that considers various influences that occur in a cooking scenario, resulting in a better assistance for the user.

Keywords: AI applications and innovations · Knowledge representation and reasoning · Planning and scheduling · Human monitoring · Assisted living · Intelligent assistive environments

1 Introduction

Cooking is a complex instrumental activity of daily living [9], whereby cooking skills correlate with healthy eating [7]. Hereby, a layperson has to continuously transfer and coordinate recipe instructions in his individual cooking environment. Due to an increasing digitization of life, multiple cooking assistants were developed that assist a cook in the preparation of a dish. *Cooking Navi* [6] focuses on the synchronization of cooking recipes, *PIC2DISH* [3] generates the cooking instructions from a picture and *MimiCook* [16] projects the instructions on the kitchen counter. While these provide a step-by-step assistance for the cook, they do not provide key characteristics of user assistance systems [11] like context-awareness and adaptation to the assisted person. In the cooking domain, this includes the detection, control, and consideration of the available devices. Although recipes provide step-by-step instructions for the cook, and in some cases customization features like exchanging ingredients [1,2,12], these are often limited to a single component. A completely flexible composition of multiple recipes to an assisted cooking process including an automated sequencing, distributed device control, and dynamic synchronization of preparation steps has not been realized so far. Time estimations in traditional recipes do not consider the skills of the cook and frequently differ greatly from the time actually needed. Similar to *MAMPF* [15], we individually measure and adapt the duration time for each step. Beyond this, we further generalize these time estimates to new recipes and ingredients. Although various assistive systems for the kitchen exist that all successfully assist the cook when preparing dishes, most of these focus on a step-by-step instruction and simplify contextual factors, as shown in Table 1. However, the consideration of these contextual factors would lead to an individualized *in situ* assistance of the user in his regular kitchen environment as an ultimate goal.

Table 1. Functionality provided by cooking assistants: Asterisk (*) means only build-in devices are controlled. The first 5 are research prototypes, the next 2 are available products, followed by our *HanKA* assistant. "Skill adaptation" means that the cooking skills of the user are considered while planning, and does not describe if the system adapts while cooking.

Cooking assistance	Device		Recipe		Skill adaptation
	Control	Detection	Combination	Synchronization	
MimiCook [16]	-	-	-	-	-
CookingNavi [6]	-	-	✓	✓	-
PIC2DISH [3]	-	-	-	-	-
KogniChef [13]	✓	-	-	-	-
MAMPF [15]	✓	-	✓	-	✓
Thermomix [17]	*	-	-	-	-
Cookit [4]	*	-	-	-	-
HanKA	✓	✓	✓	✓	✓

In this paper we present our cooking assistant *HanKA* (German acronym for: action centered coordination of assistance processes) that provides the following contributions: (i) automatic detection and control of available user devices; (ii) progress detection utilizing device interactions; (iii) combination and synchronization of freely combinable recipes; (iv) adaptation of recipes to available devices and cooking skills; (v) utilizing a distributed knowledge base for enriching minimal recipes with general cooking knowledge; (vi) a working cooking assistant that guides the user, monitors the progress, and adjusts to occurring

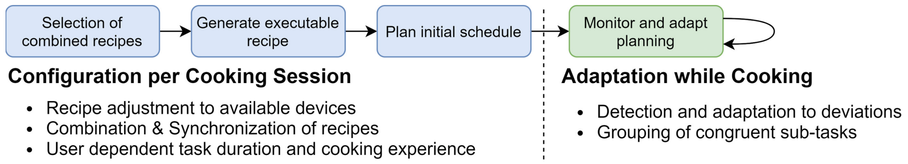
Selection of combined recipes
Generate executable recipe
Plan initial schedule
Monitor and adapt planning
Configuration per Cooking Session
Recipe adjustment to available devices
Combination & Synchronization of recipes
User dependent task duration and cooking experience
Adaptation while Cooking
Detection and adaptation to deviations
Grouping of congruent sub-tasks

Fig. 1. Visualization of the phases from a cooking session and the assignment of the procession loops with their related adaptations.

deviations. In this way, the cook can concentrate on the cooking task itself and entrusts the scheduling of the tasks and managing of the devices to the assistant.

2 Methodology

In order to create a cooking assistant that provides the previously described flexibility, the provision of multiple adaptations is required (Fig. 1). These adaptations are organized in two different processing loops. The outer loop initiates a configuration at the start of each cooking session and the inner loop repeatedly applies a process adaptation. Before the adaptation concepts are described in detail, a brief summery of the underlying knowledge representation is given.

2.1 Underlying Knowledge Representation

In the following, the concepts of the knowledge representation are briefly described. More details can be found in [14]. The distributed knowledge representation consists of three sources (component recipes, ingredients, *action_ templates*) which, when combined, result in the executable recipes for the cooking assistant. These three knowledge sources are linked by the action type of each task in the component recipe (Fig. 2). The hierarchical representations of these knowledge sources allow to reuse and generalize knowledge that is utilized in the proposed concepts. *Action_ templates* are abstract actions that are parameterized by the other knowledge sources. They provide all actions executable by the cooking assistant with generalized and action-dependent information: e.g., whether a user and/or device are required for the execution, the urgency to proceed with the following action (connection urgency) and all subordinate (pre/post) actions. The recipe representation contains a set of tasks with a reference to the corresponding *action_ template*, the logical dependency between the tasks and all recipe specific information, like ingredients and amount per serving. The ingredient representation for each usable ingredient contains actions executable with it, together with their general parameters (e.g. cooking time). If an information is not defined in the recipe, the missing information is added from the ingredient representation or, if not available for the ingredient, from the *action_ template.*

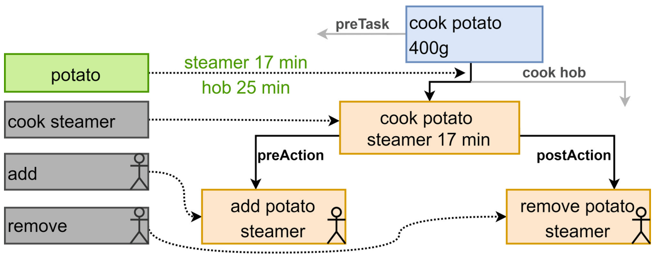

Fig. 2. Exemplary creation of executable recipe steps (orange) for the single recipe representation step *cook potato* (blue), while leaving out the connection to previous tasks or the alternative preparation with a hob. The recipe task *cook potato* is enriched with information (cooking times) from the cook action of the ingredient representation *potato* (green). These combined information parametrize the abstract *action_ template cook steamer* to create the executable *cook potato steamer* recipe step (orange). Since the *action_ template cook steamer* has (sub-)tasks, these executable steps are also generated with information from the recipe and *action_ templates*, visualized by stick figure. (Color figure online)

2.2 Configurations per Cooking Session

The configuration per cooking session mostly takes place while creating the executable representation of the recipes that are selected by the cook. This provides the necessary infrastructure to define the inference processes for any configuration or adaptation of the executable recipe proposed in this paper. This representation contains all information about the available devices and cook as well as all recipes that should be cooked together.

Recipe Adjustment to Available Devices. In order to deal with varying devices in different kitchens, available devices must be discovered and device-specific programs have to be abstracted to functionalities used by the cooking assistant. Describing the required device-independent functionality in the *action_ templates* enables the mapping of recipes to devices and the filtering of executable recipes in the context of the available devices. Through this, only recipes with executable tasks on the available devices are planned and executed. Adapting to the devices and utilizing the abstracted device program description from the *action_ templates*, enables the automatic adjustment to different setups and allows automated device control in the recipe context.

Combination and Synchronization of Recipes. Enabling the user to freely combine different recipes requires a flexible, automated enriching and scheduling of instruction steps. In this case, further challenges arise, such as, where recipes can be interrupted, which tasks are connected time-critically, which tasks can be parallelized, which tasks require specific devices or the user attention, and how are all component preparations synchronized to be served hot. Therefor, the self descriptive recipe contains all information about the recipe steps that are utilized by the scheduler to arrange the tasks into a temporal order. In order to plan the recipes based on the current setup, the scheduler needs the available devices and information about the cook in addition to the recipes. Based on the cook and device workload for each task, the scheduler can take the occupancy into account. This enables the planner to execute tasks in parallel in regard to the resources, which made it possible to synchronize the recipes. Utilizing the logical connection between tasks and their connection urgency from the recipes, enables the scheduler to know which tasks have to be carried out one after the other and gives an indication where recipes can be interrupted. When alternative methods of preparation are described by a recipe, further options are available for the scheduler. The approach described so far leads to a valid plan, but does not take the synchronization of the serving times into account. Because device tasks that heat an ingredient are critical for serving everything hot, the last one of these for each recipe must end as late as possible for all component recipes.

User Dependent Task Duration and Cooking Experience. The duration of many preparation tasks greatly varies for different cooks. A better prediction will result in less deviations, less re-planning, and a better estimation of the serving time. While storing the last time for a recipe task would be sufficient, this only provides a prediction for the step in the specific recipe. Using a more generalized approach that does not just refer to the recipe but also to general cooking skills, enables the estimation of preparation time for recipes yet not cooked by the user. A solution for this is presented by estimating factors that scale the default task duration appropriately. Such factors can be estimated on three different levels of abstraction utilizing the distributed knowledge representation of recipes, ingredients, and actions. After each cooking session, these three factors are updated for each executed step, as further explained in Sect. 3.3. While some cooking actions scale with improved cooking skills of the user like peeling, others only depend on the recipe task for actions like cooling. Therefore, it is necessary to know which tasks scale with the cooking skill and which not. Since this information is action-dependent, it is provided by the *action_ templates*, which define if an action can adapt to the user. As the duration for some tasks scale with the number of servings and some have fixed values, these information are also provided by the *action_ template*. By saving the default user duration of the task as a factor, this value is independent of the number of servings and calculated when these are selected.

2.3 Adaptation While Cooking

Based on the initial plan with a complete schedule of tasks, the user starts the cooking process. Although the plan contains only tasks executable with the given devices and uses task durations adapted to the user, this does not ensure that no further adaptations are required. Therefore, a monitoring and re-planning of the cooking process is implemented.

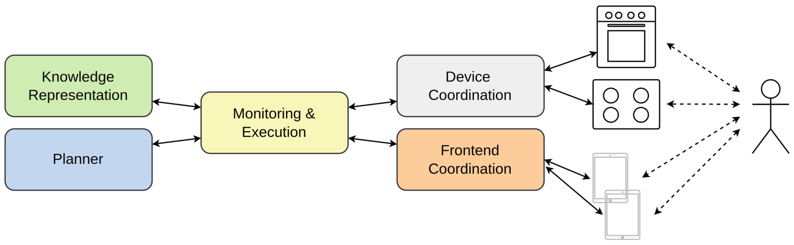

Fig. 3. Component overview of the cooking assistant: The user interacts with the devices and user interfaces which are connected to the corresponding coordination components (3.1, 3.2). These send the interaction commands to the *Monitoring and Execution* (3.4) component, that executes the recipe and detects deviations based on the scheduled task order from the *Planner* (3.5), which is calculated on the enriched knowledge representation from the *Knowledge Representation* (3.3).

Detection and Adaptation to Deviations from the Plan. While cooking a meal, the estimated duration of the tasks can differ from the real duration. Possible reasons are varying attention levels, external distractions or delays from devices. Examples are the missing confirmation of the device execution or a prolonged heating process with larger amounts of water. Therefore, it is necessary to monitor the cooking session and react to occurring deviations. These deviations can be that a task is not executable because a previous task is not finished, or that a running task should have ended but the user or a device has never confirmed it. If a deviation is detected, the discrepancy to the old plan is corrected by re-scheduling critical tasks. By adapting to the current situation, the cook receives a permanently executable plan and is supported until the dish is finished.

Grouping of Congruent Sub-tasks for Compact User Instructions. While experienced cooks profit from compact instructions, beginners require detailed simple cooking instructions. Therefore, we propose an automated strategy to merge congruent cooking instructions. Here, the frequency how often a user cooked the recipe and the appropriate ingredient-action pair beforehand is used to determine if tasks are combined. As the final task sequence is a result of the planning process, the grouping of tasks is considered after planning. Before executing a task, the following tasks in time are inspected if they are merge-able with the current task and if the user has enough experience for the combination. Whether tasks can be combined is decided in regard to the used ingredient, logical connection, action type and device usage. If a combination of tasks is possible, the description, title, and images of the tasks are merged, and both tasks are started. Clustering the task shortly before the execution retains that scheduling can find the best possible solution, while also providing compact user instructions in an adaptable manner.

3 System Description

Creating an adaptive cooking assistant covers a wide set of areas, resulting in thematically appropriate components roughly following *MAPE-K* from autonomic computing [8], as shown in Fig. 3. These components are connected with a customized request/response pattern, based on the *MQTT* protocol [10] and communicate with serialized protocol buffer messages. This results in a well-defined API and clear task areas for each component and allows the decoupling of components. In the following subsections, the individual components, their contribution to the overall system, and their adaptation strategies are described.

3.1 Device Coordination

The *Device Coordination* discovers and provides the available devices, controls them, and detects process events required in the active device tasks from the recipe. To keep it flexible, the component abstracts device-specific information into a device-independent description that is used to describe device tasks inside a recipe through which a mapping between functionalities in the recipe and device specific programs is enabled. In addition to information such as operating mode (e.g. two-sided heat), temperature, and duration, the description contains the general type of actions (*preheat*, *execute*, *add*, *remove*). This is utilized by the individual device controller which interprets the sensor data in the recipe context and tracks the task progress. Through the additional information the device functionality is extended and allows a context-aware interpretation of built-in device sensors. Whereby preheat is finished if the temperature is reached, execute is finished if the required duration is expired and add/remove are finished if other device sensors detect the addition or removal of an ingredient at the time of the task (e.g. door switch sensor for an oven). If the end of a task is detected by the device, it notifies the *Monitoring and Execution*, which results in automatic recipe progress.

3.2 Frontend Coordination

The *Frontend Coordination* is responsible for the coordination and synchronization of the user interfaces and for the generalization of the user input. While the specific user interfaces are not known beforehand for each setup analogous to the available devices, they register via a discovery service to the *Frontend Coordination*. The registered interfaces are connected with and synchronized by the *Frontend Coordination*. Hereby, they implement a defined API to interact with the system that abstracts the user input to commands, relevant for the cooking process. Through this, the assistant can connect to different user interfaces (tablet, speech interface or several of these) available in the concrete setup (e.g. Fig. 4). Via the user interfaces the cook can combine recipes into the desired dish, start and stop the cooking process, add or remove recipes to an active cooking session, and visualize the planned task order. During cooking, the cook is always able to finish any running task, select any valid new task, as well as to preview tasks before they are active.

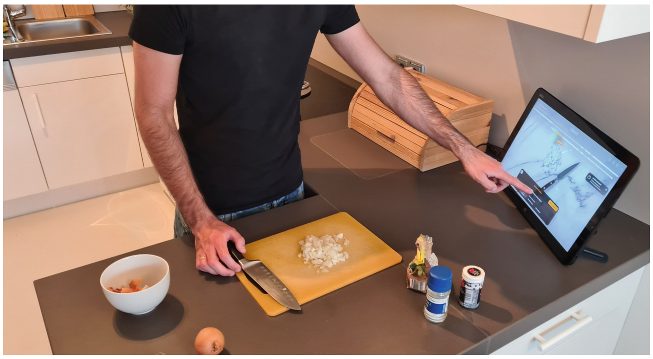

Fig. 4. Exemplary cooking scenario, where the user confirms the cooking step *cut onion* via a tablet.

3.3 Knowledge Representation

The *Knowledge Representation* provides the executable and combinable recipes by utilizing the distributed knowledge sources (recipe representation, ingredient representation and *action_ templates*), as summarized in Sect. 2.1. The enriched recipes provide the required information for scheduling (*Planner*), visualization (*Frontend Coordination*), device controlling (*Device Controller*) and monitoring (*Monitoring and Execution*). As the recipe should be cookable in the individual kitchen, the created executable recipe representation also contains the registered devices with their functionalities from the *Device Coordination* and the registered cook from the *Frontend Coordination*. Based on the available devices, non executable recipes are removed. If an unknown user starts cooking with the assistant, the system creates a user-specific ontology that stores the cooking experience and duration factors for each recipe step in reference to the recipe, ingredient, and action. Due to the different levels of abstraction and the more frequent use of more generalized actions, a weight factor X is added in regard to the referred knowledge source. In the ontology, the cooking skill of the user is saved regarding the recipe step (R *($X = 5$)*), the ingredient-action pair (I *($X = 10$)*), and the action type itself (A *($X = 20$)*). After each cooking session, these three factors (R,I,A) are updated for each executed step (Eq. 1), with the corresponding *X-value* for R,I,A.

$$newFactor(R, I, A) = \frac{X * oldFactor(R, I, A) + \frac{measuredTime}{defaultTime}}{X + 1} \quad (1)$$

$$userTaskDuration = \frac{R + I + A}{3} * taskDuration \quad (2)$$

When a new recipe is generated, the duration for each task that the user can perform is calculated by Eq. 2, with a factor of 1 if no value was saved before. Even though the R value has no impact for recipes the user has never cooked, the I and A values are recipe independent, allowing a prediction for recipes that are unknown to the cook. Through the recipe representation, *Monitoring and Execution* also receives information how often the cook executed certain recipes and utilized ingredients in order to merge tasks into more compact instructions.

3.4 Monitoring and Execution

The *Monitoring and Execution* component connects all components of the cooking assistant. However, the main task is the monitoring of the cooking process and the execution of the active tasks. Therefore, the execution of tasks and the detection of deviations is split into multiple conditions and actions that are repeated in an adaptation loop. While the *Monitoring and Execution* component detects the deviations, the adjustment of the task schedule is carried out by the *Planner* component.

- **Task removal:** Checks if the *Device* or *Frontend Coordination* finished a task and removes it from the active tasks.
- **Active task verification:** Checks if an active task should have been ended but is still running. If a task is still running with an end time in the past it is extended.
- **Task start:** If the time for a new task to run has been reached, it is verified that there are no resource conflicts resulting from an extended step. It is further checked whether all logically connected previous tasks are finished. When all tasks are finished and no resource conflicts are found, the task is added to the active tasks. If not, a re-planning is necessary due to a deviation.
- **Execution triggering:** If the list of active tasks changes, they are send to the *Device-* and *Frontend Coordination* in regard to the required resources. If the cook has experience with the recipe or ingredient, it is checked if the task can be combined with the following tasks, based on the ingredient, task connection and action type of the tasks. If detected they, are combined and communicated as one task with merged content information.
- **Planning:** If any of the previous steps detects a deviation, a planning request with the actual state, available devices and cook is send to the *Planner*. The solved response is used as the new plan for execution.

While the initial combination and synchronization of recipes is performed beforehand, the combination of recipes can be changed while cooking, e.g. removing, replacing or adding a component recipe. In this case, the appropriate action is executed and the new recipe combination in regard to the progress of the already running recipes is synchronized. This enables an even greater adaptability of the cooking process for the cook.

3.5 Planner

The scheduling is considered as an optimization problem, where the domain, rules and moves are defined and implemented in a way that the optaplanner [5] engine can be utilized to combine and synchronize the recipes with the available

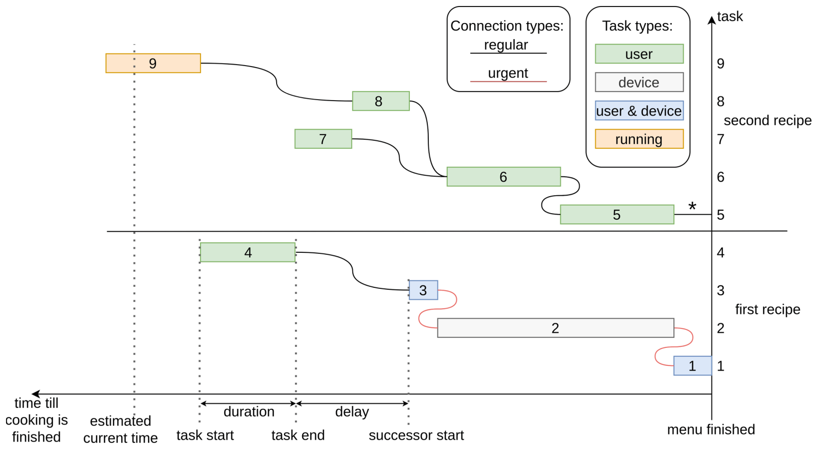

Fig. 5. Simplified planning structure with two recipes and nine tasks: The end time of a task is the earliest successor start time together with a variable delay, changeable by the planner. The task 4 has a delay to his successor task (3) as the user attention is required for tasks 7/8. In contrast, the user tasks 5/6 are parallel to the device task 2, because it only needs a device and no user. The urgent connection between 1/2 and 2/3 urges the planner to prevent a delay. For the synchronization of the recipes, the planner minimizes the delay (*). Since it is a planning request in a running state, task 9 is executed by the user at the moment. Therefore, if the task is moved, the time till cooking is finished also changes, resulting in a changed serving time.

devices. Hereby, each planning request is stateless and utilizes the cooking state of the request, which can only have small deviations for re-planning requests. The main planning entity of the domain is the task. Each task is assigned to a recipe. Since the planner synchronizes the recipes, all recipes have to be finished around the same time. Therefore, the synchronization point of our planning problem is the serving time, with the serving time defining the origin and the time axis representing the time till the cooking process is finished, as shown in Fig. 5. Utilizing this time representation, the first two optimizing criteria are the difference between the finished menu and the actual time (cook as fast as possible) and the distance from the last recipe step to the finished menu (synchronized serving). The logical dependency of the tasks is modeled as a built-in constraint, because any deviation from the logical order results in an invalid plan. Therefore, the tasks are connected as a chain where the end of a task is the earliest start time for all its successor tasks. This includes a delay that is used as a planning variable, allowing the planner to change the time between tasks. Using the time representation with the chain connection creates the advantage that if a deviation occurs in the cooking process, only the ends of the chain have to be adjusted, resulting in an easy re-planning. Solving the logical dependency with the built-in constraint, the main constraint remaining is the consideration of the workload for the user and devices. These are assigned based on the functionalities required by the task. In order to not exceed the maximum workload for user and devices,

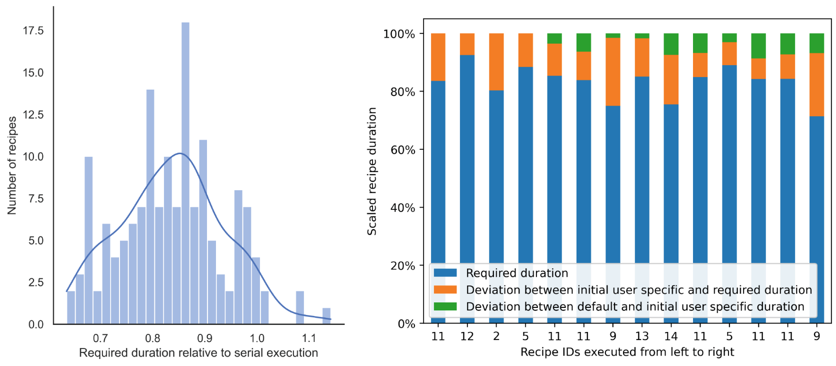

Fig. 6. Left: Distribution of the relative duration for the 153 combinations in regard to their serialized execution duration. Right: User Adaptation over multiple cooking sessions. The single recipes are executed by a new simulated user that validates each user task after 70% of the required duration defined in the base recipe. The device tasks are executed normally. The recipes, with their IDs were executed from left to right. Hereby, 100% is the necessary duration without user adaptation and the blue bar is the required duration the simulated user actually needed. As the system adjusts to the user skills, the deviation between initial plan and required duration (orange) reduces, while the deviation between the initial plan and the required duration without user adaptation increases (green). This happens especially strong if a known recipe is repeated (ID 11) due to the different factor (Sect. 3.3). (Color figure online)

their workload is calculated over all tasks and if the maximum value is exceeded, a hard constraint is detected which results in a penalty that outranks all other scores. In order to ensure that action sequences requiring the same device, e.g., adding multiple ingredients, are still consistently assigned, device groups are formed that treat them as a single compound task. For the creation of a plan that is acceptable for the user, the logical dependency/cohesion is considered as soft constraint. This takes the urgency of the connection into account where delays between tasks with a high urgency result in higher penalties. Nevertheless, tasks with a low urgency connection are still sticked together if possible. Since the planner should not always move all tasks in the chain, a custom move was implemented that moves a single task in the range between the next task and the following task, which expands the possibilities of the planner. Based on the described domain, rules, and scores, the tasks are ordered, deviations are solved, and an executable plan is returned.

4 Evaluation

The flexibility of the *HanKA* cooking assistant is provided by multiple adaptations. Therefore, they are evaluated individually as far as that is possible in a quantitative evaluation. Unless otherwise stated, a kitchen setup consisting of a steamer, oven and hob is used. The evaluation is based on 18 recipes, each consists of 6 to 64 tasks (avg. 19.6). Hereby, 1/3 of the recipes has two options of preparation, e.g. cooking noodles with a hob or steamer. The initial scheduling of a single recipe requires 2474 ms on average. The average planned execution time for a recipe with 4 servings is 46 min. and 39 s. and 38 min. and 15 s. for 1 serving. Resulting in a duration increase of 22.09% on average between 1 and 4 servings, not evenly distributed over the recipes, due to differences in task scaling (min. cooked noodle 0%/max. cabbage turnip 94%). On average the recipes have an idle time of 40.98% (4 serv.) that describes, how much of the time the cook does not perform a task. This ranges from 0% for vegan mayonnaise to 81.69% for burger buns with dough resting time. The idle time indicates optimization possibilities when combining recipes and thus is a valuable scoring criteria to evaluate the recipe combinations. For the evaluation of the recipe combination, all 153 2-pair combinations of the 18 available recipes were calculated with an average initial planning time of 3394 ms. Hereby, the planned cooking duration is reduced to 83.68% of the time that is required to cook these successively, as shown in Fig. 6 (left) for all combinations. The longer duration in 6/153 cases is noticeable and results from changing types of preparation, that take longer but provide a better synchronization at the end. For a single component, the time between heating the last heating task and serving was 129.44 s, while for the combined recipes the average for each component was 229.64 s. In this duration, the user removes the components from the devices and performs the last tasks that are only executable afterwards. The lower idle time of 25.52% for two recipes in contrast to 40.98% for a single recipe indicates a useful combination of the recipes.

The cooking skills are evaluated with a simulated user, visualized in Fig. 6 (right). This user operates the assistant including devices just as a normal user, but takes over the user interface due to the well-defined API. Hereby, the initially estimated duration approaches the required duration over multiple cooking sessions, especially if the recipe was cooked before, but also for completely unknown recipes (e.g. ID 9). Due to varying amounts of user tasks, the possible adaptation varies depending on the recipe. While executing the recipes, the average adaptation time to deviations was 908.27 ms, which happened 24.14 times per cooking process on average.

5 Conclusion

While supporting a cook at the preparation of a meal, cooking assistants and recipes focus mostly on the cooking process itself. Although the cooking process is the most important part, ignoring various other influences prevents the exploitation of the full potential. Therefore, we identified these influences, presented approaches that take these into account, and combined them into our adaptive cooking assistant *HanKA*.

As a result, the cooking assistant combines and synchronizes multiple recipes, and considers the available devices while also controlling these in the context of the task. It considers the cooking skills of the user, adjusts the granularity of steps based on the user experience, and adapts the cooking process in case of deviations. Due to the adaptation of the system, the workload for the cook is reduced whereby the user no longer has to check and control the available devices or merge the recipes by hand. Through this, the cook can focus on the cooking process itself.

References

1. Innit - Your food. Simplified and solved (2021). https://www.innit.com/. Accessed 02 Nov 2021
2. Plantjammer: Dynamic recipes (2021). https://www.plantjammer.com/dynamic-recipes. Accessed 02 Nov 2021
3. An, Y., et al.: PIC2DISH: a customized cooking assistant system. In: Proceedings of the 25th ACM International Conference on Multimedia, pp. 1269–1273. MM 2017, Association for Computing Machinery, NY (2017). https://doi.org/10.1145/3123266.3126490
4. Bosch: Cookit - küchenmaschine mit kochfunktion - bosch hausgeräte (2021). https://www.bosch-home.at/shop/kuechenmaschine-mit-kochfunktion1. Accessed 02 Nov 2021
5. De Smet, G., Open Source Contributors: OptaPlanner User Guide. Red Hat, Inc. or third-party contributors (optaPlanner is an open source constraint solver in Java) (2006). https://www.optaplanner.org
6. Hamada, R., Okabe, J., Ide, I., Satoh, S., Sakai, S., Tanaka, H.: Cooking navi: assistant for daily cooking in kitchen. In: Proceedings of the 13th Annual ACM International Conference on Multimedia, pp. 371–374. MULTIMEDIA 2005, Association for Computing Machinery, NY (2005). https://doi.org/10.1145/1101149.1101228
7. Hartmann, C., Dohle, S., Siegrist, M.: Importance of cooking skills for balanced food choices. Appetite **65**, 125–131 (2013). https://doi.org/10.1016/j.appet.2013.01.016
8. Kephart, J., Chess, D.: The vision of autonomic computing. Computer **36**(1), 41–50 (2003). https://doi.org/10.1109/MC.2003.1160055
9. Lawton, M.P., Brody, E.M.: Assessment of older people: self-maintaining and instrumental activities of daily living. Gerontologist **9**(3_Part_1), 179–186 (1969)
10. Light, R.A.: Mosquitto: server and client implementation of the MQTT protocol. J. Open Source Softw. **2**(13), 265 (2017). https://doi.org/10.21105/joss.00265 https://doi.org/10.21105/joss.00265 https://doi.org/10.21105/joss.00265
11. Maedche, A., Morana, S., Schacht, S., Werth, D., Krumeich, J.: Advanced user assistance systems. Bus. Inf. Syst. Eng. **58**(5), 367–370 (2016). https://doi.org/10.1007/s12599-016-0444-2
12. Müller, G., Bergmann, R.: CookingCAKE: a framework for the adaptation of cooking recipes represented as workflows. In: ICCBR (Workshops), pp. 221–232 (2015)
13. Neumann, A., et al.: "KogniChef": a cognitive cooking assistant. Künstl. Intell. **31**(3), 273–281 (2017). https://doi.org/10.1007/s13218-017-0488-6
14. Neumann, N., Wachsmuth, S.: Recipe enrichment: knowledge required for a cooking assistant. In: Proceedings of the 13th International Conference on Agents and Artificial Intelligence - vol. 2: ICAART, pp. 822–829. INSTICC, SciTePress (2021). https://doi.org/10.5220/0010250908220829

15. Reichel, S., Muller, T., Stamm, O., Groh, F., Wiedersheim, B., Weber, M.: MAMPF: an intelligent cooking agent for zoneless stoves. In: 2011 Seventh International Conference on Intelligent Environments, pp. 171–178 (2011). https://doi.org/10.1109/IE.2011.18
16. Sato, A., Watanabe, K., Rekimoto, J.: MimiCook: a cooking assistant system with situated guidance, pp. 121–124 (2014). https://doi.org/10.1145/2540930.2540952
17. Thermomix: Thermomix® das original | vorwerk thermomix® (2021). https://www.vorwerk.com/de/de/c/home/produkte/thermomix. Accessed 02 Nov 2021