AN EDUCATIONAL UNIFIED MODELLING LANGUAGE PROGRAMMING ENVIRONMENT AND ITS TWO CASE STUDIES

Ryoga Maruyama[1], Shinpei Ogata[1], Mizue Kayama[1], Nobuyuki Tachi[2], Takashi Nagai[3] and Naomi Taguchi[4]

[1]*Graduate School of Science & Technology, Shinshu University, Japan*
[2]*Shinshu University, Japan*
[3]*Institute of Technologists, Japan*
[4]*Minowa Junior High School, Japan*

ABSTRACT

This study aims to explore an educational learning environment that supports students to learn conceptual modelling with the unified modelling language (UML). In this study, we call the describing models "UML programming." In this paper, we show an educational UML programming environment for science, technology, engineering, art, and mathematics (STEAM) related subjects (especially for Technology or Engineering) in schools, which are able to apply from elementary school to university. At first, we explain why, what, and how doing the UML programming. In this study, we use a state machine diagram for UML programming. To draw this model, the students just put some states in rectangular shape and transitions in arrow shape. Two types of educational model notations in state machine diagram are introduced. Then, some advanced functions of the SRPS are described. They are an educational model editor, and management of users, learning tasks and models submitted by students. Next, two case studies with the SRPS are shown. One case study is adopted to the summer camp for 5th- and 6th-grade students. The participants were 20 students and were engaged in 4 hours workshop. We used a car-type robot with two DC motors, a one-touch sensor, and one infrared sensor connected to a micro:bit. The other case study is a formal technology class at one Japanese public junior high school. One teacher and five classes worked on UML programming for traffic lights. Each class had 20 9th-grade students. One student at this school was given a traffic light robot with three Light-emitting diode (LED) lights, a one-touch sensor, and one infrared sensor connected to a micro:bit. Finally, on the basis of these two case studies using our environment, we discuss the potential for innovative STEAM education with UML programming.

KEYWORDS

Conceptual Modelling, UML, Model Driven Development, Learning Environment, STEAM

1. INTRODUCTION

Recently, model-driven development (MDD)-based formal classes are focused on some educators (Starrett, 2007, Akayama et al, 2014, Kayama et al. 2015). Students in information communication technology (ICT)-related departments at universities, colleges, and high schools are the primary users. In their modelling classes, they create abstract models using the Unified Modelling Language (UML). One of the learning objectives in some of these classes is conceptual modelling with UML.

The concept of modelling is familiar with computational thinking (Wing, 2006). This way of thinking is similar to the "program-like thinking" proposed by the Japanese government (MEXT, 2018). Designing a robot model that supports human life with UML is mentioned as an example of learning activities in the most recent Japanese national school curriculum for Technology at the secondary school level (MEXT 2017).

Some years ago, educational methods or learning courses related to conceptual modelling had eagerly explored in many educational institutes, academic conferences, and academic journals (Sendall 2003, Börstler 2004, Kramer 2007, Bezivin 2009, Fuller 2010). Some institutes start their conceptual modelling course from graduate school, but other institutes teach conceptual modeling after fundamental programming courses in undergraduate school. We have been applying this educational methodology for the students in

pre university level, from elementary school, junior high school, and high school from 2010. Recently, some international educational project related to computational thinking using modelling (Rottenhofer et al. 2021, Moreno-Leon et al. 2022). In these studies, the researchers, including us focus on graphical models without coding to express conceptual modelling.

This study aims to explore an educational learning environment that supports students to learn conceptual modelling with UML. In this study, we call the describing models "UML programming." In this paper, we show an educational UML programming environment for STEAM-related subjects (for example, Technology or Engineering) in schools, which are from elementary school to university. First, we describe the feature of the MDD method and current MDD tools. Then, we describe our findings regarding the design and implementation of an educational learning environment for UML programming with MDD. Finally, on the basis of some case studies using our environment, we discuss the potential for innovative STEAM education with UML programming.

2. UML PROGRAMMING WITH MDD

2.1 Overview of MDD

The MDD method is one approach to realize model-driven engineering (MDE). MDE is a software development methodology that focuses on developing and deploying domain methods (abstract representations of the knowledge and activities that govern a specific application domain) rather than computing concepts. In the MDD method, models are usually expressed with the graphical elements, such as UML. MDD is thought to be appropriate for education, such as programming education in primary, secondary, and higher education, as well as modelling education for beginners. MDD does not necessitate that the user understands the complex syntax of programming languages. Consequently, even if users are unfamiliar with programming, they can learn the fundamentals of system development.

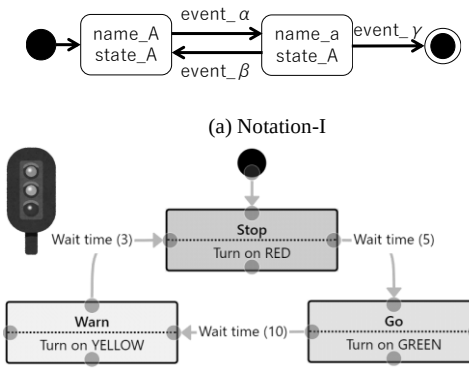
2.2 UML Programming and its Notations

The model diagrams discussed in this paper are described in simple UML notation. In this paper, we use state machine diagrams to represent the modeling target's life cycle. This diagram is made up of two parts: states and behavioral transitions. An initial state, a final state, and a simple state are the three types of states. A small, filled circle represents the initial state. Each diagram must include this state at least once. A final state is represented by a circle encircling a smaller circle. There may be zero or more final states in one diagram. A simple state is shown as a rectangle with rounded corners, and the state name and short description for a state are written inside the rectangle. A behavioral transition is a directed relationship between a source state and a target state. Transitions in our notation have an event name as a required element and an action or trigger name as an optional element. There is no event name for the incoming transition from the initial state.

In this paper, we introduce two types of state machine notation(Maruyama, 2022). Notation-I use named states and transitions [Figure 1(a)]. A state machine may contain some states with the same state name as each name if named states are used. Figure 1(b) depicts an example of a state machine diagram in Notation-I. This is a traffic light diagram. This model denotes the activities listed below. The state of light begins with an initial state and progresses to a "Turn on RED" state. "Stop" is a succinct description of this state. After 5 s passed, this light changes its state to the next. The next state is "Turn on GREEN," which means "Go." Then, after 10 s passed, the light changes its state to the next. The next state is "Turn on YELLOW." After 3 s passed, the state of light returns to the "Turn on RED.".

Figure 2 shows the other notation. Notation-II can use named states, transitions, and messaging [Figure 2(a)]. Sending messages is represented by an action in the state machine diagram for this notation, and receiving messages is represented by a trigger. Some state machines can communicate with one another cooperatively and synchronously by sending and receiving messages. Figure 2(b) shows an example of a state machine diagram using Notation-II. This diagram is a pedestrian traffic light that interacts with a vehicle traffic light that belongs to group 2. The pink-colored rectangle represents a group ID. This model

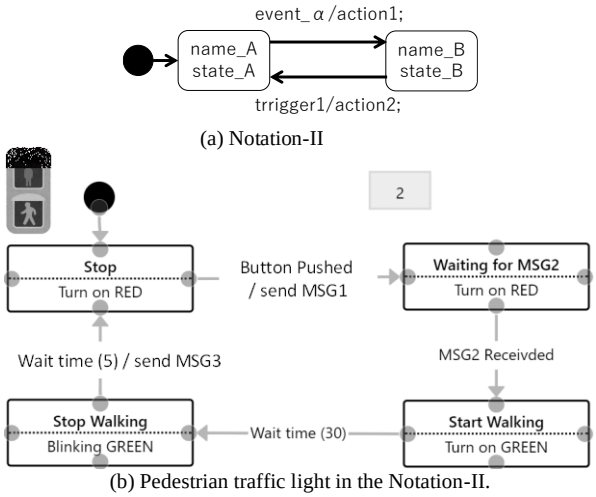
denotes the activities listed below. The state of light begins with an initial state and progresses to a "Turn on RED" state. If a walker pushes a pedestrian traffic light button ("Button Pushed" as an event), this model sends a message to the vehicle traffic light in group 2 ("send MSG1" as an action). At the same time, the state of this pedestrian traffic light changed to the next state, named "Waiting for MSG2." The color of the traffic light keeps the red. Then, if this traffic light received the message from the vehicle traffic light ("MSG2 Received" as a trigger), the state changes to the next state, named "Start Walking." The color of the traffic light shifted to green. After 30 s passed, the state of the traffic light changes to the next state, named "Stop Walking." The traffic light's color changed to blinking green. Furthermore, after 5 s, the traffic light sends a message with a different ID than the previous message sent to the vehicle traffic light ("send MSG3" as an action). The traffic light then changed to the next state, "Stop," and the color changed to red.



(a) Notation-I

(b) Traffic light model in the Notation-I

Figure 1. Notation-I and an example of an UML program in Notation-I



(a) Notation-II

(b) Pedestrian traffic light in the Notation-II.

Figure 2. Notation-II and an example of an UML program in Notation-II

This representation is useful to raise the thinking ability of students. Current Japanese school curriculum guidelines said that thinking ability is necessary for the students. In Taizan's thinking tool (Taizen, 2014), our notation is included in the "arrow and enclosure" method. Consequently, our notation will be suitable for use at any level of education.

2.3 Domain Specific Language

A DSL is "a computer programming language of limited expressiveness focused on a particular domain." Using a DSL, developers can express their implementation ideas in terms that their customers can understand and discuss. In this paper, we introduce some sets of vocabulary as DSL that represent a state action and a transition event. For example, "Turn on RED," "Turn on GREEN," and "Turn on YELLOW" are the DSL of the state; "Wait time" and "Button Pushed" are the DSL of the transition in Figures 1 and 2.

As for DSL, in this study, we can use not only natural language but also programming language or script. Consequently, we can use the DSL to control the difficulty of each laboratory task. Learners are given a set of DSL by their teacher and engage in laboratory tasks to draw models using DSL.

3. FEATURES OF MDD TOOLS

3.1 MDD with xtUML

For our learning environment, the concept of executable UML (xtUML) (xtUML.org, 2012) and MDD approach are applied. The programming language in our environment combines a subset of the UML

graphical notation (in this case, state machine diagrams) with executable semantics and timing rules. We include xtUML and MDD in our teaching method because we expect our students to be able to evaluate their models on the basis of observations of the target device's behavior without worrying about grammatical errors or exact programming language expression. The model created by the students is platform-independent. Platform dependencies emerge as a result of source code generation. In the source code of the model compiler, which is described in a later section, some specific libraries for the target device must be included. The source code compiler will be activated after the source code has been generated. We must prepare each compile tool for each target device in this source code compiler.

According to the concept of MDD with xtUML, the learning flow is following. The teacher defines practical tasks and some sets of DSL for each task. After being given tasks, students analyze the given requirements (requirement analysis) and then design a system using UML modeling diagrams (making models) with the model editor. Following the creation of models, a model compiler generates source code from the models (translating models to code). The generated code is then compiled by a source code compiler into an executable file (compiling code to executable binary data). Learners observe the behavior of the target device after transferring the executable code to it to determine whether the requirements have been met.

3.2 Comparison of MDD Tools

Many MDD tools are provided by commercial vendors. In this chapter, we provide an overview of each MDD tool and discuss the issues in educational use for beginners. Bridgepoint is an open-source MDD tool provided by xtUML.org (xtUML.org, 2012). By describing transformation rules in BridgePoint notation, users can generate executable code for a variety of platforms. BridgePoint can be used as a general-purpose MDD tool because it has a wide range of functions that are appropriate for professional use. Consequently, we believe that it is difficult for a beginner to master this tool in a short period of time. Astah is a UML modeling tool provided by Change Vision Inc. (Change Vision, 2009). Astah has special APIs for users to develop their own plug-ins. Users, for example, can generate source code from the model in astah using the m2t plug-in. Nevertheless, because there are currently no plug-ins with DSL-like functions, teachers are unable to control the task's difficulty level. Clooca is an MDD tool provided by Technical Rockstars Inc. (Hiya et al. 2013). This tool can use metamodels and template functions to generate source code from models. Because it can define DSLs using metamodels, it is an ideal modeling environment for educational use. Since 2012 (Kayama 2015), we have been running S-clooca, a UML programming environment that combines clooca and a compilation server that we developed.

Since 2018, our product has also been applied to formal junior high school technology classes (Maruyama, 2022). During these practices, we got some educational problems with S-clooca. For example, in junior high schools, all students in the class work on some common tasks. Teachers want to assess students' performance as efficiently as possible. Clooca, conversely, does not allow teachers to review their students' model diagrams. Furthermore, the UX provided to programming beginners was limited because it was difficult to rewrite the diagrams or difficult to review their diagrams.

4. OUR PROPOSAL: SRPS

4.1 Overview

In this paper, we propose SRPS on the basis of the educational problems we have found with 3 years of UML programming practices with S-clooca in junior high school. The features of SRPS are as follows:

- Definition of DSLs according to the tasks
- Management of models created by learners
- Suitable UX for the beginners of programming

SRPS comprises a web application, a compilation server, and a target device. The web application is the core of SRPS. The compilation server is the same as S-clooca. The target device is a micro:bit. The SRPS web application consists of a model editor, a model compiler, and a content management module. In SRPS, we designed the UX for beginners of programming learners and enhanced the management functions for teachers to create task topics and created models by learners.

The model editor of SRPS has a notation checking function, and the source code is generated using the model compiler only when there is no defect in the notation checking function. The model compiler of SRPS generates C++ source code from the model diagram described by the model editor. On the server, the generated source code is compiled, and the executable code is downloaded using the learner's browser. The learner confirms the behaviors described in the model diagram by transferring this executable code to the target device.

4.2 Functions of SRPS

Users of SRPS are categorized into three types of roles: administrators, teachers, and learners. SRPS provides functions according to the roles of the users. In the following, we describe the functions for learners, administrators, and teachers.

4.2.1 Model Editor

Figure 3 shows the model editor is a function for learners. The interface of the model editor. The available behaviors and events are defined by the DSL for each task. By selecting a floating-point value in this model editor, you can access an interactive tutorial (the red double circle icon on the right side of Figure 3). The teacher can avoid repeating the explanation of how to use the model editor by allowing students to check the basic usage on their own. SRPS records the user's operations to deal with erroneous operations by the user (the Undo and Redo functions). Even when describing a single model diagram with multiple class days, it is possible to go back through the days and repeat operations. The model editor's diagram element includes a "memo" element. The teacher has access to this memo. The teacher can also write memos on the model diagrams of the students. The Model Editor can be used to check for notational errors in model diagrams. A dialog alerts you to notation errors, and the related element of the model diagram in the editor turns red. Only model diagrams in which all notational errors have been corrected can run the model compiler function. The model diagram notation checker assists students in correcting errors in their model diagrams. The diagrams of the models can be saved and submitted. The model diagram saving function in SRPS saves the model diagram for each learner and task. The model diagram submission function saves model diagrams for teacher evaluation. Model diagrams can be submitted as many times as desired.

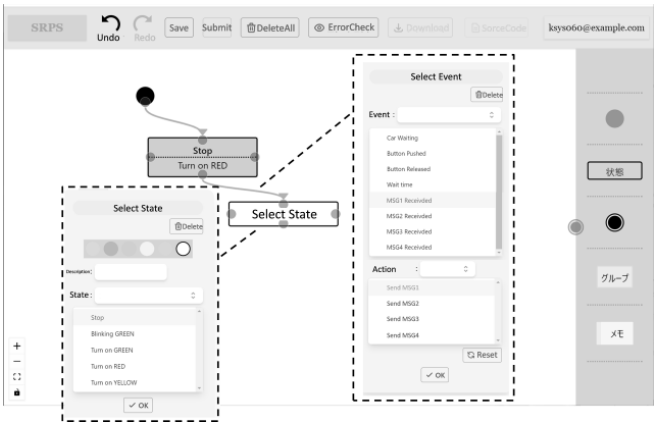
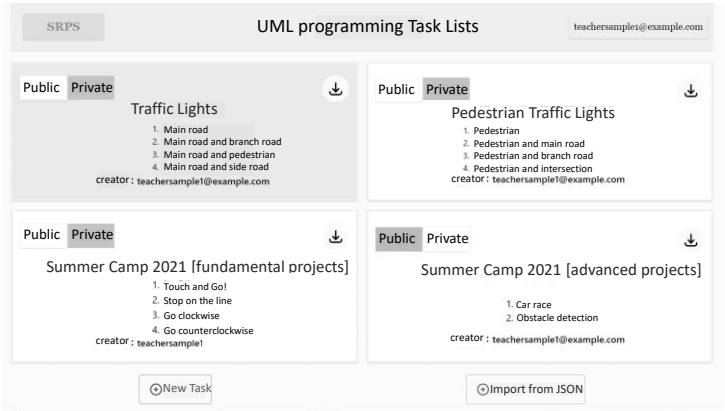


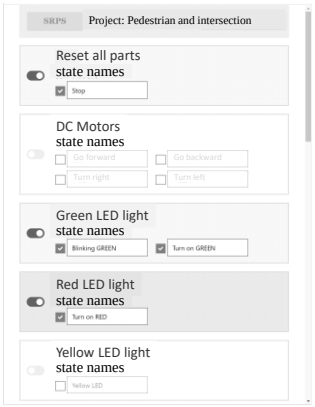
Figure 3. Model editor interface of the SPRS

4.2.2 Management of User

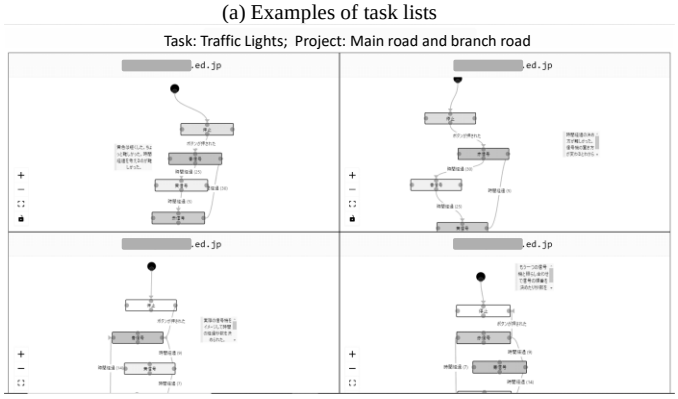
SRPS users are distinguished by their e-mail address, password, group (e.g., equivalent to a class in school), and role. The role is related to the permissions to use the SRPS functions. A teacher can be a member of several groups. Teachers' functions include managing students in their group, creating learning tasks, assigning learning tasks to groups, and reviewing model diagrams submitted by students. A student is a member of one group, and their functions are model diagram description and submission. The administrator can use the same functions as the teacher for all groups and all users. The registration of users is a function for administrators and teachers. The administrator can register users in all roles. A teacher can register a group that he manages and then register students who belong to that group. A teacher can also register teachers who manage the same group.



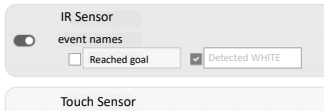
(a) Examples of task lists



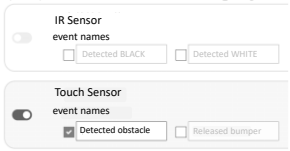
(b) An example of DSL definition (state)



(d) An example of list view for diagrams summited by students



(c-1) DLS for “Car race” project



(c-2) DSL for “Obstacle detection” project

(c) Vocabularies for the DSL of event

Figure 4. Examples of the SPRS interface

4.2.3 Management of Learning Tasks

A learning task (ex: the name of a learning topic such as “traffic lights” or an event name such as “summer camp”) is organized by several projects [see Figure 4(a)]. A project is a type of assignment that is given to students. Each learning task has the option of being public or private. When set to public, the learning task is accessible to all SRPS-registered teachers. The groups that are available for each learning task are determined by the teacher or administrator. The learning tasks are then made available to students in that group. The DSL definition of the learning project is shown in Figures 4(b) and (c). Select the actions (rectangles) and events (rounded rectangles) to be used in that project, and give them names that reflect the contents of the project. Figure 4(c) shows an example of using the infrared (IR) sensor and the touch sensor in different projects. In the race project, the IR sensor’s black color detection is used in the vocabulary “Reached goal,” and the touch sensor’s press event is used in the vocabulary “Pushed start button.” The IR sensor, on the other hand, is not used in the obstacle detection project, and the event of pressing the touch sensor is used in the vocabulary of “Detected obstacle.”

4.2.4 Management of Models Submitted by Students

The administrator and teacher can review the model diagrams submitted by the student. In SRPS, the submitted diagrams are managed by assignment or by the user. SRPS keeps track of all submissions. When checking submissions, there is no need to refer to the submission history because the most recent submissions are always displayed. The submitted diagrams can be viewed in two ways: list view and individual view. Figure 4 shows an example of a list view (d). When reviewing all of a group’s submissions for a specific assignment or all of a specific student’s submissions, the list view is appropriate. In the case of checking the submission history of a particular student for a particular assignment or checking a particular submission in detail, the individual view is appropriate.

5. DISCUSSION

In this chapter, we discuss the feasibility of SRPS on the basis of two different use cases.

5.1 Case Study-I: Summer Camp for 5th- and 6th-Grade Students

Using SRPS, we conducted a 4-hour in-person workshop for 20 participants (female, 50%) in 5th and 6th grades in August 2021. The theme of the workshop was the "collaborative car controlling." They used a car-type robot with two DC motors, a one-touch sensor, and one infrared sensor connected to a micro:bit, and one PC for the SRPS per a participant. One teaching assistant (undergraduate and graduate students) supported two participants during this workshop. Four tasks were assigned as fundamental projects during the first 2 hours of the workshop. Students learned how to use the touch sensor, how to read the driving course using the infrared sensor, and how to trace a line with complex curves in these tasks. These projects made use of Notation-I. All students completed their tasks with a teaching assistant. In the latter 2 hours of the workshop, the students were given a race project and an obstacle detection project to work on in groups of four. Notation-II, which includes sending and receiving messages, was used in these projects. Both Notation-I and Notation-II were easily understood by all students.

It was their first time to use UML and MDD. At the beginning of the workshop, participants understood how to use the SRPS model editor with only 5 min explanation. All participants completed all the projects given to them using the mouse to select diagram elements and the error checking function. This suggests that SRPS was able to provide a UX that could be used by 5th- and 6th-grade students.

5.2 Case Study-II: Formal Technology Classes for 9th-Grade Students

One Japanese junior high school has been using our UML programming environment since 2018. From October 2021, SRPS has started its practical usage in the formal Technology classed in this school. One teacher and five classes work on UML programming for traffic lights in Notation-I and Notation-II in-person. Each class has 20 9th-grade students (50% of whom are female). One student at this school is given a traffic light robot with three LED lights, a one-touch sensor, and one infrared sensor connected to a micro:bit, and one desktop PC per a student. These classed were All students' first time to know and use UML and MDD.

In the first lesson, in the first 30 min, students learned notation of the state machine diagram and the usage of the SRPS. They discussed the timing of color changes in traffic lights for vehicles in the last 20 min. They programmed two types of traffic lights from the second lesson to the third lesson: main road traffic lights and intersection traffic lights. Students were asked to design on their own using Notation-I. Each student had one computer and one robot. In the fourth and fifth lessons, they programmed the pedestrian traffic light, which works in tandem with the vehicle traffic light. They were asked to make a program in a group of two or four students in Notation-II.

A Technology teacher has set whole learning tasks and projects using SRPS. He defined whether or not the predefined states and events should be used, as well as the names of the states and events for each project. After each lesson, the teacher reviewed all submitted diagrams and provided brief comments for each one. This teacher registered all his students by himself or herself to the SRPS.

5.3 Possibility for STEAM Education

We got useful outcomes of these use cases as follows: 1) The MDD with the xtUML method expands the thinking abilities of our learners with the interest of learning. 2) Students master the operation of the SRPS and transfer executable code to the robot in quite short time. 3) The notation error checking function of the SRPS was useful for self-assessment and self-correction of the model notation. And 4) Management functions for the user, tasks, submitted diagrams are useful and easy to use for the teacher.

The MDD-based UML programming shows a great educational possibility for STEAM-related subjects, especially for Technology, at any level of school. The learners can design and make a program for their problem-solving using this method.

6. CONCLUSION

In this paper, we proposed an educational UML programming environment with user/task/diagram management functions for STEAM-related subjects. Based on the two types of feasibility studies for 5th and 6th grade student group and 9th grade student groups, we found that UML programming has great possibility for the STEAM education. These two use cases have been conducted continuously in-person in 2022.

In the next step, the effectiveness of UML programming about creativity or abstract ability will be evaluated on the basis of practical case studies.

ACKNOWLEDGEMENT

This work was supported by JSPS KAKENHI Grant Numbers JP22H03702,JP20K03146,JP16H03074.

REFERENCES

Akayama, S., Hisazumi, K., Hiya, S, et al. (2014). An Empirical Study of the Usage of Model-Driven Development Tool for an Object-Oriented Modeling Education, *In Transaction of IPSJ*, Vol.55, No.1, pp.72-84.

Bezivin,J., France,R., Gogola,M., et al. (2009). Teaching Modeling: Why, When, What?, *Proceeding of 12th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, pp.55-62.

Börstler,J., Michiels, I. & Fjuk,A. (2004). ECOOP 2004 Workshop Report: Eighth Workshop on Pedagogies and Tools for the Teaching and Learning Object-Oriented Concepts, *ECOOP 2004 Workshop Reader*, pp.36-48.

Fuller,R.M., Murthy,U. Schafer,B.A. (2010). The effects of data model representation method on task performance, *In Information & Management*, Vol.47, No.4, pp.208-218.

Hiya, S., Hisazumi, K., Fukuda, et al. (2013). clooca: Web based tool for Domain Specific Modeling, *Proceeding of 16th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, pp.31-35.

Kayama, M., Ogata, S., Nagai, T., et al. (2015). Effectiveness of Model-Driven Development in conceptual modeling education for university freshmen, *Proceeding of 6th IEEE Global Engineering Education Conference*, pp.274-282.

Kramer,J. (2007). Is Abstraction The Key To Computing?, *In CACM*, Vol.50, No.4,pp.37-42.

MEXT. 2018. Elementary school programming education guide (first edition), http://www.mext.go.jp/a_menu/shotou/zyouhou/detail/1403162.htm.

MEXT. (2017). Japanese national school curriculum guideline for the Technology and Home Economics in junior high school, http://www.mext.go.jp/component/a_menu/education/micro_detail/__icsFiles/afieldfile/2019/03/18/1387018_009.pdf

Maruyama, R., Kayama, M., Nagai, T., et al. (2022). Practical UML Programming based on the Executable UML Method at Secondary School Students, *Proceedings of 13th IEEE Global Engineering Education Conference*, 10.1109/EDUCON52537.2022.9766752.

Moreno-León, J., Chushig C., Robles, G. (2022). MaS:Modeling At School –On the Benefits and Skill Development of the Use of Modeling Diagrams at School, *Proceedings of 13th IEEE Global Engineering Education Conference*, 10.1109/EDUCON52537.2022.9766646.

Starrett, C. (2007). Teaching UML Modeling Before Programming at the High School Level, *Proceeding of 7th IEEE ICALT*, pp.713-714.

Rottenhofer,M., Sabitzer, B., Rankin T. (2021). Developing Computational Thinking Skills Through Modeling in Language Lessons, *In Open Education Studies*, Vol.3, No.1, pp17-25.

Sendall,S. (2003). Model Transformation: The Heart and Soul of Model Driven Software Development, *In IEEE SOFTWARE*, Vol.20, No.5, pp.42-45.

Taizan, Y., Kojima, A., Kurokami, H. (2014). A Study of Subject-Common Thinking Skill for Systematic Information Education: From Analysis of the Government Curriculum Guidelines and Commentary to it, *In Japan Journal of Educational Technology*, Vol.37, No.4, pp.375-386.

xtUML.org. (2012). https://xtuml.org/

Change Vision. (2009), astah. https://astah.change-vision.com/.

Wing, J.M. (2006). Computational Thinking. *In Communication of ACM*, Vol.49, No.3, pp. 33-35.