X World Conference on Computers in Education
July 2-5, 2013; Toruń, Poland

Multiplicity Doctor: A Diagnosis Tool for Clarity and Consistency of the Multiplicity in a Static UML Model

Yoshiaki Matsuzawa, matsuzawa@*inf.shizuoka.ac.jp*
Faculty of Informatics, Shizuoka University

Kotaro Nozawa, nozawa@*sakailab.info*
Graduate School of Informatics, Shizuoka University

Yasuhiro Noguchi, noguchi@*inf.shizuoka.ac.jp*
Faculty of Informatics, Shizuoka University

Sanshiro Sakai, sakai@*inf.shizuoka.ac.jp*
Faculty of Informatics, Shizuoka University

Abstract
We have developed a system that diagnoses issues of clarity and consistency between a class diagram and instance diagrams. Our system helps beginners create an understandable model for Object-oriented modeling education. The system checks a given diagram to ensure that it is mutually consistent in multiplicity and that the multiplicity is unambiguous. We conducted a control experiment with 13 students in which only the experimental group was allowed to use the proposed system. The results of the experiment showed that the scores for the experimental group were higher than the control group, whereas the average times for finishing tasks were similar between the groups. We also confirmed by a qualitative analysis of students' recorded procedures that the proposed functions for checking both consistency and clarity successfully supported students in the experimental group.

Keywords
modeling education, diagnosis, object diagram, inconsistency, clarity, multiplicity

INTRODUCTION
Modeling education is becoming increasingly important because recent developers are using higher-order programming languages. Model-driven architecture (MDA) or model-driven development (MDD) environments can now be used by developers in the practical development of information systems. Object-oriented modeling skills involving Unified Modeling Language (UML) are required as a basic skill for software development, although such skills are often not acquired in traditional programming training.

In this paper, we chose to use the terms "model" and "modeling" for the structure model represented by UML class diagrams and object diagrams. We selected these terms because the structure model is located in the lowest layer of the semantics architecture of UML. Furthermore, it provides a basic semantic infrastructure to other behavioral models.

In structure modeling, a modeler creates class diagrams that represent the real world by re-interpreting it using aspects of the object-oriented paradigm. Hence, one of the difficulties in teaching modeling is that we cannot create a quiz that has one correct answer. The traditional model of testing is not effective for this type of learner, because solving quizzes does not effectively measure the learner's understanding. The only way to evaluate models written by leaners is to evaluate whether the model can be used to successfully communicate the system to a separate unbiased individual.

We believe the only way to teach such skills is to have teachers conduct model reviews. Some research proposes teaching modeling skills by peer reviewing and collaborative learning styles; however, in our experience, peer reviews between novice modelers produce no productive discussion to improve the models. The reviews by novice modelers stay at the superficial level. Many students in the classroom cannot accurately communicate with each other using UML.

In this paper, we propose a system to support students in learning how to create an accurate structure model using class and object diagrams. The system can work on a UML modeling tool, and check the model's consistency and clarity at the UML semantic level. Teachers can focus on the higher-level review of student models if the models have already been checked by the system.

The remainder of this paper is organized as follows. We discuss works related to our research in Section 2. We introduce our system in Section 3. We describe the method of the experimental study in Section 4. The results of an experimental study are given in Section 5. Section 6 concludes with a discussion of the results and limitations of the study.

RELATED WORK

Object-oriented modeling education

Our research builds on the following object-oriented modeling education research.

Kuznaiarz et al. (2006) summarized seven best practices for designing Object-Oriented Software Design (OOSD) education based on their five years experiences. They concluded by emphasizing the importance of two of the seven best practices. One is the consistency management between all artifacts in the entire process. The other is to let learners have at least two iterations of the project to have experiences in both the pedagogical purpose project and the concerning realization project. The aim of our tool is to support both of these practices as follows: (1) support consistency between diagrams; and (2) develop student skills in reading class diagrams that support the second iteration of the given project.

Although numerous tools for consistency management have been proposed in the past decade, there are few tools for students that diagnosis problems in class and object diagrams (Lucas et al., 2009). Even if we include tools that support communication diagrams instead of object diagrams, there is no tool to manage multiplicity (Hausmann et al., 2002; Laleau et al., 2007; Lucas et al., 2005; Paige et al., 2002; Paige et al., 2007).

Very few researchers and authors of modeling textbooks proposed a method to use object diagrams for reading class diagrams. Jonouchi (2006) argued that to treat the reading of class diagrams as a first step in the classroom requires too much abstract thinking; he therefore proposed the use of object diagrams. Engels et al. (2006) proposed that "teaching UML is teaching abstraction" and they recommended to use object diagrams first from a video that contains the target of the model, then to create the class diagram by abstracting the object diagrams. Kodama (2008) introduced a method to check errors in class diagrams by creating object diagrams. Chiorean et al. (2011) argued that using snapshots would help better understand and improve the requirements and model. They argued the snapshots could be used to check whether the informally described requirements are fulfilled or not. Hence, the idea to use object diagrams in education is not new; however, it seems to not be a widely accepted idea, because there are modeling textbooks for beginners that do not mention object diagrams at all (e.g. Steavens & Pooley, 2000).

Model evaluation support for students

Hasker et al. (2011) proposed a system to find defects in class and use case diagrams for education (Hasker & Rowe, 2011; Hasker, 2011). Hasker et al identified typical defects that beginners embed in their model, such as "improper multiplicities for composition," "a composited class owned by two classes," and "reversed generalization." Their proposed UMLInt analyzes a class diagram to detect such defects. An improved version of their system called UMLGRADER (Hasker, 2011) has also been proposed. One goal of the improved system was to present more concrete advice by focusing on typical defects. Similar research was done by Auxepaules et al. (2008) in which they proposed a method to detect the difference between two class diagrams. The system was designed for students and compared student class diagrams to expert ones.

The problem with the above research is that it assumed a model of providing "answer diagrams by teachers" in the classroom. We believe requiring the "correct answer" is not appropriate in teaching modeling. The answers of modeling tasks are typically rich in variety. Hence, we have attempted to develop an evaluation method that focuses on the consistency and clarity of the class and object diagrams.

CASE tools designed for learners

There are many CASE tools designed for students. Auer et al. (2003) proposed UMLet and argued the tool (1) can run anywhere because it is written in Java, (2) has an intuitive user interface, and (3) can output the model in generalized form. Scott et al. (2005) proposed minimUML in which functions are limited for beginners. They limited their focus not only to class diagrams, but also to basic elements, such as class, interface, and association. As our research focuses on supporting the evaluation of the semantic level of the UML model, it is fundamentally different than the research described above.

Dranidis et al. (2007) proposed and evaluated a tool named StudentUML as a CASE tool for UML education (Dranidis, 2007; Ramollari & Dranidis, 2007). StudentUML provides some functions for checking consistency between class and sequence diagrams. Our approach is similar to StudentUML, though our goal is to address some of the limitations of the tool. One of these limitations is that the tool does not support object diagrams, meaning it lacks a function to check the multiplicity in a class diagram. Another limitation is the lack of clarity management for consistency checking; note that clarity management is discussed in the next section. Hence, the goal of our research is to propose an improved version of this consistency management system.

MULTIPLICITY DOCTOR

Quality of the model by clarity and consistency

The system we developed is called "Multiplicity Doctor" and is designed for students to create static UML model consisting of a class diagram and several object diagrams. The tool works within a CASE tool, and provides students with the ability to create diagrams that are "clear" and "consistent" in multiplicity between diagrams.

Fulfilling both "clarity" and "consistency" is an important concept in our research. Our model is shown in Figure 1. For multiplicity consistency management, students have one strategy to keep their models consistent between class and object diagrams. Their strategy is to set all multiplicities to "zero to many (0..*)"; by doing so, all the patterns of links in the object diagrams are consistent with the class diagram; however, such a model is ambiguous to the point of being a "meaningless model," as shown at the bottom-right of Figure 1.

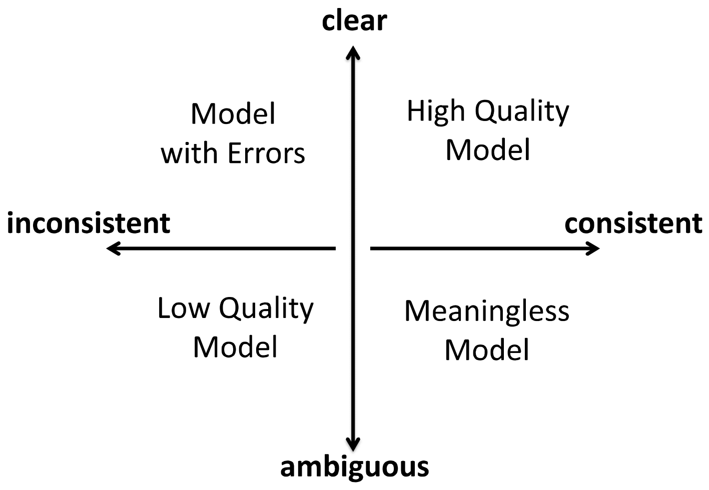
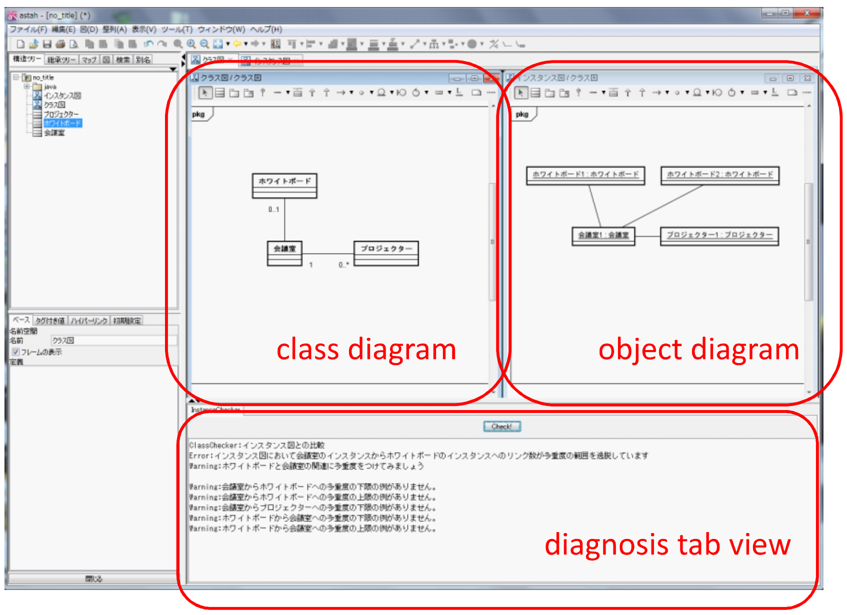


Figure 1: Diagnosis model of clarity and consistency

Hence, the system diagnoses issues with both clarity and consistency in student models, showing the results in the CASE tool. Students are expected to check the advice and revise the model to improve it, ultimately achieving the “high quality model” goal shown in Figure 1.

System user interface

A screenshot of our system is shown in Figure 2. The system was developed as a plugin for “Astah*,” one of the commercial CASE tools produced by ChangeVision. We added the “Diagnosis” button on the diagnosis tab view, which is located at the bottom of the window. When the button is clicked, all the diagrams in the project are set as a target of the diagnosis, and results are shown in the rest of the diagnosis tab view. Students can launch the function whenever they want to use it.



```
ClassChecker:インスタンス図との比較
Error:インスタンス図において会議室のインスタンスからホワイトボードのインスタンスへのリンク数が多重度の範囲を逸脱しています
Warning:ホワイトボードと会議室の関連に多重度をつけてみましょう

Warning:会議室からホワイトボードへの多重度の下限の例がありません。
Warning:会議室からホワイトボードへの多重度の上限の例がありません。
Warning:会議室からプロジェクターへの多重度の下限の例がありません。
Warning:ホワイトボードから会議室への多重度の下限の例がありません。
Warning:ホワイトボードから会議室への多重度の上限の例がありません。
```

Figure 2: User interface

Diagnostic criteria

1. Diagnostic criteria for consistency

The system checks consistency between a multiplicity description in the class diagram and the number of links in the object diagrams. An inconsistent definition means the number of links in an object diagram is out of the range of the multiplicity described in the class diagram. An example of an inconsistency is shown in Figure 3.

Although the multiplicity of class B from class A is 1 in the class diagram, the instance a1, whose class is class A, has 2 links for instance b1 and b2, whose classes are both class B. In this case, the system shows the following message:

- In instance diagram X, the number of links from instances of A to instances of B is out of the multiplicity (1..1) from class A to B defined in class diagram Y.

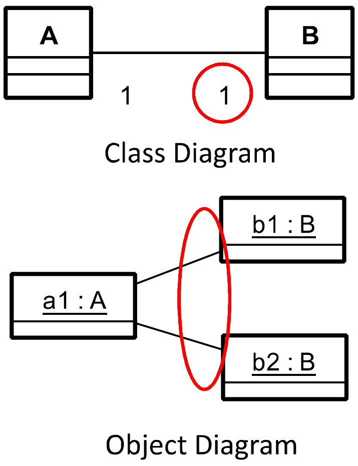


Figure 3: Example of an inconsistency between multiplicity and the number of links

In response, students should either change the number of links in the object diagram or correct the multiplicity (e.g. 1..2).

2. Diagnostic criteria for clarity

To make a diagnosis for clarity, the system requires “sufficient patterns” of examples in object diagrams to fully explain a class diagram. The system checks whether sufficient patterns have been given in the input model; otherwise, it shows warning messages to students. More specifically, covering the “sufficient patterns” means to cover at least the examples for the lowest case and the highest case of multiplicity. When the highest case is many (*), the system requires an example of more than two instances.

An example of the insufficient case is shown in Figure 4. Although the multiplicity of class B from class A is one-to-many (0..*) in the class diagram, the number of links between classes A and B in the object diagrams are 1 (i.e. in the left case of Figure 4) and 2 (in the right case of Figure 4). The example consists of two links and is detected as the highest case for many (*), but there is no example for the lowest case. Therefore, the system shows the following advice:

- There is no example for the lower case in multiplicity (0..*) from class A to class B defined in class diagram Y.

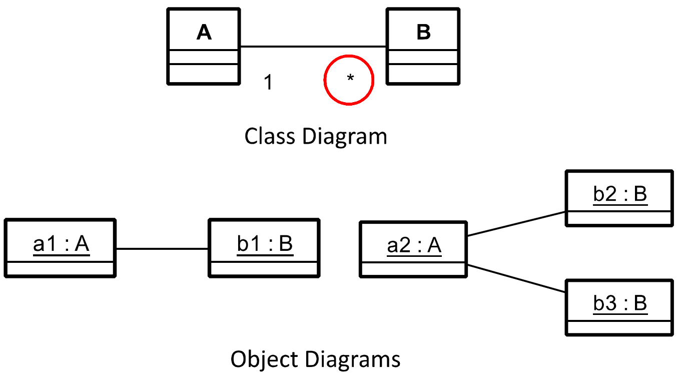


Figure 4: Example of insufficient instance examples in the object diagrams

In response, students should either add the required example or change the multiplicity to the clearer and more specific form (e.g. 1..*).

EXPERIMENTAL STUDY METHOD

Tasks

Two kinds of tasks named “Task A: identifying multiplicity” and “Task B: creating a static model” were created to evaluate the system. A description of each task is given in the following subsections.

1. Task A: identifying multiplicity

The aim of Task A is to verify the effects of checking multiplicity. Inputs of Task A are a class diagram without multiplicity and three object diagrams. Students are required to identify the sufficient multiplicities in the class diagram, keeping the consistency and clarity of the whole model.

An example of Task A is shown as follows:
<Task> Read the model represented in Figure 5 and identify the multiplicities of the class diagram. The model represents the management system of convenience stores. There is no limit for the number of staff for a store.

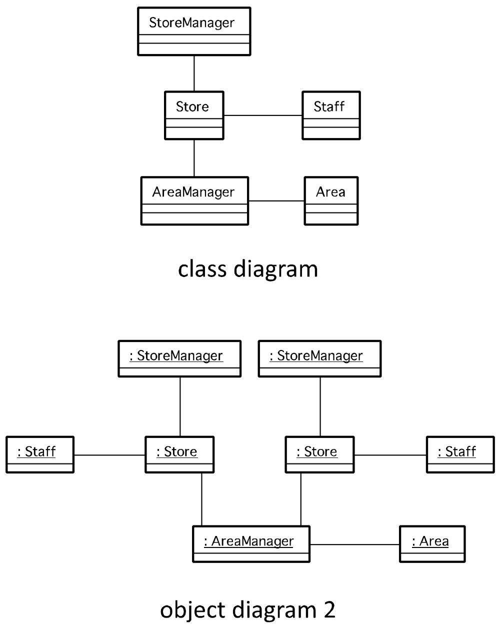


Figure 5: Example of Task A: identifying multiplicity

2. Task B: creating a static model

The aim of Task B is to verify the effects of checking multiplicity in the context of creating a comprehensive model. Students are required to create a static model by class and object diagrams that represent a model described in natural language. An input description is composed of the specification descriptions and scenes. The specification descriptions are written in an abstract way to create a class diagram. Scenes are concrete descriptions and each represents a snapshot of the model to create object diagrams.

An example of Task B is shown as follows:
<Task> Create a static model that represents the system described as follows. Next, create three object diagrams, each of them representing one of the scenes given in the description. This is a management system for intensive course registration for students in a university.

Specification:
(A) There are 4 intensive courses.
(B) There is no limit on the number of registrations a student can have.
(C) Each intensive course has one lecturer.
(D) A course will be canceled if the number of registered students is zero.

Scenes:
(i) “Probability” class by Dr. Yamada has 3 students.
(ii) Mr. Masuda registered for the “Probability” class taught by Dr. Yamada and the “English” class taught by Dr. Suzuki. Mr. Kato did not register for them because they are not mandatory classes.
(iii) Mr. Sato registered for the “Probability” class taught by Dr. Yamada, the “English” class taught by Dr. Suzuki, the “Programming” class taught by Dr. Tanaka, and the “Management” class taught by Dr. Nagata.

Evaluation criteria of the task are composed of the following two factors: (1) the created class and object diagrams are consistent with the input model description; and (2) the created class and object diagrams are consistent with each other. In this experimental study, the correct answer required accuracy in both criteria. An example of the expected answer of the example task is shown in Figure 6.

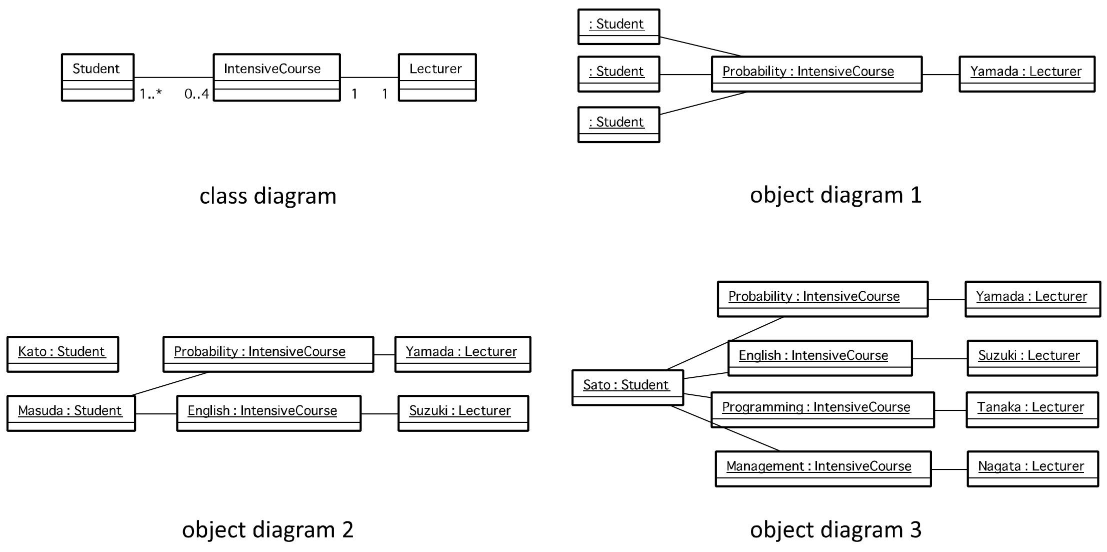


Figure 6: Example of Task B: creating a static model

Experimental study plan

The experimental study was conducted with thirteen university students randomly assigned to one of two treatment groups; therefore, six are assigned to a control group and seven are assigned to an experimental group. All subjects had at least three class hours about the fundamentals of modeling and describing object diagrams. Additionally, they had practice using the CASE tool, and we confirmed all participants could create class and object diagrams using the tool.

All subjects were given three tasks in the same order (Task A-1, Task A-2, and Task B). Task A-1 and Task A-2 are different in the complexity of the model. Only the experimental group could use our system to solve the given tasks.

The study was conducted in six steps:

1. Instructions for the usage of the system (4 min): the opportunity to watch instructional video showing how to use our system was given to only subjects in the experimental group; others had no work during this time.

2. Instructions for Task A (5 min): the opportunity to watch instructional video showing Task A was given to all subjects.
3. Task A (10 min): Task A-1 and Task A-2 were conducted.
4. Instructions for Task B (5 min): the opportunity to watch instructional video for Task B was given to all subjects.
5. Task B (20 min): Task B was conducted.
6. Questionnaire (5 min)

The screen that the subjects were operating was recorded by a screen capture application and was analyzed to discover the system usage in performing the tasks.

RESULTS

Performance: accuracy rate and answer time

1. Results for Task A

Results for Task A are summarized in Table 1. For each of the groups, the top numbers in each section of Table 1 represent the percentage of correct answers, and the bottom numbers in each section represent average answer time. The time frame represented in the table is in minutes and seconds (e.g., 3:06 represents 3 min and 6 s).

Table 1: Results for Task A

	Task A-1	Task A-2
Control Group	85.7% (6/7) 3:06	71.4% (5/7) 2:18
Experimental Group	100% (6/6) 3:28	100% (6/6) 3:01

In both TaskA-1 and TaskA-2, the accuracy rate of the experimental group was 100%, whereas a few subjects missed their answer in the control group. We analyzed the reasons for missing answers and found the causes were missing multiplicity or missing links in object diagrams. Although there were a few subjects who had a similar mistake, they fixed it thanks to the advice of our system.

2. Results for Task B

Results for Task B are summarized in Table 2. Just as in Table 1, the top numbers in each section of Table 2 represent the percentage of correct answers, and the bottom numbers in each section represent average answer time. The accuracy rate for the experimental group was 83.3% whereas the rate for the control group was 57.1%. Both groups had a low rate compared with Task A's scores. Three subjects missed the task in the control group, one of them missed Task A-2, and another one missed Task A-1 and Task A-2.

Table 2: Results for Task B

	Task B
Control Group	57.1% (4/7) 18:18
Experimental Group	83.3% (5/6) 19:05

We analyzed how participants performed for the missed subject in the experimental group. As a result, we found the subject who could not find a way to fix the inconsistency shown by our system. More specifically, we concluded that the subject missed it because multiple messages were shown to students in that case.

Student usage of our system

To determine how many functions of the system were used in the experimental group, we counted the number of checking functions were used in the group. Results are summarized in Table 3. The "Check" column in Table 3 represents the number of checking functions used. The column "OK" represents the number of results for which the check was OK. The column "INCST" represents the number of inconsistency messages shown to subjects. The column "AMB" represents the number of ambiguous messages shown to subjects. Both the number of inconsistency and ambiguous message are increased when both of the messages were shown in one check.

Table 3: Student usage of our system

	Task A (1, 2)				Task B			
SubjectID	Check	OK	INCST	AMB	Check	OK	INCST	AMB
A	4	2	2	0	8	1	5	6
B	2	2	0	0	7	1	5	2
C	2	2	0	0	4	1	0	3
D	3	2	1	0	4	0	4	4
E	4	2	2	0	0	-	-	-
F	2	2	0	0	6	1	4	5

For all the cases in the inconsistency and ambiguous messages shown, subjects tried to change the diagrams to fix the errors. That indicates that the system succeeded to promote improving the model for clarity and consistency of multiplicity. Exceptionally, there is a subject (E) who did not use the function. Although we failed to detect the reason for that, it is assumed to be a shortage of time for checking due to taking time to create the initial version of diagrams.

DISCUSSIONS

Evaluation of the functions

Although the number of subjects was limited, the scores for the experimental group was higher than that of the control group in all three of the tasks. Results of the qualitative consideration (i.e. observing how participants in the experimental group performed their tasks) supports the success of our system in preventing careless mistakes and promoted the opportunity to revise the model by giving messages to students. Additionally, no significant differences were shown in the average time of task completion. These results indicate the effectiveness of the proposed system for managing the clarity and consistency of multiplicity between class and object diagrams.

We designed the system for beginners who face difficulties managing consistency and clarity. Results showed that the subjects had increased confidence for their models by passing the diagnosis function. In short, we attained the objective. There were some cases, such as "finding careless mistakes" and "confirming the correction" that indicate that the function is effective for all students to create models.

Complexity of the model and the effects of the system

The total average score for Task B was lower than that of Task A, indicating that Task B has more difficult factors, which is reasonable because students have to not only read diagrams, but also create diagrams. The difference of the score for Task B between the experimental and control group is larger than that of Task A. These results indicate the system is more effective for the model created by students themselves even if it contains only simple relations without inheritance or recursion.

We hypothesize that the effect of the function will be emphasized for the more complex tasks, for example, models composed of inheritance or recursive relationships. In the next step of our research, we will try to support such elements. Supporting the Object Constraint Language (OCL) is another of our further considerations.

References

Auer, M., Tschurtschenthaler, T., & Biffl, S. (2003). A flyweight UML modelling tool for software development in heterogeneous environments. *The 29th EUROMICRO Conference*, 267-272.

Dranidis, D. (2007). Evaluation of StudentUML:An educational tool for consistent modeling with UML. *Proceeding of the Informatics Education Europe II Conference*, 248-256.

Engels, G., Hausmann, J., & Lohmann, M. (2006). Teaching UML is teaching software engineering is teaching abstraction. *In J.-M. Bruel (Ed.), International Conference on Model Driven Engineering Languages & Systems (MoDELS) 2005 Workshops*, 306-319.

Hasker, R. W. (2011). UMLgrader:An automated class diagram grader. *Journal of Computer Sciences in Colleges*, *27*, 47-54.

Hasker, R. W., & Rowe, M. (2011). UMLint: Identifying defects in UML diagrams. *118th Annual Conference of the American Society for Engineering Education*.

Hausmann, J. H., Heckel, R., & Sauer, S. (2002). Extended model relations with graphical consistency conditions. *In Proceedings UML 2002 workshop on consistency problems in UML-based software development*, 61-74.

Jonouchi, T. (2006). Learning object-oriented software development by creating game. *Transactions in Environmental Information at Yotsukaichi University (in Japanese)*, 49-64.

Kodama, K. (2008). *Introduction to the UML modeling*, NikkeiBP (in Japanese).

Kuznaiarz, L., & Staron, M. (2006). Best practice for teaching UML based software development. *International Conference on Model Driven Engineering Languages & Systems (MoDELS) 2005 Workshops*, 320-332.

Laleau, R., & Polack, F. (2007). Using formal metamodels to check consistency of functional views in information systems specification. *Information and Software Technology*, *50*(7), 797-814.

Lucas, F. J., & Alvarez, A. T. (2005). A precise approach for the analysis of the UML models consistency. *In Bp-UML'05: 1st international workshop on best practices of UML, in 24th international conference on conceptual modeling (er2005)*, 74-84.

Lucas, F. J., Molina, F., & Toval, A. (2009). A systematic review of UML model consistency management. *Information and Software Technology*, *51*(12), 1631-1645.

Paige, R. F., Brooke, P. J., & Ostroff, J. S. (2007). Metamodel-based model conformance and multiview consistency checking. *ACM Transactions on Software Engineering and Methodology*, *16*(3), 1-49.

Paige, R. F., Kaminskaya, L., Ostroff, J. S., & Lancaric, J. (2002). BON-CASE: An extensible CASE tool for formal specification and reasoning. *The Journal of Object Technology*, *1*(3), 77-96.

Ramollari, E., & Dranidis, D. (2007). StudentUML: an educational tool supporting object-oriented analysis and design. *Proceeding of the 11th Panhellenic Conference on Informatics*.

Scott A. T., Manuel A. PQ., & Stephen H. E. (2005). minimUML: A minimalist approach to UML diagraming for early computer science education. *ACM Journal on Educational Resources in Computing*, *5*(4), 1-28.

Steavens, P., & Pooley, R. (2000). *Using UML: Software engineering with objects and components*.

Biographies

Yoshiaki Matsuzawa received his PhD in Media and Governance in 2008 from Keio University, Japan. He is an assistant professor in the Faculty of Informatics at Shizuoka University. His research interests include software-engineering education, educational software development, modeling and simulation of complex systems, and learning sciences.

Kotaro Nozawa is a student in the Graduate School of Informatics at Shizuoka University. His research interests include educational software development, and UML modeling.

Yasuhiro Noguchi received his PhD in engineering in 2008 from Shizuoka University, Japan. He is a specially appointed assistant professor in the Faculty of Informatics at Shizuoka University. His research interests include intelligent tutoring system, natural language processing, dialogue system, and education for system engineers.

Sanshiro Sakai received his PhD in engineering in 1984 from Shizuoka University, Japan. He is a professor in the Faculty of Informatics at Shizuoka University. His research interests include computer supported collaborative learning, software development environment and programming learning support system.