

su-chef: Adaptive coordination of intelligent home environments

Evangelos Kotsovinos and Maja Vukovic
University of Cambridge Computer Laboratory

Abstract

Traditional recipes are the static products of an empirical, off-line procedure, and contain implicit environmental assumptions, as they are devised based on the exact ingredients and household devices that were available in the environment where they were created. When the parameters of the environment in which a dish is being cooked do not match those of the one in which the corresponding recipe was prepared, frustration is imminent.

We present our work on dynamic service composition to facilitate adaptive coordination of smart home environments, and focus on su-chef, an application of these techniques in the cooking domain. We present initial evaluation results demonstrating that our implementation provides a practical and scalable solution.

1 Introduction

Inaccuracy and often failure of the cooking process can happen when parameters of the environment in which the dish is cooked do not match those of the environment where the recipe was prepared, or when unanticipated changes occur – e.g. failure of a device. Augmenting recipes with conditional statements – such as `case` structures introduces complexity to recipe preparation and execution, does not accommodate for *unanticipated* environmental parameters – e.g. when new household devices going into the market, and does not support the adaptation of recipes when ingredients or devices become unavailable.

We propose *goal-driven recipes* or *greplets*, which, unlike recipes, do not contain instructions, but information about the state each of the ingredients has to be brought at in all intermediate stages for the cooking process to succeed. We use *AI planning* to dynamically compose

recipes based on the goals given in greplets and contextual information, and to re-build recipes if environmental information, such as availability of ingredients or status of devices, changes. While we choose to focus on an application of the our methodology on cooking, this does not limit the applicability of the system in other domains.

In this work, we assume a secure smart home environment – free of malicious devices – able to find and control devices, and discover ingredients and utensils and their condition. We examine an example usage scenario in Section 2, analyse our framework in Section 3, and present our implementation in Section 4. In Section 5 we discuss our initial evaluation results. In Section 6 we position our work in the relevant research context, and in Section 7 we conclude and outline future work.

2 Usage scenario

Gertrude’s kitchen is equipped with sensors that detect, for instance, the availability of ingredients and the condition of devices and dishes that are being cooked. We assume that the household devices can receive instructions either directly from a computer, or through a device proxy, as discussed in Section 6.

Discovering what is available. Gertrude buys the ingredients required to make lasagne and places them on the kitchen counter. Her smart kitchen automatically recognises the ingredients, their quantities, and their characteristics – e.g. minced meat is tagged with information about its thermal conductivity. Household devices also advertise their availability, capabilities, and characteristics.

Building a recipe. Gertrude asks for a lighter version of “that pasta and minced meat dish she had a few days ago”,

1. Place onions in chopper
2. Place mushrooms in slicer
3. Place mozzarella in grater
4. Put one tablespoon of oil in frying pan
5. Place frying pan on hot plate
6. Chopper: Chop onions in 50x50x50mm cubes
7. Slicer: Slice mushrooms in 60mm slices
8. Grater: Grate mozzarella at 0.5mm thickness
9. Hot plate: Turn on at maximum temperature
10. Hot plate: Wait for 2 minutes 45 seconds
11. Add onions in pan
12. Hot plate: Wait for 3 minutes 30 seconds
13. Add minced beef in pan
14. Hot plate: Wait for 5 minutes
15. Add tomatoes, mushrooms, herbs and spices in pan
16. Hot plate: Wait for 1 minute 30 seconds
17. Repeat 4 times:
 - Lay lasagne sheet in baking tray*
 - Place 1/4 of pan contents in baking tray*
18. Lay lasagne sheet in baking tray
19. Top with mozzarella
20. Oven: Turn on at 200 degrees
21. Oven: Wait for 8 minutes
22. Place baking tray in oven
23. Oven: Wait for 47 minutes
24. Take baking tray out of oven

Figure 1. Example recipe produced by su-chef, in human-readable form.

as she is on diet. su-chef finds the sequence of goals to be reached for the cooking process to succeed – the greplet for lasagne – by searching in the list of recently prepared dishes. Noticing that mozzarella is required but not available, su-chef decides to use provolone cheese, which is similar to mozzarella and available. Taking into account Gertrude’s preference for a lighter meal, su-chef also decides not to top the dish with the usual white, creamy sauce. According to the available devices and ingredients, and the greplet, su-chef produces the recipe in Figure 1.

Using the recipe. Ingredients cannot be automatically moved around and transported to devices, as the kitchen is not equipped with conveyor belts and robot arms – in Figure 1, the actions that Gertrude needs to take are shown in italics. su-chef minimises Gertrude’s inconvenience by grouping user actions together if possible. Gertrude is initially asked to place ingredients in the right devices; after that she can rest, and will be automatically alerted when further user actions need to be taken – e.g. adding more ingredients. Even if no automation mechanisms are available for transporting ingredients between devices, cooking is significantly *simplified*; all Gertrude needs to do is follow trivial instructions – which ingredient to place in which device.

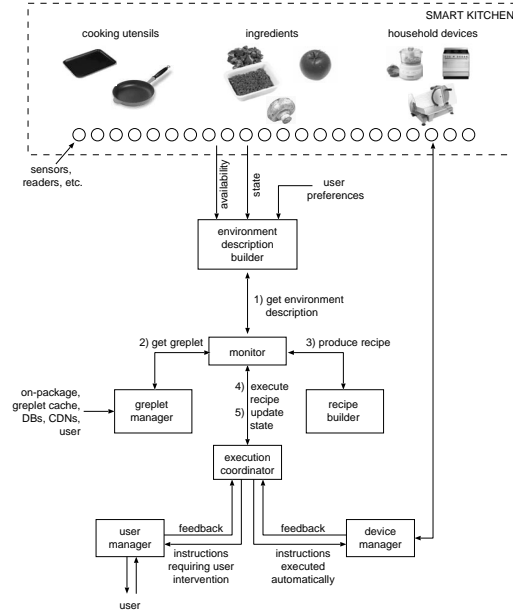


Figure 2. The su-chef architecture.

Dealing with the unexpected. su-chef is at step 12 of the recipe, sauteing the onions that Gertrude has just added, when Zacharias, Gertrude’s dog, enters the kitchen and eats the minced beef that was on the kitchen counter.

When su-chef realises that a necessary ingredient is no longer available, it tries to adapt and build a different recipe. Knowing that there is minced chicken and minced lamb in Gertrude’s fridge, it looks up for dishes similar to beef lasagne that can be cooked with the available ingredients, and figures out that lasagne can be made using either chicken or lamb. Since Gertrude prefers low-calorie dishes, su-chef decides to go for chicken, and produces a new recipe where step 13 of Figure 1 is replaced with “Add minced chicken in pan” and step 14 with “Hot plate: Wait for 4 minutes” – chicken cooks slightly quicker than the previously available beef.

3 System architecture

The operation of su-chef can be separated in five steps, as shown in Figure 2.

Get environment description. The environment description contains information about ingredients, utensils, and devices present. *Availability* of an item describes whether it is present and operational, as well as its phys-

ical characteristics – the minced beef’s weight and thermal conductivity – or or capabilities – such as the oven’s temperature-time curve. An item’s *state* is not part of its static characteristics, and can not be on the tag it carries – e.g. the current oven temperature and whether the onion is sauteed. *User preferences* are used in selecting the dish to be cooked as well as in customising the recipe.

The *environment description builder* constructs a structured environment representation in the common su-chef language – operation 1 in Figure 2 – by collecting representations of available items – potentially expressed in different languages, their state, and the user specifications.

The monitor requests the current environment description periodically; being aware of the greplet and recipe that are being used, it checks whether the availability and state of items allows for the actions specified in the recipe to be performed. When “unexpected” changes occur the monitor may trigger recipe recomposition.

Get greplet. The monitor submits the environment description to the *greplet manager* – operation 2 in Figure 2. The manager *acquires* an existing greplet from a source (e.g. local greplet cache, a remote greplet database, or a content distribution network), and then *customises* it according to user preferences. If so configured, the manager can also contact intelligent external services that are able to develop entirely new greplets.

Produce recipe. The monitor passes the environment description and the greplet to the *recipe builder*, which composes the recipe to be used dynamically, based on the goals defined in the greplet and according to item availability and initial state – operation 3 in Figure 2.

Execute recipe. The monitor feeds the recipe to the *execution coordinator*, which ensures that recipe instructions are carried out in the right *order* and *timing* – operation 4 in Figure 2. Instructions of the recipe that are to be executed manually by the user – e.g. “place onion in chopper” if no robot arm is available – are passed on to the *user manager* component, which alerts the user and returns feedback to the coordinator when the user indicates that actions are finished. The rest are executed by direct communication with devices, coordinated by the *device*

manager, which maintains any device-specific knowledge and hardware required.

Update state. The monitor periodically requests a description of the current environment, and evaluates changes in it. “Expected” changes – for instance, “oven temperature is at 200 degrees” – do not affect the recipe itself, as the sequence of instructions remains the same. They are passed on to the execution coordinator as *state updates* – operation 5 in Figure 2 as feedback complementary to the timings of instructions that are devised at recipe composition time.

4 Implementation

In this section, we analyse the internals of the components described in the previous section. We have implemented a software module to feed a list of available devices, ingredients, and characteristics to su-chef and generate unexpected changes, and we are confident that integration of our work in a real intelligent environment testbed would be reasonably straightforward.

Environment Description Builder The environment is represented in STRIPS [7], where each state of the world is a first-order logic atom. An example description is shown in below, where we define an *onion*, which can be either *whole* or *chopped*. Furthermore, each device, such as an *oven*, is described in terms of their characteristics and current state. For example, the atom `(device_temperature oven_1 0)` is used to define the oven’s temperature, which is initially set to 0 degrees.

```
(vegetable onion) (whole onion)
(quantity onion piece 3)
(device oven) (oven oven_1)
(device_temperature oven_1 0) (can_bake oven)
(available oven_1) (operational oven_1)
```

Possible *actions* in the environment description are state transition functions, expressed as STRIPS operators, as shown below.

```
(def-strips-operator (chop ?i ?u)
  (pre (vegetable ?i) (width ?x)
        (height ?y) (length ?z)
        (whole ?i) (can_chop ?u) (raw ?i))
  (add (chopped ?i ?x ?y ?z))
  (del (whole ?i)))
```

Each defines a precondition list, an add list reflecting the new state of an item after the operation has been performed on it – for instance, “onion is now chopped”, and a delete list representing the state that it is no longer in – “onion is no longer in one piece” (whole).

Greplet manager Below is a greplet, also represented in STRIPS, for the first stage of lasagne cooking – preparation of ingredients. In that example, the (grated mozzarella grate_thickness) atom describes that cheese should be grated to a specified thickness.

```
(define (preparation_step)
  (chopped onion cube_width cube_height cube_length)
  (sliced mushroom slice_thickness)
  (grated mozzarella grate_thickness)
  (whole tomato))
```

To obtain a greplet, the greplet manager first looks up in the local *greplet cache* to reduce unnecessary remote lookups. If that does not succeed, su-chef contacts a remote *greplet database*, which stores a larger number of greplets but is more costly to look up in – in terms of time, as it is remote, and money, as it may charge for queries.

Monitor The monitor observes the cooking process, evaluates changes in the environment, and reasons about the necessity of adapting the recipe or notifying the execution coordinator about state updates. When *adaptation of the recipe* is required – for instance, when an ingredient is no longer available – re-planning is triggered to compose a new recipe that can be executed in the current environment. This requires a search for a new, suitable greplet, which is then passed on to the recipe builder. When *adaptation of the execution* is required – for example, when the state of an ingredient changes to indicate that a step is complete, such as “oven warmed up” – the monitor deploys state updates to the execution coordinator.

To achieve this, we use monitoring procedures [9] defining the actions and events to be monitored, the methods that can be used for monitoring each action or event, and common recovery procedures. At present all ingredients and devices carry their assigned category (e.g. chicken mince → minced meat → meat), allowing us to substitute unavailable ingredients or devices. We plan to investigate using a more comprehensive domain ontology to facilitate smarter substitution.

Recipe Builder Dynamic composition of the recipe can be mapped to the planning problem – finding a finite sequence of actions that will transform a given initial state (i.e. available ingredients) to a state that satisfies a given goal (i.e. wanted dish). The core part of the recipe builder is an embedded planner, which takes the domain definition and initial state – in our terms, *environment description* – and goal – a *greplet* – as input to produce a recipe.

su-chef requires modelling a number of sophisticated features in composition, such as: *concurrently executing actions with varying action durations* (e.g. chopping onions, slicing mushrooms, and heating up hot plate simultaneously), *iteration* (e.g. put lasagne sheet then cover it with sauce – repeat three times), and *resource constraints* (e.g. optimise the recipe based on preparation time or even energy consumed by the devices. Our prototype employs TLPlan [6], a forward-chaining search planning system, as it meets our functionality requirements.

```
(chop chopper onion)
(slice food_processor mushroom slice_thickness)
(grate grater provolone grate_thickness)
```

The code above shows the plan generated for the preparation step. Despite the fact that the greplet originally required mozzarella, and that Gertrude ran out of it, su-chef still successfully devises a recipe – and suggests use of provolone instead. We have implemented support for *weak goals* to facilitate lookup of similar ingredients during the planning process, meaning that planning will not fail but rather ingredient substitution will occur.

Execution Coordinator Our prototype execution coordinator component uses the Business Process Execution Language For Web Services (BPEL4WS)¹ to describe the steps to be taken as interactions with the user and device managers and orchestrate their execution.

The cooking instructions produced by the recipe builder are translated automatically from STRIPS to BPEL4WS. The coordinator distinguishes the instructions to be carried out by the user and the devices, and accordingly assigns those to either the corresponding Manager. The resulting BPEL4WS plan is then executed on the IBM BPEL for Web Services JavaTM Run Time (BPWS4J)².

¹<http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/>

²<http://www.alphaworks.ibm.com/tech/bpws4j>

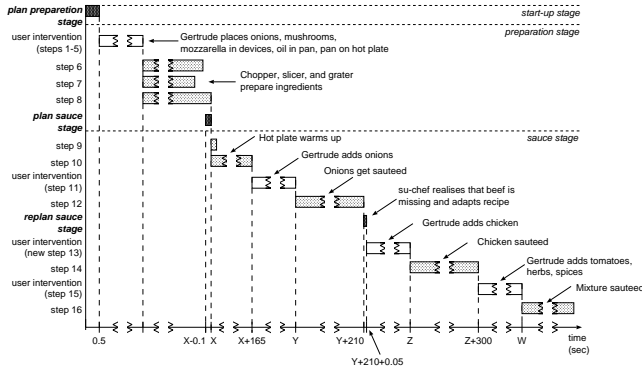


Figure 3. Timeline of su-chef's operation

User and Device Managers The user manager requests attention when a task is to be executed by the user by playing an audio file. It displays the task on the screen on a web-based interface. We anticipate to extend this by introducing a context-aware notification dispatcher to address the issue of personal mobility.

su-chef employs Web Service technology to represent kitchen devices, by describing operations accessible through standard XML messaging. The device manager receives requests from the main BPEL4WS execution description and accordingly launches and coordinates the services representing the requested devices.

5 Evaluation

The experiments were performed on an 800 MHz P3 with 2 GB RAM. We modelled a kitchen environment that contained 400 axioms – ingredients, devices, etc. We ran TLPlan in breadth first search mode with no search control knowledge.

Cooking timeline The time that each step required for the first two stages of lasagne cooking – preparation and sauce cooking – is shown in Figure 3. The step numbers in this figure correspond to the steps as described in Figure 1.

The system *interleaves* planning – i.e. recipe composition – and execution; it composes sub-recipes for each stage in the greplet separately and as late as possible – for example, the sauce stage is composed just before step 9, as shown in Figure 3. This has two main advantages; first, the start-up latency is minimised – in this case, less than half a second. Second, it ensures that sub-recipes are

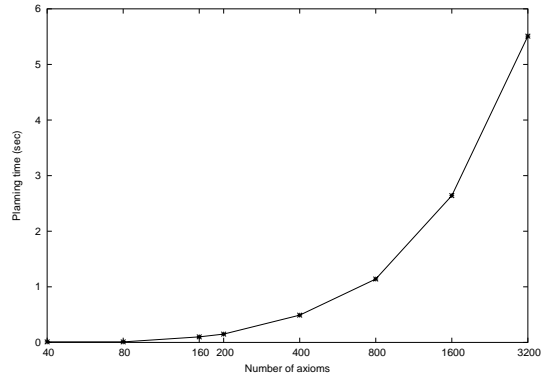


Figure 4. Recipe composition time as environment size increases (log scale)

devised based on a picture of the environment that is as up-to-date as possible, thus reducing the probability that recomposition to adapt to changes will be required.

One important result from our evaluation is that recipe composition – in a relatively rich kitchen domain of 400 items – is more than adequately efficient, requiring about 0.1 seconds for the sauce preparation stage on inexpensive hardware. Furthermore, as shown in Figure 3 – just before step 13, recomposing the recipe due to changes in ingredient availability is also sufficiently efficient (0.05 sec).

Scalability We measured the time required to create a recipe for the sauce preparation stage for increasing initial environment sizes, ranging from the minimal number of axioms to allow for lasagne to be cooked, to an exceptionally rich environment containing 3200 axioms.

As shown in Figure 4, su-chef performs well for smart worlds of a relatively limited size – up to around 1000 axioms, such as home or kitchen environments, requiring less than two seconds. However, for larger environments, containing hundreds of thousands or millions of axioms, the applicability of this technique is limited, as performance degrades linearly. This is hardly surprising – planning is by nature computationally difficult, even for the restricted versions of the planning problems.

Discussion We have come across several important research challenges that planning needs to address before it can be applicable to the full spectrum of process coordination in smart spaces. First, the system needs to be

resilient to *planner failures* – e.g. “no plan found”, arising due to incomplete or incorrect knowledge of the world – e.g. corrupted device descriptions or incorrect greplets. Furthermore, the coordination of smart environments often involves *looping*, *conditional*, and *concurrent* actions, thus planning technology needs to be able to automatically assemble non-sequential complex processes from atomic actions. TLPlan supports all these control constructs, as analysed in [12], but only if – often painstakingly hand-coded – templates indicating the non-sequential actions are provided by the user.

6 Research context

The approach closest to our work – in terms of using planning for home automation – is chef [10], a case-based planner that builds new recipes through adaptation of existing ones. In our system, this corresponds to functionality the greplet builder provides – building a greplet or adapting an existing one, based on the available ingredients and devices. Our system does more than that: it dynamically composes recipes based on current environmental information, and integrates the overall cooking process in intelligent environments, facilitating monitoring, execution, and adaptation to run-time changes, therefore constituting a superset of chef.

Intelligent home environments [2, 5, 3], built on sensor technology and smart appliances, and smart objects [8, 11], augmenting everyday objects with computer technology and sensors, compose the environment in which su-chef is envisaged to operate. Research efforts in food science have achieved the assessment of food properties, such as textural changes [4], sweetness and consistency [1], using electronic sensing systems to facilitate food development and processing. su-chef utilises such sensory systems, if available, to enable refined detection of the changes in the state of ingredients (e.g. onion has been sauteed).

7 Conclusions and future work

In this paper, we identified challenges for coordinating smart home environments as results of their dynamism and unpredictability, using the example of recipes. We

proposed su-chef; a system that automates coordination of a smart kitchen by building and adapting recipes at run-time based on greplets. The core of su-chef is the monitor that observes environmental changes, and the coordinator that orchestrates the actions of users and devices.

At present we are looking at privacy issues related to the exposure of users’ gastronomic preferences. In the future, we aim to integrate su-chef in an intelligent home environment for further experimentation. Furthermore, we will investigate the use of formal methods for greplet verification, to control the creation of dangerous mixtures that might, for instance, be poisonous or explosive.

References

- [1] Durum wheat semolinas and alimentary pasta – Estimation of cooking quality of spaghetti by sensory analysis. Technical Report ISO 7304:1985, ISO, 1985.
- [2] R. Brooks. The Intelligent Room Project. In *Proc. of the 2nd Int. Cognitive Technology Conf.*, Aizu, Japan, 1997.
- [3] B. Brumitt, B. Meyers, J. Krumm, A. Kern, and S. A. Shafer. EasyLiving: Technologies for Intelligent Environments. In *HUC*, pages 12–29, 2000.
- [4] L. Carson, X. Sun, C. Setser, and Y. Peng. Assessing Food Firmness Using an Electronic Sensing System with a Model Food System. *Journal of Texture Studies*, 33(5), 2002.
- [5] C. D. K. et al. The Aware Home: A Living Laboratory for Ubiquitous Computing Research. In *Cooperative Buildings*, pages 191–198, 1999.
- [6] Fahiem Bacchus and Frodoald Kabanza. Using Temporal Logics to Express Search Control Knowledge for Planning. *Artificial Intelligence*, 116(1-2):123–191, 2000.
- [7] R. E. Fikes and N. J. Nilsson. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 2(3-4):189–208, 1971.
- [8] H.-W. Gellersen, M. Beigl, and H. Krull. The MediaCup: Awareness Technology Embedded in an Everyday Object. *Lecture Notes in Computer Science*, 1707, 1999.
- [9] K. Z. Haigh and M. Veloso. Interleaving Planning and Robot Execution for Asynchronous User Requests. In *Planning with Incomplete Information for Robot Problems: AAAI Spring Symposium 1996*, pages 35–44.
- [10] K. J. Hammond. Chef: A model of case-based planning. In *Proc. of the 5th Nat. Conf. on Artificial Intelligence*, volume 1, pages 261–271, Aug. 1986.
- [11] T. Selker, E. Arroyo, and W. Burleson. Chameleon Tables: Using Context Information in Everyday Objects. In *CHI '02*, pages 580–581. ACM Press, 2002.
- [12] M. Vukovic and P. Robinson. SHOP2 and TLPlan for proactive service composition. In *UK-Russia Workshop on Proactive Computing*, Feb. 2005.