Modelling and Analysis of Self-Adaptive Systems using Petri Nets: A Robot Kitchen Case Study

Finn Wellershaus[1], Michael Köhler-Bußmeier[1] and Jan Sudeikat[1]

[1]Hamburg University of Applied Sciences, Berliner Tor 7, D-20099 Hamburg, Germany

Abstract

Current cyber-physical systems (CPS) consist of connected sub-systems. Recently, flexibility and adaptivity gain more and more attention in application areas, e.g. in the smart factory context. Multi-Agent Systems (MAS) are used to model CPS sub-systems and their interaction. Recent research focuses on the self-organization of these CPS-MAS. In MAS the concept of an organization (and consequently that of self-organization) is a research topic on its own. Within our research, we address the modeling and the analysis of these self-organizing MAS-Organizations. As part of this paper, we demonstrate the usefulness of formal modeling techniques (here: high-level Petri Nets) to obtain qualitative and quantitative insights.

Here, a robot kitchen is used as a metaphor for a flexible production scenario: We have recipes as production jobs, cooking robots with tools as interacting CPS, a conveyor belt as a transportation system for the dishes under preparation and so on. The scenario provides several sources of adaptivity: the robot cooks may decide to change the priority of jobs due to changing external signals (prices); the kitchen may run out of stock; a specialized robot may become a bottleneck and other robots change their tools to support it. Our model is implemented in RENEW, an interactive Petri net simulator that supports nets-within-nets, which is a formalism defined to support self-modification in a direct manner. The graphical model deepens the understanding of the application domain (the real world) among the project partners; the formal model allows for qualitative analysis, for example the absence of deadlocks, as well as quantitative aspects like finishing time and throughput.

Keywords

Self-Adaptation, MAPE-loop, Cyber Physical Systems, Multi-Agent Systems, Petri Nets

1. Introduction

Current cyber-physical systems (CPS) [1] consist of connected sub-systems. Recently, flexibility and adaptivity [2] gain more and more attention in application areas, for example in the smart factory context [3]. Multi-Agent Systems (MAS) [4] are used to model CPS sub-systems and their interaction. Recent research focuses on the self-organization of these CPS-MAS, usually supported by a MAPE-loop pattern [5]. In MAS the concept of an organization (and, consequently, that of self-organization) is a research topic on its own [6]. In our research we address the modeling [7] and the analysis [8] of these self-organizing MAS-Organizations. In this paper we demonstrate the usefulness of the proposed Petri net model to obtain qualitative and quantitative insights. It a probabilistic selection mechanism to execute transitions as part of the simulator.

Multi-robot systems and cyber-physical system applications consist of a multitude of separate objects. Dynamic environments challenge static systems and require dynamic changes during runtime. This can be done externally, or within the system itself. These self-adaptive systems provide a more organic and direct approach to dealing with systems in heterogeneous and dynamic environments.

Here, a robot kitchen is used as a metaphor for a flexible production scenario: We have recipes as production jobs, cooking robots with tools as interacting CPS, a conveyor belt as a transportation system for the dishes under preparation and so on. Robots act as simple actors within the system

PNSE'25, International Workshop on Petri Nets and Software Engineering, 2025

janne.wellershaus@haw-hamburg.de (F. Wellershaus); michael.koehler-bussmeier@haw-hamburg.de (M. Köhler-Bußmeier); jan.sudeikat@haw-hamburg.de (J. Sudeikat)

https://www.haw-hamburg.de/michael-koehler-bussmeier (M. Köhler-Bußmeier); https://www.haw-hamburg.de/jan-sudeikat (J. Sudeikat)

0009-0006-8729-038X (F. Wellershaus); 0000-0002-3074-4145 (M. Köhler-Bußmeier); 0009-0007-7871-9006 (J. Sudeikat)

and interact with their environment by creating dishes and passing intermediate products to other specialized robots.

This scenario is rather static and completely depends on the initial configuration. Here, we consider extensions to make it more dynamic and adaptive. The scenario provides several sources of adaptivity: the robot cooks may decide to change the priority of jobs due to changing external signals (prices); the kitchen may run out of stock; a specialized robot may become a bottleneck and other robots change their tools to support it. The robot kitchen case study can be used as environment to answer more general problems of flexible production scenarios. In this paper this is shown by integrating a MAPE loop to enable self-adaptivity. This shows how the model could be used to us with general insights about costs and benefits of adaptation for flexible production systems in general.

The primary goal is to make a Petri net based modeling system that can be extended to answer more general problems from flexible production scenarios. To achieve this, a generic robot-kitchen model is developed. This includes the necessary infrastructure to simulate that type of environment. By implementing a MAPE [5] loop, which is an established approach for self-adaptive systems, the applicability of this model to observe challenges and successes for production scenarios is shown. One expected outcome of this adaptivity is that it should improve the utilization of the available resources (robots) and improve delivery times. To achieve this, the basic model is extended and a simple MAPE loop is implemented and evaluated to what extent the adaptivity has a tangible effect on the system's performance. The loop monitors the utilization of each capability. If one exceeds predefined limits, it triggers a reconfiguration, which causes robots to change their tools to support another robot with a high demand capability.

Petri nets provide a modeling approach for distributed systems. Especially colored nets as well as nets-within-nets enable simulating complex systems. Our model is implemented in RENEW [9], an interactive Petri net simulator that supports nets-within-nets [10], which is a formalism defined to support self-modification in a direct manner. The graphical model deepens the understanding of the application domain (the real world) among the project partners; the formal model allows for qualitative analysis (like the absence of deadlocks) and for quantitative aspects like finishing time or throughput.

This paper is structured as follows: The Section 2 describes the structure of the basic model as well as some initial design decisions. In Section 3 the conceptional overview is given, and the extensions required for the MAPE loop, as well as the implemented components of the loop are described. Section 4 talks about the simulation, its results as well as some observations made during integrating the new components. The last Section 5 summarizes the findings and highlights some possible further areas of interest and possible extension points.

2. Simulation Scenario and Environment

The basic scenario is a group of robots that are used as kitchen robots to create food based on the recipes provided. The idea of using a kitchen as a scenario for multi-agent interaction is inspired by the bakery example in [11]. The robots are organized in a circle around a conveyor belt over which plates with the intermediate food are passed around and worked on.

2.1. The Net Model

Our robot kitchen is modelled using the simulator RENEW [9] in the version 4.1[1]. There are multiple independent nets that are connected via channels or passed as "nets within nets" [10]. The model makes use of Coloured Petri Nets (CPN) [12] as well as the timed extensions. The high-level features of RENEW allow to model a complex and large system more easily [13]. Using the timed net formalism, it

[1]RENEW is freely available from http://renew.de/. The sources of the model are available at https://github.com/finnWellers/publication-cps-petri-nets. Our Petri net model of this paper is based on a model initally develeopped by Helen Haase and Finn Wellershaus.

is possible to simulate the (temporal) performance and behaviour of a system, which is of interest for the model of a physical robot kitchen.

The base-simulation consists of $n = 6$ dedicated robot stations. One is only used for delivering the finished meals and another one is responsible for assembling dishes from different previously cooked steps. The others are interchangeable, but configured to provide different services. Plates with the orders, the intermediate prepared food and the recipe IDs are passed around using a conveyor belt. The position of all orders of a specific recipe are stored within a single recipe net instance. This was done to make potential global planning and analysis easier. Part of the conceptual idea is that not the actual recipes are passed around, just an information which recipe is used. All the stations and belt positions are explicitly modelled for easier readability (a simplified variant of the Petri net is given in Fig. 1). In the following, the important aspects of the model are described.

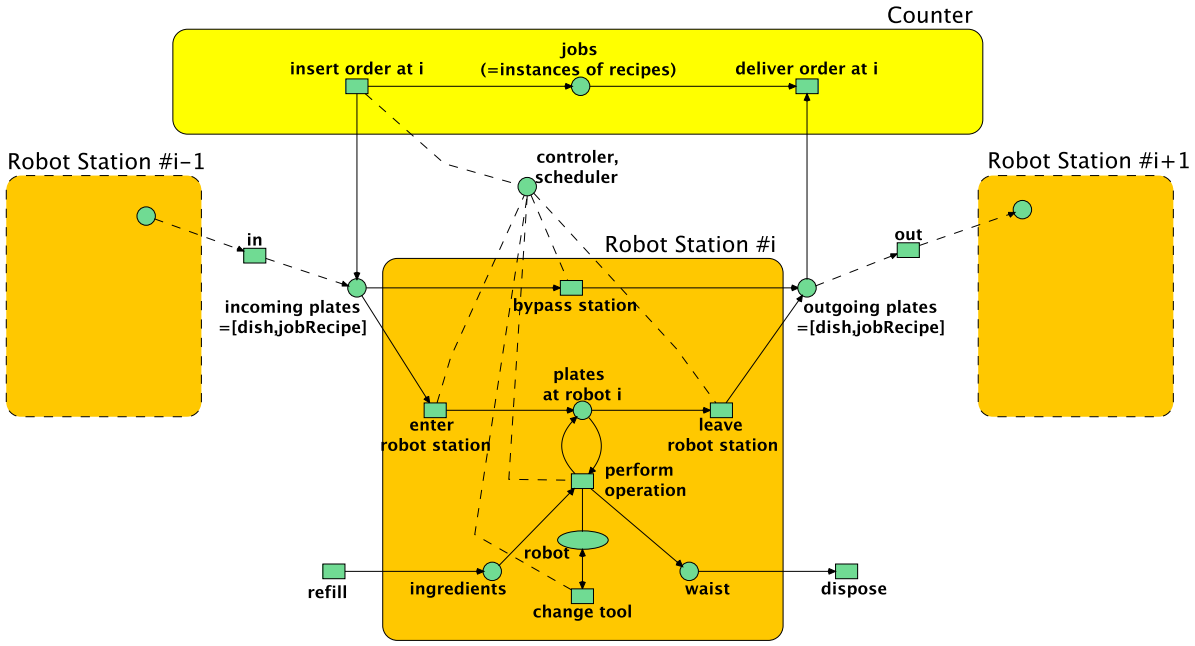

Figure 1: A Robot Station within the Kitchen

Robot Stations The robot stations describe the part that integrates the robot nets within the kitchen net, Figure 7. Based on the information from the scheduler, a synchronisation takes place at the robot station to take the correct plates with food and recipes. Furthermore, the robot station takes care of providing the needed ingredients for the robot to work and reserves the robot for the needed time to finish a step in the recipe. This time depends on the step and, as technical information, is provided by the robot. This enables a more complex time calculation. After this, the station calls the scheduler to send the created food to its next processing step and passes it to the belt. The last robot station is only responsible for delivering the created meals. In case of the assembly robot, the robot station also contains the logic for dish assembly.

Assembly The assembly is not like the other generic robot steps. Instead, it is used to combine a set of intermediate dishes to a whole dish. For this, it collects all the required partial dishes as lists. The first part of the actual assembly loop is the logic to determine whether all the dishes are present. For this a separate net is used to calculate the length of a list. This is used to determine whether the right amount of required dishes is available. The number is retrieved from the recipe. Once everything is available and a lock is acquired, the robot net can be called to assemble the dish. One challenge here is that the recipe requires the ingredients in a specific order. In other stations this is solved via the flexible arc in RENEW. With these arcs, it is possible to annotate the arc with a list of tokens and the arc is

responsible to pick the tokens, if available, from the connected place and provide a list of them to the connected transition. This makes the interaction between tokens and lists of tokens much easier within the net. An order-lock prevents mixing the intermediate products for a dish assembly.

The robot net describes some hardware and physical behaviour of the actual robots. It differentiates between tools and capabilities. While tools describe which physical tools, like knives, a cooking station or peeler a robot has access to, the capabilities are inferred from those and describe, which actions a robot can actually perform. The tools are mostly internal information, while the capabilities are more relevant for outside components like the scheduler. It can track, which tool is selected and which capabilities it provides, as well as the time required to perform each capability.

Scheduler The schedule exists within the kitchen net and is responsible for two tasks. First, it generates all the required plates for intermediate dishes based on the requested recipe. Because of this, it is used for initial assigning of new orders, as well as for scheduling new steps. Second, it assigns each plate to a fitting robot station. This way, parallel steps within the recipe can actually be processed in parallel. For this mapping, a list of available capabilities and their respective robots is used, which is stored within the kitchen.

Recipe The recipes are separate nets and contain all the required steps to create a specific dish. They are created once in the beginning of the simulation and kept as nets within the kitchen net. Each step within a recipe can use an arbitrary combination of basic ingredients and other intermediate dishes while requiring a specific capability. These are used by the robot station to perform the step. Furthermore, each step contains information, which capability is required to perform the step. This information is used by the scheduler to assign the appropriate robots. To enable easy parallel execution, multiple entry points to a recipe are provided whenever it contains steps that do not depend on each other as can be seen in Figure A. These entry points are used by the scheduler to create enough plates for each intermediate dish. The first recipe is for fried tofu with cooked beans and mashed potatoes and the second is for filled baked peppers. The respective capability usage and total duration spent for each capability can be seen in Table 1. The duration is not specified in the recipe, but in the robot, since it is dependent on the hardware how long it takes to finish the steps. For cooking, frying and baking it is assumed the robot is able to recognize when a dish is cooked. The model could also be extended to combine robot specific durations with time information from the recipe.

Table 1
Frequency and Duration of the Capabilities in the Recipes.

		Fried Tofu		Filled pepper	
Capability	Duration	Freq.	Total Duration	Freq.	Total Duration
combine	2	1	2	1	2
assemble	2	1	2	1	2
cook	15	2	30	2	30
fry	10	1	10	0	-
bake	20	0	-	1	20
mash	6	1	6	0	-
cut	5	3	15	3	15
peel	4	1	4	0	-

The base model has several configurable options. Important are the durations for the capabilities, as described in Table 1. There is a quite a big difference between some of the durations, for example between the combine and the bake steps. This affects the behavior of the kitchen and the robots workload and enable changes for runtime adaption by influencing the demand for different tools over the course of time. Furthermore, new recipe nets could be added.

2.2. Adaption Options

There are different adaption options, like changing the scheduler to change the assignment of plates to robots or the general policies. On a more permanent level, the conveyor belt itself can be changed to a more fitting pattern and new kitchen stations can be added. As a step between these two approaches, one can also enable to robots to change between active tool sets. This last approach is the one applied within this paper. To enable this, several actions in the different MAPE-loop steps need to get added.

Monitor Within the monitoring step, the workload of robots is monitored and once configurable thresholds are surpassed, analyse steps are triggered.

```
1. IF currentCapabilities < requiredCapabilities THEN analysisRequired()
2. IF unfinishedOperations > threshold THEN analysisRequired()
```

Analyse In the analysis step, the results from the monitoring are evaluated and if applicable planning actions for a reconfiguration are triggered.

```
1. IF robotUsage[capability] > upperThreshold[capability]
    THEN planRequired()
2. IF robotUsage[capability] < upperThreshold[capability]
    THEN discardMonitorEvent()
```

Plan Within the planning step, candidates for reconfiguration are identified and assigned. There are two cases covered: Firstly, if there is a robot that is underutilized and secondly, if there is no underutilized robot.

```
1. findUnterutilizedRobot(robot);planReconfigure(robot,tool);
    executeRequired(robot,tool)
2. oneOfMultipleRobots(robot,capability);planReconfigure(robot,newTool);
    executeRequired(robot,newTool)
```

Execute The execute step is responsible for ensuring the robots finish their old operations and are blocked during the process of reconfiguration. This is necessary to prevent it getting tasks it potentially can‘t fulfill anymore after switching tools.

```
1.removeromCapabilityRegister(robot); block(robot);
    performReconfiguration(robot, newTool); free(robot);
    addToCapabilityRegister(robot,newTool)
```

3. Integrating MAPE

Since focusing on all high-level self-adaption goals is out of scope of this paper, there will be a focus on a few specific ones. A basic form of self-awareness is necessary for decision-making. Additionally, global parameters control the system's goals. The major level aspects focused on in this paper are self-optimization and self-healing. Both are integrated using the lens of robot utilization. The system should discover when single robots are overworked and if possible react and improve the situation. In case no robot can provide a needed capability, the system should react and if possible reconfigure another robot. These aspects should not only work once, but continuously during the models runtime. The assumption is that by optimizing for utilization and enabling reconfiguration to that end, the system performs better and due to the monitoring and reconfiguration can deal with problems the robots have.

3.1. Conceptual Overview

Conceptually, the realizations of the different stages are separated into "tasks" that can be computed. There are different tasks to monitor, analyse, plan and execute actions. These tasks can communicate with each other using places to trigger the next step, if required. In addition, there are minor monitoring tasks, which are used to monitor and manage the places used for communication between the major tasks. The system is continuously monitored on a general level to discover potential issues. If one is discovered, the analysis tasks are scheduled to validate or discard the discovered issue. Based on the result of the analysis, available robots can decide to take action, which happens within the plan task and the actual reconfiguration is triggered within the execution task. The planned approach can be contextualized within a classification for adaptivity [14], since changing the behaviour and capabilities of robots based on the system state is planned, a basic form of compositional adaption takes place. Since the base model is not folded and has shared places for all robots, a complete reconfiguration of the robots positions can not take place.

The monitoring keeps track of the combined rate of unfinished jobs for each of the robots and based on a global threshold triggers further analysis. In the analysis process, a more detailed analysis of the situation takes place. Instead of only looking at the combined rate of unfinished jobs, it evaluates the rate of unfinished jobs per capability to be able to make capability based decisions and discover struggling capabilities. In the basic version, global and fixed thresholds are used for the different capabilities. However, these could alternatively be adapted dynamically. For the plans, robots should initially validate their ability to support a needed capability. The major deciding factor is either if a robot's own utilization is below a pre-configured minimum utilization, or it is not the only robot supporting its original capabilities. Within the execution phase, this plan is applied.

This model does not include a function to rate itself yet. This might be of particular interest for dynamical optimisation and improvement of the currently static parameters.

3.2. Extensions Required for Runtime Adaption

To realize these specific goals required for MAPE, a few extensions within the model were added:

General The first addition to the general kitchen model was to enable changing the list of robots and their capabilities at runtime. In addition, counters were added to keep track of how many robots are available for each capability and tool. Another counter was introduced to keep track of robot level scheduled and finished executions. At the end of the delivery robot, a new check is added to provide information whether all meals are delivered, which means the simulation is done. While in the default model a number of orders are pushed directly into this system, this was changed. The goal is to simulate a continuous list of orders arriving during the runtime of the system, contrary to all orders entering within one time step. For this, several new order transitions were introduced, that are supposed to feed new orders to the system in configured time intervals.

Robot The robots are organized within the main net as part of robot stations, see Figure 7. Within these stations, there are robot nets that encapsulate behavior and capabilities. To enable reconfiguration of robots at runtime, the old net had to be extended to cover the specifics for a MAPE loop. The former structure is used to provide information whether a given reconfiguration is possible. The currently selected tool is stored in a new place that also has to be initialized. The duration of using capabilities was extracted to separate transitions and can be retrieved using channels. Part of the new transition that uses capabilities is a mapping for tools and transitions. This is used to match required capabilities to the currently active tool, as can be seen in Figure A. Two counters were introduced to keep track of the absolute use of capabilities, differentiating between scheduled and finished executions, and their ratio. This information is required by the MAPE loop for decision-making. A simple blocking mechanism is included to block it from being targeted by the scheduler. To enable runtime reconfiguration, a new transition responsible for switching tools was introduced. It is available via a channel for other

nets to call. Time passes when switching. The exact moment is split into the time to remove the old tool and equip the new tool. This information can also be queried from outside the net for external decision-making. After switching tools, the robot is automatically unblocked to directly be available for scheduling.

Scheduler The scheduler was extended to react if there exist no robots with a required capability to enable the monitoring of this aspect. Since it is not possible to know if a transition did not fire, counters are used to monitor the current number of robots for a given capability. When the counter reaches zero, it means there is no robot with the corresponding tool. In case this is discovered, the scheduler notifies one of the monitoring tasks and the scheduled job waits until it can be scheduled.

For the MAPE loop, the nets were organized based on their position in the loop. All nets are colored nets that model all the robots within a single net. In a physical system, the checks would most likely happen as part of the robot stations. For readability, networks were organized separately, using virtual places. Acceptable lower and upper limits for unfinished jobs can now be configured. There are counters and limits for all the tasks part of the loop, to prevent excessive accumulation of potentially outdated scheduled actions. The limits can be configured via parameters. The whole MAPE loop starts after an initial waiting period and can be toggled off as part of the initialization.

3.3. The MAPE Loop

There is one major **monitor** task, as can be seen in the second net in Figure 2.

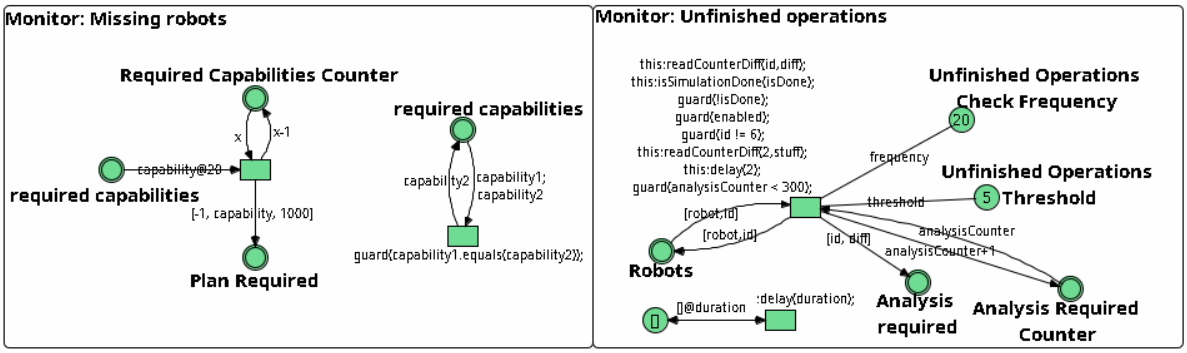

Figure 2: Two of the monitor actions. The first checks for missing robots and the other monitors the absolute utilization.

Periodically, the difference between scheduled and finished executions for each robot are checked. Once a configurable global threshold is surpassed, an analysis job is scheduled. It checks if some of the tasks can be discarded because they don‘t fit the analysis requirements. This task only runs while there are still unfinished meals. When selecting the threshold, it is important to select a value that is at least equal to the lowest configured upper limit for unfinished jobs, otherwise those will get ignored. This only works on robot-level statistics and does not yet check the more detailed, capability based statistics. The parameters passed on are the ID of the potentially problematic robot and the monitored difference.

Furthermore, there are multiple minor monitoring tasks:

- Missing robots: Triggers every time the scheduler discovers a capability that is required, but not served by any of the robots. While it would be possible to enforce that at least one robot is available for each capability, this has two problems. On the one hand, it severely limits the possible actions a system with a limited number of robots can take. If there are five tools and five robots, each robot would need to always keep that tool and would not be able to switch. On the other hand, it is not needed. When a robot has no tasks anymore, because the capability is not needed anymore, it makes sense to be able to switch to another capability that is still in high demand. With the used solution, missing robots are tolerated, as long as they are not required.

- Check “Execute Required” place: Monitors the place between plan and execute where the reconfigurations are scheduled. The first task checks if a requested reconfiguration actually already took place. The second task removes executions for blocked robots. A blocked robot is already in the process of either reconfiguration or working through its jobs with disabled scheduling to get below the limit.
- Remove duplicates: Monitors the scheduling places for the execution and planning of the reconfiguration to remove duplicate reconfigurations.

There are two **analysis** tasks. The first checks if a reported difference between the number of scheduled and finished jobs of a robot is not only above the general limit, but also its capability specific limitations. This can be seen in Figure 3. Differentiating between capabilities makes sense, since some

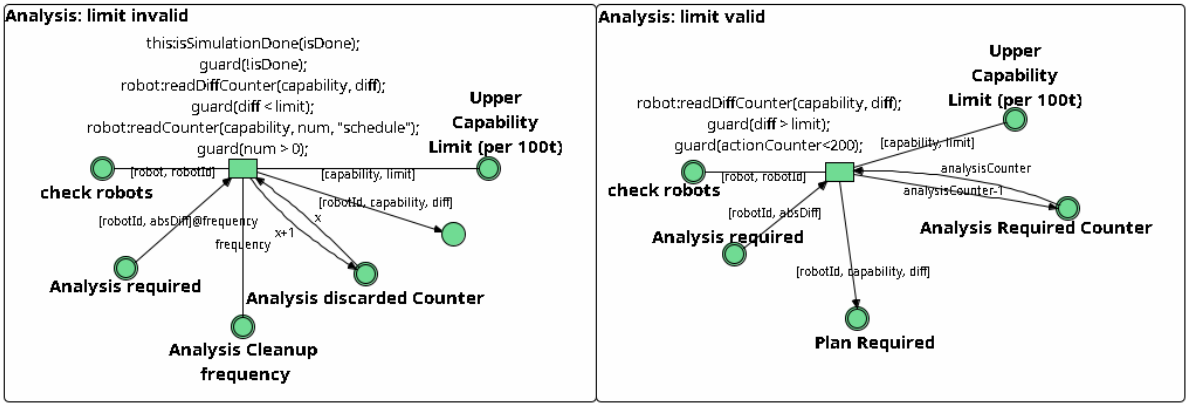

Figure 3: The analyse actions implemented. The first discards monitoring events in case the tool specific thresholds aren't actually broken. The second one triggers a planning action in case the events actually require an action.

capabilities need more time than others, which results in different expected values. If this is found to be true, a plan task is scheduled. The new parameters are the ID of the problematic robot, the problematic capability and its difference. The other task is responsible for cleaning up false positives. After a configurable delay, it checks if the capability limits are actually below their respective thresholds, as can be seen in Figure 3. This can happen if a robot has a tool that provides multiple capabilities and has a small amount of unfinished jobs for each capability, which in sum are larger than the general limit, but in detail are fine.

For the **planning** stage, there are also two major tasks. The nets can be seen in Figure 4. Robots can check if they are valid candidates for a reconfiguration. This entails a check whether the robot has the required tools available and is not already in the reconfiguration process the other validity requirement depends on the task: Based on the first task, robots are only valid if their own current workload is below a configurable limit. The second task is specialized for scenarios where a switch is required, but all available robots are already fully occupied. If multiple robots share one capability, one of those is selected. This is based on the assumption that it is better to serve different capabilities than to serve some fast and others not at all. If a robot is considered as a valid candidate for reconfiguration, it is passed to the execution task. To relieve the other problematic robot, it is blocked from scheduling and reconfiguration until its rate of unfinished jobs is again below the limit.

The **execution** happens within a single task. The implementation in RENEW can be seen in Figure 5. To prevent different reconfigurations from infringing on each other, a lock is used which limits the parallel executions. In the beginning, the robot scheduled for reconfiguration is removed from the list of available robots for scheduling and is blocked. This prevents it from receiving new jobs. Once it has finished its previously assigned jobs, it performs the reconfiguration, unblocks and is registered with its new capabilities. There is a synchronization between monitoring, planning and execution: Once a robot performs a reconfiguration and is blocked, it is exempt from planning or monitoring, since it is in a state of transition. Similarly, a robot that is currently in the process of planning or being monitored,

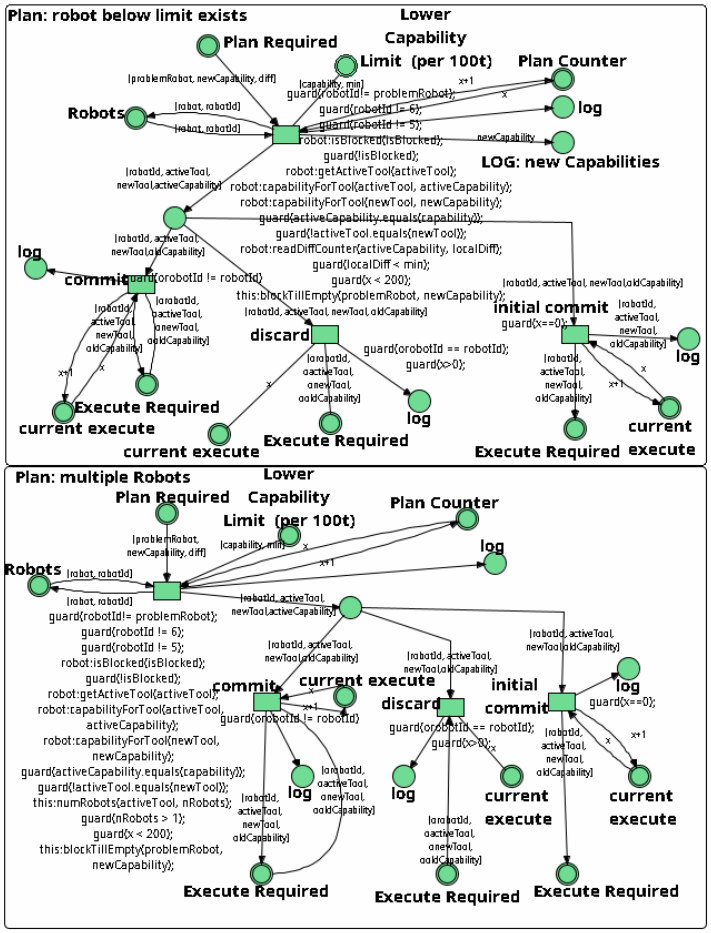

Figure 4: The two plan actions implemented. The first plan acts in case there is an underutilized robot. The second plan checks for multiple robots sharing the same capabilities and chooses one of them to reinforce the overloaded robot.

is not able to reconfigure at the same time. Once the reconfiguration took place, the robot is again available to perform the other steps of the loop and the lock is released.

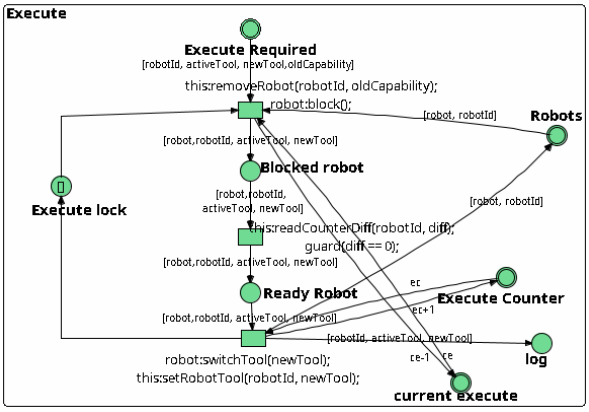

Figure 5: The MAPE execute step. The robots gets removed via a channel from the registry of available robots, blocked to not receive any new jobs or be subject to other adaptions and finally reconfigured.

4. Evaluation

When determining the available tools for the robots, there are a few limitations within the original simulation environment. In the simulation there are only six robots. One is basic and only used for delivery. Another one is specialized for assembly. While that wouldn't prevent it from switching tools, no other robot can adequately perform the assembly and that way the assembly needs to wait for everything to be finished, it requires the scheduler. While it might be possible to finish partial assembles in theory, it is not possible in the current model. Because of this, there are actually only four robots available for reconfiguration. Since the main kitchen loop network with the different stations and positions is explicitly modelled, it is not that easy to just add multiple new robots. With five required tools and, depending on the required configuration, four or five robots, there is not much room for reconfiguration. Because of this, the four free robots are configured to have all possible tools available, to maximize the options for reconfiguration. Each robot has a different tool selected at the beginning of the simulation to have full initial coverage of all tools. Since the assembly could not be shared between robots, the corresponding capability was excluded from the reconfiguration.

Table 2
Values for desired system state limits.

capability	lower limit	upper limit
combine	5	8
assemble	-	-
cook	2	13
fry	2	10
bake	2	10
mash	4	18
cut	4	18
peel	4	20

Table 3
Average results after 5 simulations.

	First Meal finished		Last Meal Finished	
	Default	MAPE	Default	MAPE
Steps	412	330	1800	1350

One observation during the development of the different possible MAPE tasks was that new tasks could introduce new challenges and problems within the system. As an example, switching tools can help to reduce the load, but at the same time it can leave capabilities not covered. This in turn requires dedicated monitoring and handling, not only within the loop, but also in other components like the scheduler that need to be able to handle those system states.

For the simulation, 40 orders were served. This proved to already provide sufficient orders to make adaption beneficial. The values used for the system utilization limits, which can trigger reconfiguration, are described in Table 2. These can be configured based on the requirements. Since there is no reconfiguration for the assembly capability, there is no need for a limit.

During the simulation it can be observed that as soon as the loop starts, the robots 1 or 3 reconfigure once their jobs are finished and switch to the cooking station to support robot 4. Once the next batches of orders arrive, the robots switch back again to provide the temporarily not covered capabilities *peel* and *mash*. Switches happen mostly between cooking and peeling or mashing and cutting and peeling or mashing, which is not surprising when considering the total time needed on those activities based on the selected recipes.

An interesting behaviour that could be observed was that robot 1 switches to the cooking station, finishes a few jobs and directly switches again to the knife to help cutting. This rather rapid rate of reconfiguration might not always be desirable. To limit it, one could add a form of cool-down until a robot is again available for reconfiguration, instead of directly returning it to the pool of available robots.

While the robots can focus on their work, it is currently not possible to switch to robots that stopped working, if not other robots are under their workload limit or share capabilities due to a lack of a matching planning algorithm. In that case the un-supported jobs have to wait until enough other jobs have finished, which comes at the expense of the total duration. Since only a basic scheduler is used, no rescheduling is possible at the moment. Because of this, a completely broken robot would result in a

loss of recipes, which makes it impossible for dishes to complete. The use of the MAPE loop resulted in a speed-up of the creation of the dishes, as can be seen in Table 3. However, this comes at the cost of computation time. Due to the additional computations required by the MAPE processes, the simulation time to evaluate almost doubles when using the MAPE loop.

5. Conclusion

In this paper a simple MAPE loop for a Petri net based kitchen simulation was developed and modelled. The kitchen serves as a metaphor for flexible production systems in general. When comparing the simulation results without a loop with those obtained when enabling it, it was found that enabling the system to adapt resulted in faster delivery of the meals, at the cost of higher computational requirements. Due to the monitoring of the system, a unavailable robot can be discovered and replaced by another robot taking its place.

However, it was found that only adding a MAPE loop without significantly extending the scheduler to take care of rescheduling of jobs and a mechanism to memorize jobs is not enough to achieve complete independence of failing robots. Furthermore, it was found that even simple self-adaptive processes can have possibly unintended effects on the system and pose a challenge to its overall design. To fully make use of self-adaptive systems and their benefits, the whole system needs to be adapted to those ends. This highlights the need for a systematic approach to introducing MAPE loops, with multiple iterations to enable the system to properly deal with the desired level of self-adaptability.

The used model currently only supports a basic configuration and leaves room for improvement on that end, especially in regard to prioritized selection of reconfiguration candidates, considering setup times and enabling more significant changes during runtime. Extending the model with more complex scheduling capabilities could also be of interest. On a more abstract level, it would be interesting to have a closer look at the process of adding MAPE loops to systems and which methodological approaches it would require or benefit from. It was already shown that introducing self-adaptability can introduce new system states and require changes in the system to properly handle those. A systematic approach to dealing with those challenges could be beneficial.

Declaration on Generative AI

The author(s) have not employed any Generative AI tools.

References

[1] E. R. Griffor, C. Greer, D. A. Wollman, M. J. Burns, Framework for cyber-physical systems: volume 1, overview, National Institute of Standards and Technology (2017). doi:10.6028/nist.sp.1500-201.

[2] B. H. C. Cheng, others, Software engineering for self-adaptive systems: A research roadmap, in: B. H. C. Cheng, R. de Lemos, H. Giese, P. Inverardi, J. Magee (Eds.), Software Engineering for Self-Adaptive Systems, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 1–26.

[3] P. Leitao, S. Karnouskos, L. Ribeiro, J. Lee, T. Strasser, A. W. Colombo, Smart agents in industrial cyber-physical systems, Proceedings of the IEEE 104 (2016) 1086–1101.

[4] P. Leitao, S. Karnouskos, Industrial Agents: Emerging Applications of Software Agents in Industry, 1st ed., Elsevier Science Publishers B. V., Amsterdam, The Netherlands, 2015.

[5] D. Weyns, An Introduction to Self-Adaptive Systems: A Contemporary Software Engineering Perspective, John Wiley & Sons Ltd, 2020.

[6] V. Dignum, J. Padget, Multiagent organizations, in: G. Weiss (Ed.), Multiagent Systems, 2nd ed., Intelligent Robotics; Autonomous Agents Series, MIT Press, 2013, pp. 51–98.

[7] J. Sudeikat, M. Köhler-Bußmeier, On combining domain modeling and organizational modeling for developing adaptive cyber-physical systems, in: ICAART'22, 2022.

[8] M. Köhler-Bußmeier, J. Sudeikat, Balance vs. contingency: Adaption measures for organizational multi-agent systems, in: K. Jander, L. Braubach, C. Badica (Eds.), 15th International Symposium on Intelligent Distributed Computing (IDC'22), volume 1089 of *Studies in Computational Intelligence*, Springer-Verlag, 2023, pp. 224–233.

[9] O. Kummer, F. Wienberg, M. Duvigneau, J. Schumacher, M. Köhler, D. Moldt, H. Rölke, R. Valk, An extensible editor and simulation engine for Petri nets: Renew, in: J. Cortadella, W. Reisig (Eds.), International Conference on Application and Theory of Petri Nets 2004, volume 3099 of *Lecture Notes in Computer Science*, Springer-Verlag, 2004, pp. 484 – 493.

[10] R. Valk, Object Petri nets: Using the nets-within-nets paradigm, in: J. Desel, W. Reisig, G. Rozenberg (Eds.), Advanced Course on Petri Nets 2003, volume 3098 of *Lecture Notes in Computer Science*, Springer-Verlag, 2003, pp. 819–848.

[11] O. Boissier, R. H. Bordini, J. Hübner, A. Ricci, Multi-Agent Oriented Programming Programming Multi-Agent Systems Using JaCaMo, The MIT Press, 2020.

[12] K. Jensen, Coloured Petri nets: basic concepts, analysis methods and practical use, vol. 2, Springer-Verlag, Berlin, Heidelberg, 1995.

[13] V. Gehlot, From Petri Nets to Colored Petri Nets: A tutorial introduction to nets based formalism for modeling and simulation, in: 2019 Winter Simulation Conference (WSC), 2019, pp. 1519–1533. doi:10.1109/WSC40007.2019.9004691.

[14] K. Geihs, Selbst-adaptive Software, Informatik-Spektrum 31 (2008) 133–145. URL: https://doi.org/10.1007/s00287-007-0198-9. doi:10.1007/s00287-007-0198-9.

A. Petri Nets

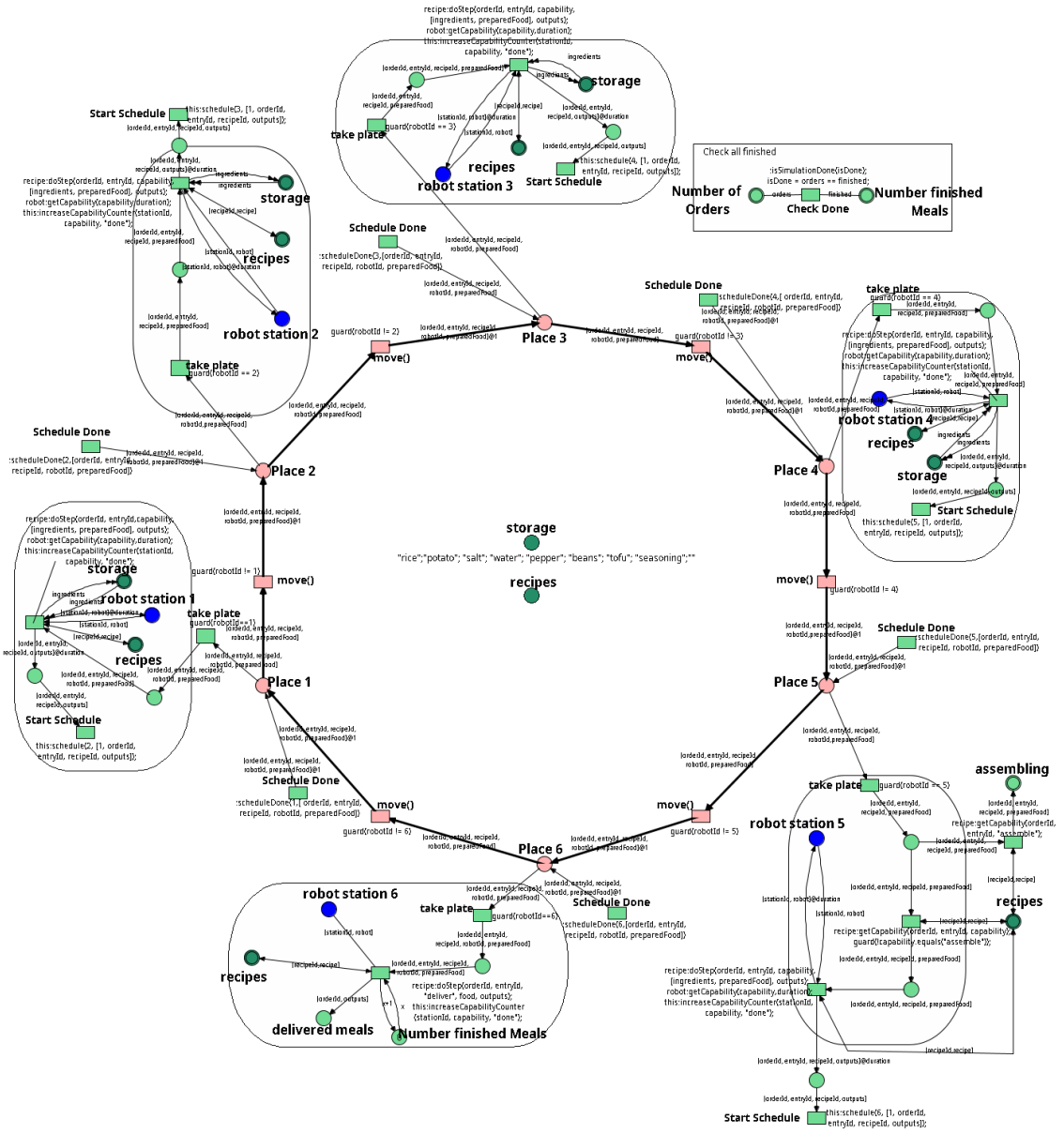

Figure 6: Overview of the kitchen net with robot stations and conveyor belt with places. Monitoring endpoints, initialization and the details of assembly and the scheduler are omitted, but can be found in the repository.

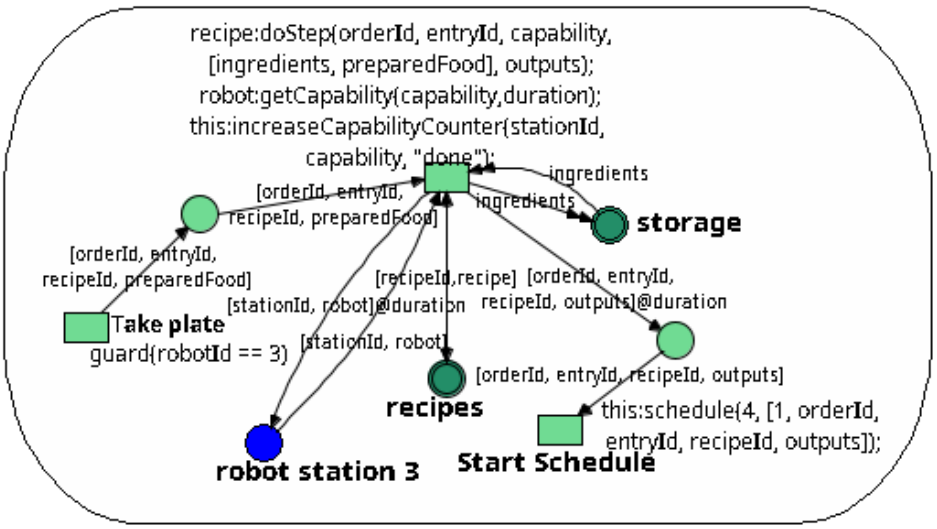

Figure 7: Overview of a robot station within the kitchen net.

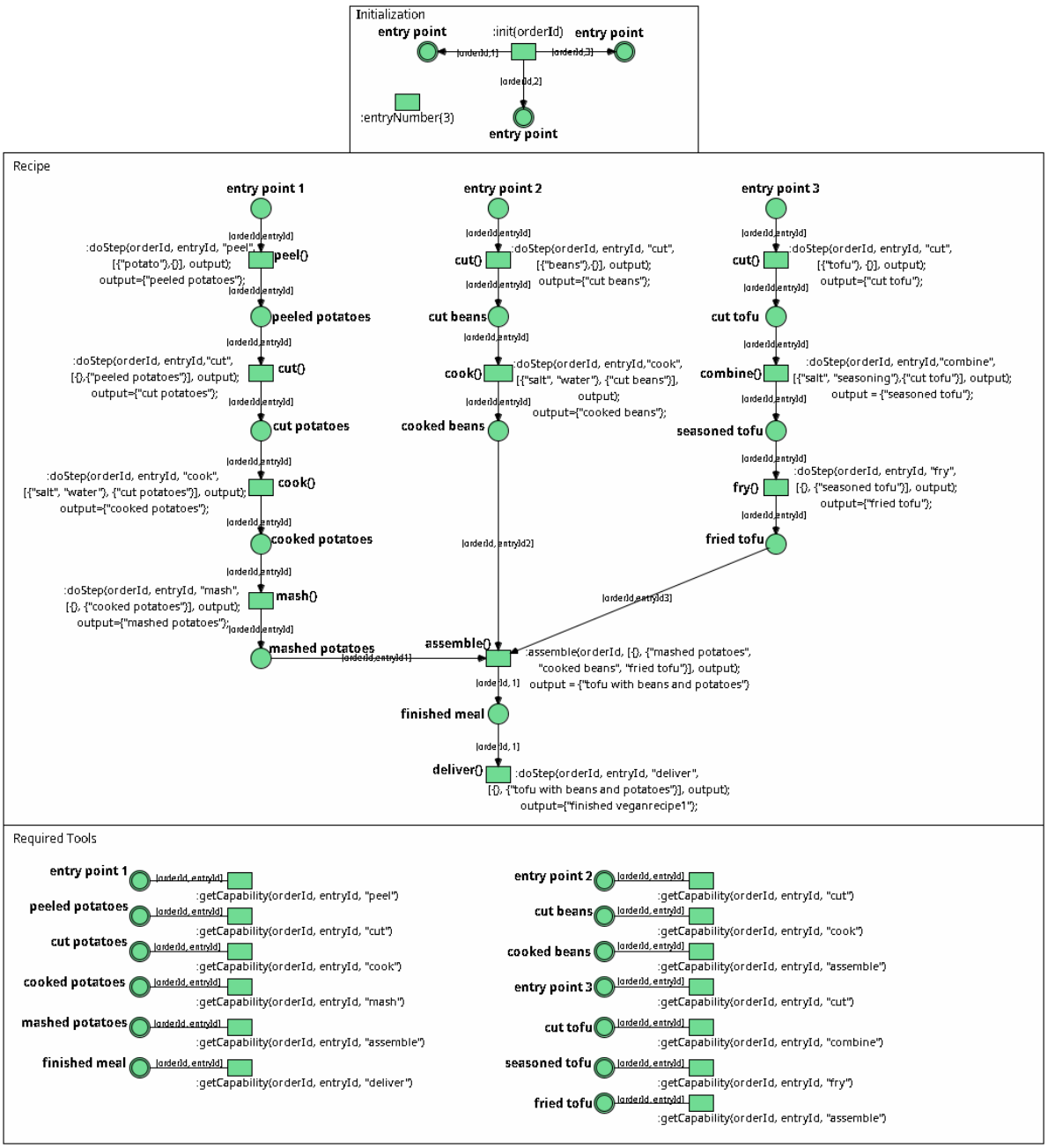

Figure 8: A Recipe net, containing multiple entry points for parallel execution and channels to retrieve the required capabilities for each execution step.

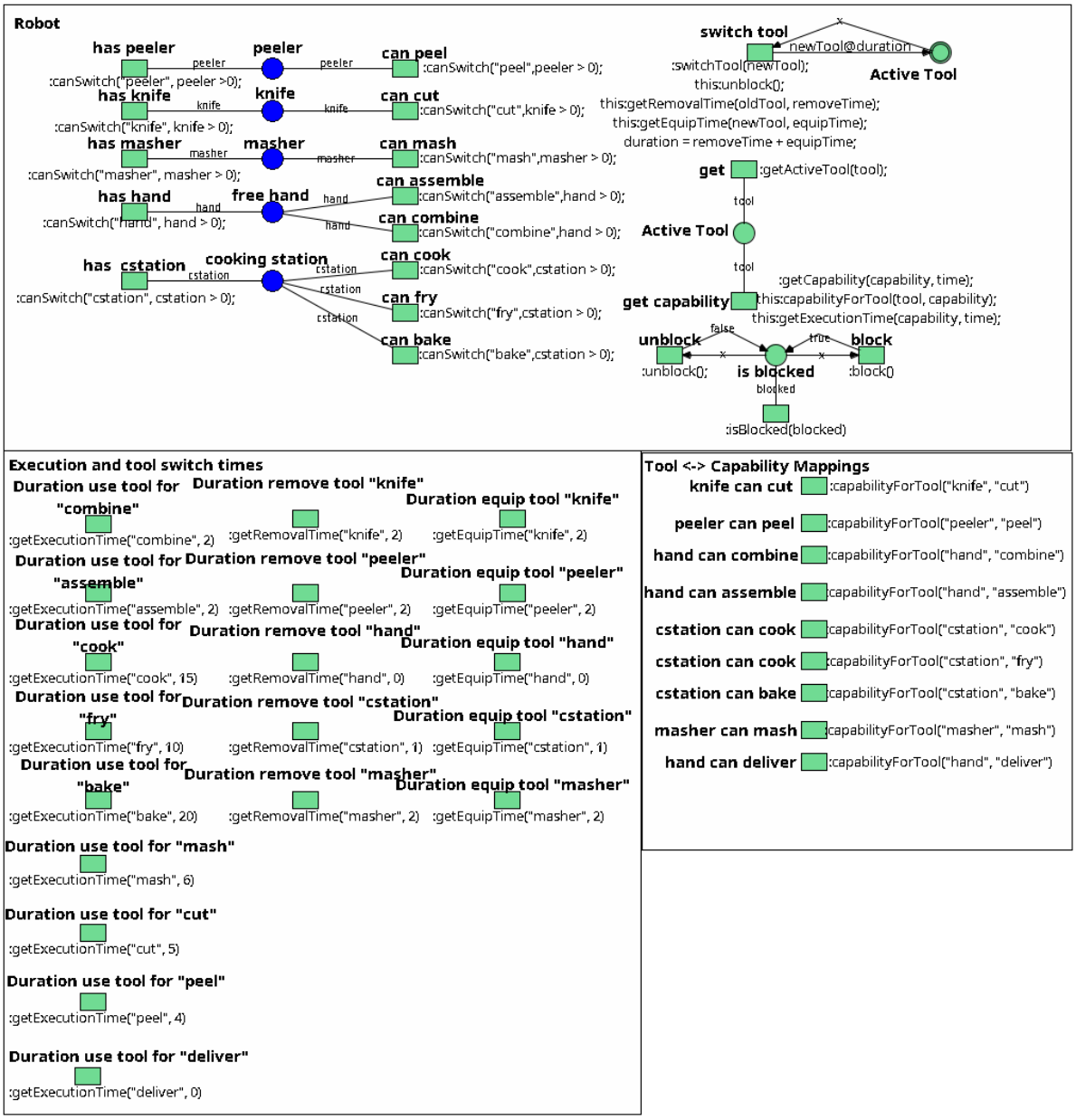

Figure 9: Overview of the robot net. The part responsible for monitoring and initialization are omitted and can be found in the repository.