Shih, JL. et al. (Eds.) (2023). Proceedings of the 31st International Conference on Computers in Education. Asia-Pacific Society for Computers in Education

Object Identification Training Support System for Object-Oriented Design with Cooking Recipes

Daiki Maeda[a*], Kota Kunori[a] & Tomoko Kojiri[b]
[a] *Graduate School of Science and Engineering, Kansai University, Japan*
[b]*Faculty of Engineering Science, Kansai University, Japan*
*k554875@kansai-u.ac.jp

Abstract: In object-oriented programming (OOP), each process is accomplished by the interaction of objects. In the first step of designing a system using OOP, programmers need to decompose a given use case into smaller sub-processes and identify objects. However, for novice programmers, it can be difficult to imagine things that are not explicitly stated in the use case, and some may not understand that the process is achieved by the interaction of objects. We have therefore developed a training method that identifies objects by connecting them with things that novice programmers can more easily imagine, such as recipes. Similar to use cases, recipes are a set of processes whose subjects are not clearly mentioned. When we decompose a recipe set into smaller sub-processes made up of kitchenware items, it is possible to recognize these items when imagining the cooking scene. We also developed a system that helps novice programmers to extract the appropriate kitchenware from recipes. Through the experience of rewriting a recipe utilizing the process of cooking, novice programmers are able to recognize that there are subjects hiding in a process description whose subjects are not mentioned, which helps them understand how to identify the hiding objects.
Keywords: object-oriented programming, extraction of objects, recipe, training support system, drawing scene

1. Introduction

Object-oriented programming (OOP) is widely utilized in the software industry (Teif & Hazzan, 2006). In OOP, program behavior is implemented as an interaction between objects. When designing software using OOP, the classes of objects need to be designed in accordance with the requirement specifications (e.g., use case). To design a system so that the degree of coupling between classes is low and the degree of condensation within classes is high, it is necessary to identify functions that can be grouped together and to identify the things that can carry those functions as objects. However, it can be difficult for novice programmers to identify objects from the use case that are not explicitly mentioned, and some do not understand that the process is accomplished by the interaction of objects. Therefore, in extreme cases, they may design a system with a single object called “system” and assign all functions to it.

Many studies have been conducted to help programmers understand the object-oriented concept (Lian, Varoy, & Giacama, 2022; Seng, Yatim, & Hoe, 2018; Dwarika & De Villiers, 2015), some of which specifically focus on the behaviors of the objects (Abidin & Zawai, 2020; Gestwicki & Jayaraman, 2005). Although these studies provide an understanding of the object-oriented concept, they do not support the design process of OOP, which is essentially to identify objects in accordance with the requirement specifications. In order to identify objects, it is necessary to imagine the potential interactions of multiple objects from the given requirement specification.

In the current study, we explore two key research questions:

1) What kind of thinking is necessary to derive adequate objects from the process description in which subjects are not clearly mentioned?
2) What kind of training is appropriate to foster such thinking?

To derive the answers to these questions, we developed a training method and evaluated its effectiveness.

The proposed training method identifies objects by connecting them with things that novice programmers can more easily imagine, such as recipes. Similar to use cases recipes are a set of actions whose subjects are not clearly mentioned. Our method circumvents this difficulty by decomposing the set into smaller sub-processes of kitchenware. For example, “to cook meat” can be rewritten as “a stove heats up a frying pan and the frying pan cooks the meat.” We can recognize the kitchenware items when we imagine the cooking scene, so it is easier to grasp that the recipes are essentially a series of kitchenware interactions and thereby to imagine what kind of kitchenware is involved in the recipe. Our system basically encourages novice programmers to identify kitchenware from the written recipe that they imagine from its scene. The system asks the users to 1) extract the kitchenware from the recipe, 2) rewrite the recipe in accordance with the extracted kitchenware, and 3) draw a picture using the identified kitchenware. This process is repeated until they are able to accurately depict the cooking scene that they imagine from the recipe. Through the experience of rewriting a recipe utilizing the process of kitchenware, novice programmers are able to recognize that there are subjects hiding in the process description whose subjects are not mentioned. In addition, by continuing this activity until they draw the scene that they imagine, they begin to grasp that imagining the scene is necessary for deriving adequate objects.

2. Approach

2.1 Requirement Specification of System Development

A requirement specification is a document that summarizes the functions and characteristics that a system to be developed should have. It consists of a base action series, exemplified action series, and other conditions, and it is established in the requirements definition phase of system development to form the policy for the subsequent development plan. An example of a requirement specification is a use case description, which describes specific interactions between an actor and a system in that scene, as well as related conditions (Technologic Arts Inc., 2009). Since actors are the users of the system, the external hardware, the external systems related to the target system, etc., the use case description is a summary of the requirements from the user's point of view. Therefore, the configuration of the system (e.g., the objects) are not typically mentioned.

Figure 1 shows an example of the basic action series for the use case “withdrawing money from an ATM”.

1. **Select “cash withdrawal”**
2. **Enter your PIN**
3. **Enter the amount of money**
4. **Confirm the amount displayed on the screen and press the confirm button**
5. **Take out the amount of money you wish to withdraw from the ATM withdrawal slot**
6. **Receive cash from the withdrawal slot**

Figure 1. Basic action series for cash withdrawal from an ATM.

2.2 Object Identification Method

Since the basic action series of a use case description does not always mention the subject or target of each action, the programmer will need to derive objects that can be the subjects or targets. If the scene in the use case can be imagined, the identified objects should be in the imagination.

We propose a method for deriving adequate objects by having the programmers draw a picture of the processing scene. Since the adequate objects are those described in the picture, this method encourages programmers to derive subjects and targets from the use case that are the same as what they imagine. By having them draw the picture, they become aware that the objects they add to the image are the objects that they should identify.

2.3 Acquisition of Object Identification Methods Using Recipes

In order to acquire the thinking method that the objects in the scene of the use case specification are the objects to be identified, it is desirable to train the thinking method by a use case whose scene is easy to imagine. In this study, we utilize a cooking recipe as the substitution for the use case.

A recipe is a sequence of actions to take when preparing a specific dish. In this context, the subjects of the actions can be regarded as the kitchenware items, and some data (e.g., ingredients) can be viewed as objects being processed by the kitchenware. If we view an item of kitchenware as a subjective object, the recipe can be viewed as the interaction of multiple kitchenware items. However, in a typical recipe, the kitchenware is not described clearly, if at all. In this respect, we believe that recipes are similar to use cases. If it becomes possible to identify kitchenware from recipes, novice programmers can learn how to identify objects from the use case and thereby successfully identify objects. This is why our method utilizes a recipe, not a software use case, as the training target. We came up with the following steps to train the thinking of object identification.

Step 1. Derive the objects (e.g., subject and target) corresponding to each action of the recipe.
Step 2. Rewrite the action with the derived objects.
Step 3. Draw a picture of each cooking action in the recipe using the identified objects.
Step 4. Check whether the picture in Step 3 satisfies the imagined scene. If not, identify the missing objects and return to Step 2.

As an example of the steps, let's focus on the cooking action "boil potatoes" in Figure 2. If we assume the subject of this process is a "person" and the target is a "potato" (Step 1), the cooking action can be rewritten as "a person boils potatoes" (Step 2). If we draw a picture like the one in Fig. 3 using the extracted objects (Step 3) and find that it is different from the imagined scene, as shown in Fig. 4, we extract the missing objects by examining the differences between the drawn picture (Fig. 3) and our imagination of the scene (Step 4), such as "stove", "fire", "pot", and "water". We then return to Step 2 and rewrite the "boil potatoes" action on the basis of these derived objects as "1. a person turns on a stove, 2. the stove turns on the flame, 3. the flame heats a pot, 4. the pot heats the water, and 5. the (heated) water boils the potatoes". In Step 3, we draw the scenes from these objects and, if we satisfied with the picture, the process ends, indicating that we have derived adequate objects. If not, we go back to Step 4 and look for the missing objects again.

1. **Boil potatoes**
2. **Mash potatoes**
3. **Cut onions**
4. **Cut carrots**
5. **Heat onions**
6. **Heat carrots**
7. **Cut cucumbers**
8. **Cut ham**
9. **Mix mayonnaise**

Figure 2.
Potato salad recipe.

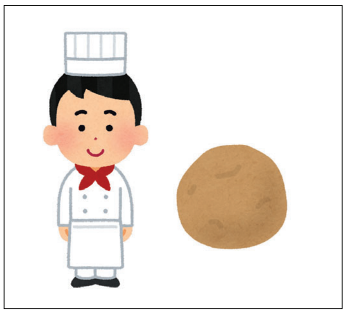

Figure 3.
Cooking scene drawn in accordance with identified objects.

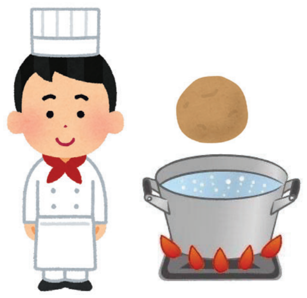

Figure 4.
Imagined cooking scene.

2.4 Overview of Object Identification Training System

The object identification training system is intended to provide a place where users can experience how to identify objects by using recipes, thereby enabling them to acquire the thinking necessary to identify objects from the process. A system configuration diagram is shown in Fig. 5. The system consists of two interfaces and a feedback function, and it utilizes recipes in a recipe database (DB) for the training.

The identification interface is a screen for inputting the objects identified from each action of the recipe. The user inputs the subject and target from the displayed recipe sentence and uses them to rewrite the sentence. The confirmation interface is a screen where the user draws a picture using the objects (subject and target) input to the identification interface to confirm whether adequate objects have been input. After the picture is drawn, the confirmation interface gives a comment to the user asking if there are any lacking objects. If not, it finishes the process. If yes, it activates the identification interface again. The identification interface asks the user to rewrite the recipe sentence using the lacking objects. It then checks whether the written sentences represent the action of the original process, and if not, it gives feedback to modify the sentences again.

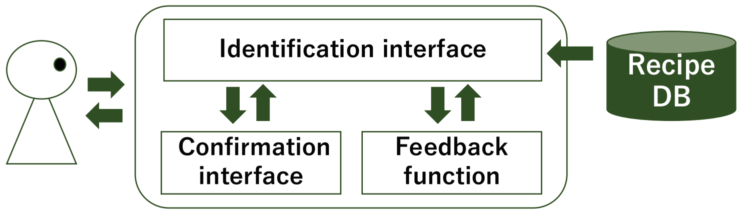

Figure 5. System configuration.

3. Feedback Function

A sentence corresponding to an action in the recipe represents that the subject executes the action and the target is manipulated by the action. If the sentence is rewritten by using several sentences related to sub-actions, it is important that the first subject of the sub-actions, the last target of the sub-actions, and the last action be the same as the original action. The subjects and targets of other actions are intermediary; that is, instead of indicating a direct manipulation of the target by the subject, it means that others are told to manipulate the target. Figure 6 illustrates the relations between the original action and the sub-actions. Suppose that the original action P is a process from subject A to target B. If the action is replaced by sub-actions, the first subject needs to be A, the last target needs to be B, and the last action must be P. Oher actions (a, b, and c) and other objects (X, Y, and Z) might be placed at various positions between them.

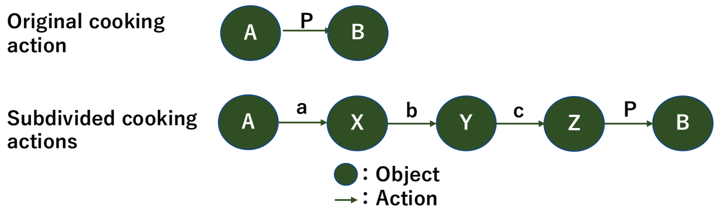

Figure 6. Relations between original action and sub-actions.

The feedback function checks whether the rewritten recipes satisfy such conditions. In addition to the first subject, the last target, and the last action, it needs to check that there are no omissions in the transfer of actions. The actions are connected when the target of the former action and the subject of the latter action are the same. By considering these conditions, the feedback function determines whether the rewritten recipe is the same as the original one by the following steps:

Step 1. Determine the subject and the target of each sub-action in the rewritten recipe.

Step 2. If the previous target and the subsequent subject are the same, eliminate the target/subject and previous action and combine them into one action.

Step 3. If the rewritten recipe is changed to one action, compare its subject, target, and action to the original action.

Here, we show an example with the original recipe “a person boils potatoes” and the rewritten recipe as “1. a person starts up a stove, 2. the stove turns on a flame, 3. the flame heats a pot, and 4. the pot boils the potatoes”. Since the target of 1 and the subject of 2 are the same (i.e., “a stove”), these actions are combined as “a person turns on the fire of a stove". As a result, we have “1. a person turns on the flame, 2. the flame heats a pot, and 3. the pot boils the potatoes”. Since the target of 1 and the subject of 2 are the same in the rewritten recipe, they are combined to create “a person heats a pot”. As a result, "1. a person heats a pot, and 2. a pot boils the potatoes” is created. In the same way, “1. a person boils the potatoes” is generated. Since it is the same as the original recipe, the rewritten recipe is regarded as the same action as the original one.

The feedback function provides feedback in accordance with the result. If the original recipe and the rewritten one are the same, it gives “The re-written recipe is appropriate.” If not, it points out the inappropriateness and suggests how to fix it. There are two possible causes of inappropriateness. The first is that there is an omission in the transfer of processing. For example, if the sub-actions are “1. a person starts up a stove, and 2. a pot boils the potatoes,” some actions might be missing between the two actions, since the target of action 1 and the subject of action 2 are different. In this case, the feedback function shows the user where the cooking steps are not connected and gives a hint if the subject and target are different. The second cause is that the combined action is different from the original one. In this case, the system gives the comment “The action in the original recipe and the sub-actions in the rewritten recipe are not the same”.

4. Prototype System

4.1 Identification Interface

Figure 7 shows an image of the identification interface screen. When a recipe name is selected, the actions in the selected recipe are displayed in the recipe display area. The user needs to input the subject, target, and action extracted from the recipe into the cooking action input area. By selecting a row in the cooking action input area and clicking the “Add Row” button, a new row is added under the selected row. By clicking on each cell of the row, the subject, target, and process can be entered. To delete the row, click on the row to be deleted and select the “Delete” button. The order of the input in the cooking action input area represents the order of the cooking from start to finish. When the “Judgment” button is clicked, the feedback function evaluates whether the input cooking actions are appropriate and displays the result in a message box.

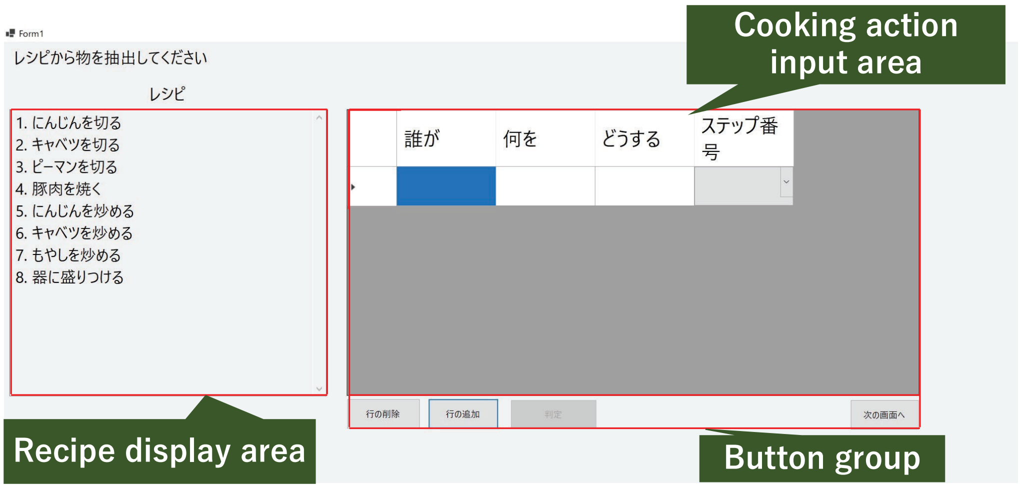

Figure 7. Identification interface.

4.2 Confirmation Interface

The confirmation interface is a screen for creating a picture of the cooking scene for each action using the objects identified in the identification interface. Figure 8 shows an example of the confirmation interface screen. The user can select the action to be depicted in the cooking action display list. When a target action is changed, a new canvas for the action appears in the picture creation area, and the user can draw a picture representing the scene of the action. The picture is drawn by placing the images displayed in the image display area onto the picture creation area. The image display area displays images of objects stored in the system in advance that may appear in the cooking scene. The user can also add their own images by dragging and dropping image files. To draw the picture, a user selects an object from the list of identified objects, clicks on the image representing the object, and then clicks on the “Match

String to Image" button to display the corresponding image. The user can also move, delete, and change the size of the image. When the cooking scenes of all actions are drawn using the identified objects and the "Finish Creation" button is clicked, the system presents a message to consider whether the drawn pictures represent the user's imagination of the scene. The intent of this message is to prompt the user to think about whether there are any missing objects. If there are, the system moves to the identification interface again and lets the user subdivide the cooking action using the missing objects recognized in the confirmation interface. If there are no missing objects, the operation terminates.

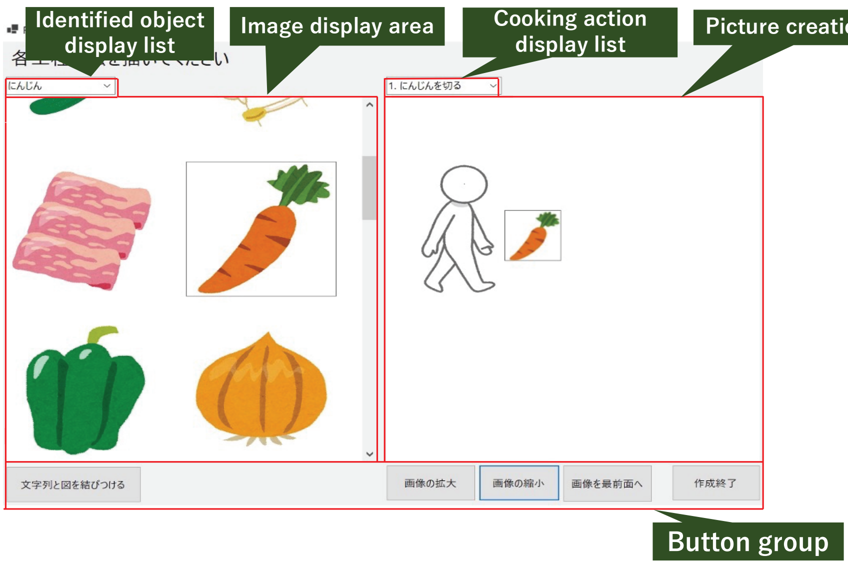

Figure 8. Confirmation Interface

5. Conclusion

We have developed an OOP training system for identifying objects that are responsible for the sub-actions of requirement specifications, such as use case descriptions. The training system introduces a recipe instead of a use case as a training target, which makes it easier for novices to imagine the objects and their interactions in the scene. Our system helps the user to identify valid objects by having them draw a picture of the cooking scene with the identified objects. It also provides a feedback function to check whether a rewritten recipe using the identified objects represents the same actions as the original recipe.

The recipes handled by the system in this study were linearly connected, with only one subject and one target per process. However, there are also actions that have multiple targets. In such cases, some actions are performed in parallel and are therefore not connected linearly. To cope with such recipes, the feedback function needs to be improved so as to check the validity with not only simultaneous actions but also actions occurring separately.

References

Abidin, Z. Z., & Zawai, M. A. (2020). Oop-Ar:Learn Object Oriented Programming Using Augmented Reality. *International Journanl of Mulimedia and Recent Innovation, 2*(1), 60-75.

Dwarika, J., & De Villiers, R. M. (2015). Use of Alice Visual Environment in Teaching and Learning Object-Oriented Programming. *In Proceedings of The 2015 Annual Research Conference on South African Institute of Computer Scientists and Information Technologists*, (pp. 1-10).

Gestwicki, P., & Jayaraman, B. (2005). Methodology and Architecture of JIVE. *In Proceedings of The 2005 ACM Symposium on Software Visualization*, (pp. 95-104).

Lian, V., Varoy, E., & Giacama, N. (2022). Learning Object-Oriented Programming Concepts Through Visual Analogies. *IEEE Transactions on Learning Technologies, 15*(1), 78-92.

Seng, Y. W., Yatim, M. H., & Hoe, T. W. (2018). Learning Object-Oriented Programming Paradigm Via Game-Based Learning Game—Pliot Study. *The International Journal of Multimedia & Its Applications, 10*(6), 181-197.

Technologic Arts Inc. (2009). *Teaching Yourself UML 4th Edition in Japanese.*

Teif, M., & Hazzan, O. (2006). *Partonomy and Taxonomy in Object-Oriented Thinking : Junior High School Students" Perceptions of Object-Oriented Basic Concepts".* Working Group Reports on ITiCSE on Innovation and Technology in Computer Science Education.