Teaching Models @ BIG: On Efficiently Assessing Modeling Concepts*

Marion Brandsteidl, Martina Seidl, and Gerti Kappel
Institute of Software Technology and Interactive Systems
Vienna University of Technology
Vienna, Austria
{brandsteidl|seidl|kappel}@big.tuwien.ac.at

ABSTRACT

Approximately 1000 students of computer science and business informatics attend the course *Introduction to Object-Oriented Modeling* (OOM) offered by the Business Informatics Group (BIG) of the Vienna University of Technology each year in order to learn the basic concepts of the Unified Modeling Language (UML) and to obtain a certificate. For finishing the course successfully, the students must pass three small tests where they have to prove their theoretical knowledge about UML concepts as well as their ability to apply this knowledge in practical exercises. In this paper we report our experiences in assessing the modeling knowledge of our students and we reveal how we design the tests.

Keywords

Teaching Object-Oriented Modeling, Basic Modeling Course, Assessing UML

1. INTRODUCTION

One of the most challenging tasks in teaching is the preparation of suitable and adequate exams. Exam questions should fulfill multiple (sometimes orthogonal) requirements at the same time: the questions should cover the most important aspects of the lecture, they should assess whether the students have reached the learning goals—and also if we have reached our teaching goals, they should have an adequate level of difficulty and they should allow to test if the students have not only learned the contents of the course by heart, but if they also understand the teaching material as well as if they are able to apply the taught concepts to "real world problems". The questions should be answerable within the duration of the test and they should be capable of being assessed with reasonable effort. In this paper we report how we design the tests of our university course *Introduction to Object-Oriented Modeling* (OOM).

*This work has been partly funded by the Austrian Federal Ministry of Transport, Innovation and Technology (BMVIT) and FFG under grant FIT-IT-819584.

OOM is attended by about 1000 undergraduate students per year who study computer science or business informatics at the Vienna University of Technology. In OOM we teach modeling basics by introducing syntax and semantics of UML 2 models[1] [3, 5]. Profound knowledge about the UML is a prerequisite for advanced courses like Model Engineering or Software Engineering. Despite the huge amount of students, we try to avoid mass processing, but we try to establish personal mentoring instead. Besides a traditional lecture where structural as well as behavioral modeling techniques are introduced, we organize the lab as exercise courses in smaller groups where the theoretical contents of the lecture are practiced. Furthermore we provide support via online forums on the e-learning platform TUWEL[2] (a Moodle adaptation of the Vienna University of Technology). A detailed description of the course is given in [1].

The *lecture* consists of eight units covering the following UML 2 diagrams: class and object diagram, sequence diagram, state diagram, activity diagram, and use case diagram. Although not mandatory, the attendance of the lecture is recommended as there is room for discussion and asking questions. The attendance of the lecture has no direct influence on the grade. For the *practical part* the students are divided into groups of 50 persons. Each group meets six times during the semester in order to discuss the solution of exercise sheets for practicing modeling. For each exercise the assistant professor chooses one student who must present and explain his/her solution as well as answer questions about the theoretical background. To pass the course, the students have to solve at least 24 of the 36 exercises. For further support, we provide e-learning exercises which aim at helping the students to get some modeling practice. Completing those exercises has no influence on the grade.

The participation in OOM is awarded with 3 ECTS points resulting in a total workload of 75 hours for an average student. To obtain a positive grade, three tests have to be passed besides the successful completion of the practical part. We consider tests not only as means to obtain grades, but also as important didactical resource. Hence we put much effort in the development of a question catalogue. Over the time we have collected a wide range of exam questions for assessing modeling basics and we have gained much experience in the organization which allows us to deal with the huge amount of students in an effective manner.

[1] http://www.uml.org/
[2] http://tuwel.tuwien.ac.at

2. TEST STRUCTURE

We assess the modeling knowledge of our students with three tests, each covering one or two different UML 2 diagram types—the topic of the first test is the class diagram, test two is about sequence diagram and state diagram, and test three contains activity diagram and use case diagram. Each test has a duration of 30 minutes and one of the three tests may be repeated at the end of the term. We design most of the questions new each semester, using parts of the old test questions (or at least the problem scope) for the exercise sheets.

Although Use Case Diagrams are modeled within the early phases of the software development life cycle, we teach it as the last diagram type in our lecture for didactic purposes. Students tend to underestimate the importance and the level of difficulty this diagram type has, thus not putting enough effort into the whole lecture. We decided to teach Class Diagrams first due to the fact that students of computer science tend to pay more attention to "wicked cool java" than to "just drawing boxes and lines". The Class Diagram includes many familiar programming concepts.

We decided to give three small tests instead of one large test at the end of the term as we experienced that it better supports the students in learning when they do not have to study the complete content at once and they may prepare themselves in a more focused manner. From their feedback we learned that students prefer multiple small tests.

When designing the questions for the three tests, we have to keep two important issues in mind: first of all, the tasks have to be solvable within 30 minutes, but even so they have to cover all the important teaching contents to make sure the students have learned enough and understand the concepts. Second, keeping in mind that three tests for 1000 students result in 3000 tests which have to be marked each year, it has to be possible to correct the tests within reasonable time—but without cutting back on the quality of the test questions.

Carefully designing the questions saves a lot of time and effort later on, because a little mistake in the declaration of a question might result in big confusion during the test itself or might increase the marking effort as the question was not formulated precisely enough and the person who corrects the tests has to deal with a bunch of different solution approaches.

Assuming one single person has to mark 3000 Tests consisting of answers to free response questions, marking the tests could last weeks or even months. In our exams, a mixture of multiple choice questions, semi-open questions and modeling exercises has turned out to be a good compromise, some of the question types allowing to be marked by people with little UML knowledge or even with no UML knowledge at all (like, e.g., our secretary).

In our tests we try to include various kinds of exercises. Overall one test is worth 100 points and we try that at least the half of the points demands understanding the contents instead of simple memorizing. The different types of exam questions and their characteristics as well as their pros and cons are described in the next sections.

3. MULTIPLE CHOICE QUESTIONS

With the term of multiple choice questions, we refer to a set of statements about the syntax and semantics of a UML diagram type in general or about a given UML diagram, where the students have to mark each statement as true or false.

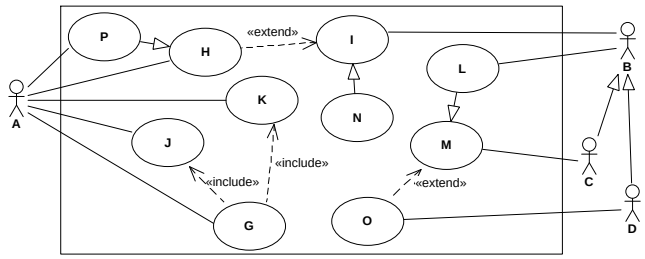
It is easily possible to vary the complexity of the statements and the correction of this question type may be done very quickly without the need of UML knowledge. Unfortunately, besides the big advantages of multiple choice questions in correction speed, there are also some severe drawbacks: each statement has to be formulated with care, thus being as precise as possible and avoiding any kind of misunderstanding. In contrast to free response questions, multiple choice questions do not allow the students to make an argument for their viewpoint, which could result in wrong answers if the statements leave space for interpretation. This exercise type is also very hard if the language of the statements is not the student's mother tongue, thus students with migration background tend to have problems understanding the statements. For those students, other types which are less dependent on a (natural) language are more suitable.

In our tests, one multiple choice exercise consists of five statements each of which is either true or false. Each correct answer is awarded 3 points, an incorrect answer is marked -1 point and no answer at all results in 0 points. The minus points deter the students from trying their luck checking the true and false boxes randomly. The answer pattern of a block of five statements is easy to memorize and therefore it is graded very quickly. We have also tried to put six or seven questions in one block, which resulted in a significant increase of correction time, because in general people seem to have problems memorizing more than five answers. Then they have to correct each statement separately instead of simply matching the pattern of the student's solution to the pattern of our sample solution.

Exercise 1. Are the following statements true or false?

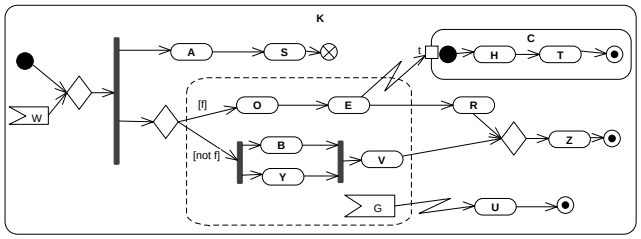
Class Diagram: A class may have zero or more superclasses and zero or more subclasses.	☐ true ☐ false
Sequence Diagram: The interaction operator `opt` designates that the combined fragment represents traces that are defined to be invalid.	☐ true ☐ false
Activity Diagram: A merge is a control node that synchronizes multiple flows.	☐ true ☐ false
Use Case Diagram: An actor is an idealization of a role played by an external person, process, or thing.	☐ true ☐ false
State Diagram: A guard condition is evaluated when the corresponding trigger event occurs.	☐ true ☐ false

Exercise 2. The given Use Case Diagram was modeled strictly according to the UML 2 standard. Are the following statements true or false?



Actor `D` is involved in the use cases `L`, `M`, and `O`.	☐ true ☐ false
Every time `G` is executed, `J` is also executed.	☐ true ☐ false
Actor `B` can execute `O`.	☐ true ☐ false
The behavior defined in `H` can be inserted into the behavior defined in `I`.	☐ true ☐ false
The behavior of `N` may be extended by the behavior of `P`.	☐ true ☐ false

Exercise 3. Are the following statements about the given UML 2 Activity Diagram true or false?



As soon as the execution of action `S` is complete, the whole activity `K` is terminated.	☐ true ☐ false
Assume error `t` occurs during the execution of `E`. Then, `C` is executed and afterwards the control flow continues the proposed way, hence `R` is executed regularly.	☐ true ☐ false
`Z` is executed only after waiting for both `R` and `V` to complete.	☐ true ☐ false
It is impossible that both actions `E` and `B` are executed in a single passage of activity `K`.	☐ true ☐ false
The acceptance of signal `G` invokes action `U`.	☐ true ☐ false

3.1 Theoretical Multiple Choice Questions

This kind of question focuses on testing the knowledge of the UML syntax and the definitions as well as the usage of certain modeling objects and their practical appliance. As each of our tests focuses on one or two UML 2 diagram types, we usually have one block of questions about a single diagram type, in this paper we mixed the questions about the diagram types for demonstration reasons. Exercise 1 illustrates an example of this type of multiple choice questions.

3.2 Multiple Choice Questions Based on a Given UML Diagram

Another type of multiple choice exercise is based on a given UML diagram, to which the statements refer. Hence this type of questions assess not only theory, but also if certain concepts and their relationships are well understood. Exercise 2 shows an example of the assessment of the inheritance and include and extend relationships in Use Case Diagrams. In Exercise 3, the knowledge about the control flow in activity diagrams is tested.

4. ERROR FINDING

This question type consists of a concrete UML diagram and a corresponding textual description, pseudo-code fragment, or another UML diagram. The examinee has to find errors in the given diagram and correct them. When designing erroneous diagrams, it has to be taken care that the errors are non-ambiguous errors and that the rest of the diagram is definitely correct. If there is only one way—or at least a manageable amount of ways—of correcting the planted errors, this type of question does not have to be marked by an expert modeler, but it may be corrected by a person with basic modeling skills, e.g. a student tutor (assuming he/she is trained first).

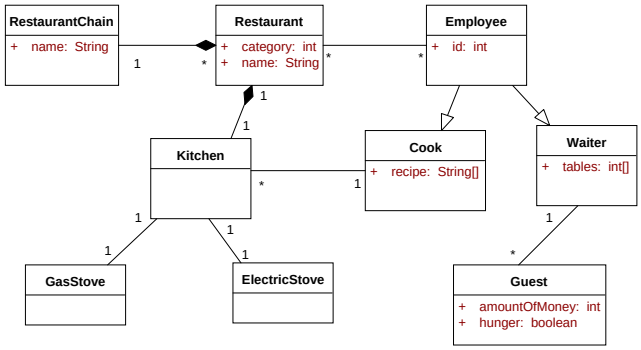
Depending on the diagram type and the designated size of the question, we integrate 3 or 5 errors, each worth 3 to 5 points, depending on the kind and complexity of the error. When first using this question types into our tests, we asked the students to mark the errors and explain textually why the found error is an incorrect construct. That was a good way to find out if the examinee really understood for what reason the error was an error and not just marked any 3 or 5 parts of the given diagram randomly. Unfortunately, the correction of the textual justifications took a lot of time. So we changed the question specification to correcting the errors instead of just marking and describing them. To avoid the students from crossing out the whole diagram (thus eliminating all errors which could possibly exist), each corrected error which has not been an actual error costs 3 to 5 points.

4.1 Error Finding Based on a Model and a Corresponding Text

The most obvious way to design an error finding task is to give a diagram and a text describing the problem space the diagram is intended to be showing (please refer to Exercise 4). The biggest challenge with this question type is mentioning every single part of the diagram in the corresponding text, ensuring that the only things left out or noted differently in the text than displayed in the diagram are the errors the students have to find. At the same time the textual description should be kept as brief and clear as possible to avoid unnecessary confusion. Usually, the students are extremely fond of this kind of exercise and achieve good results.

Exercise 4. The following Class Diagram should conform to the given textual specification. Unfortunately, the diagram contains some errors. Find 5 errors and correct them in the diagram.

A restaurant chain consists of several restaurants. Every restaurant chain has a name, a restaurant has a category and a name. Each restaurant has exactly one kitchen. In each restaurant there work several employees (identified through an identification number), an employee can work in several restaurants. For each employee the salary is known. There are two different kinds of employees: waiters (the tables he/she serves are stored) and cooks (his/her recipes are stored). In each kitchen, at least one cook works. Each waiter serves several guests. Each guest is hungry (or not) and spends a certain amount of money. Each kitchen has a gas stove or an electric stove.



4.2 Error Finding Based on Two Corresponding UML 2 Diagrams

A possibility to avoid misunderstandings caused by ambiguous statements, lack of knowledge about the problem domain, or language problems is to replace the textual description with a second UML diagram or with pseudo code. This section focuses on the linking between two diagrams, the next section will focus on a diagram and corresponding code. Students often tend to concentrate on one specific diagram type, forgetting that the types are linked, each one of them describing the problem scope at a different level of detail and/or from a different point of view. To assess the ability to link between two diagrams, one possible way is to give two corresponding diagrams and plant errors in one of them. Exercise 6 shows a task, where a Class Diagram and a corresponding object diagram have to be compared.

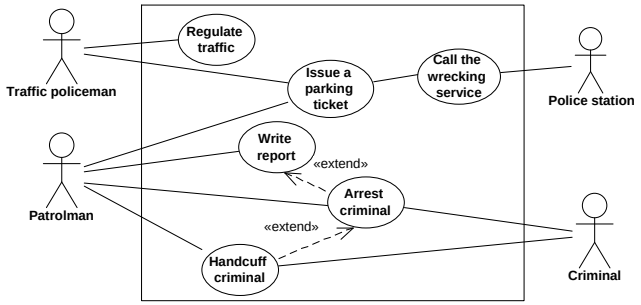
4.3 Error Finding Based on a UML Diagram and Corresponding Code

The diagrams are not only linked between each other, but some diagram types can also be used for reverse engineering the code. To assess this connection between code and a UML diagram, the students are given a fragment of pseudocode and a corresponding clipping of a sequence diagram and they have to correct the errors in the sequence diagram (see Exercise 7).

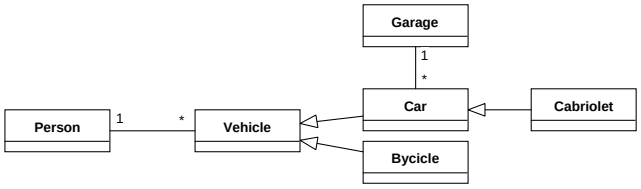
Exercise 5. The following Use Case Diagram was modeled strictly according to UML 2 standard. Unfortunately, the diagram contains some errors. Find 3 errors and correct them in the diagram.

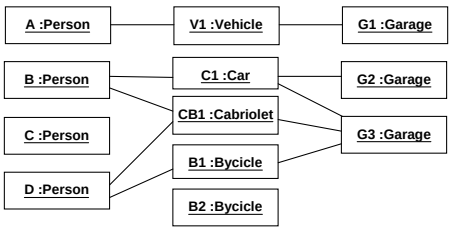
There are two different kinds of policemen, traffic policemen and patrolmen. Traffic policemen regulate traffic. Patrolmen arrest criminals. When arresting a criminal, the patrolman always has to write a report. Sometimes the criminal has to be handcuffed.

A parking ticket can be issued by a traffic policeman or a patrolman. Sometimes it is also necessary that the police station calls the wrecking service.



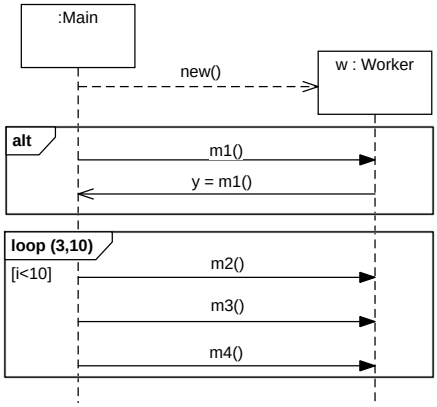
Exercise 6. You are given the following UML 2 Class Diagram. A modeler tried to create a corresponding Object Diagram. Unfortunately, the modeler made some mistakes. Scratch out as many associations and objects of the Object Diagram as possible to make the Object Diagram conform to the Class Diagram.





Exercise 7. You are given the following pseudo-code. A modeler tried to visualize the communication with a UML 2 Sequence Diagram. Unfortunately, the modeler made some mistakes. Find 5 errors and correct them in the diagram. Assume that all used variables are already declared and initialized and that the method calls are modeled correctly.

```
class Main {
...
  Worker w = new Worker();
  if (x == 5) { y = w.m1(); }
  for (int i = 3; i < 10; i++) {
    w.m2();
    w.m4();
    w.m3();
  }
...
}
```



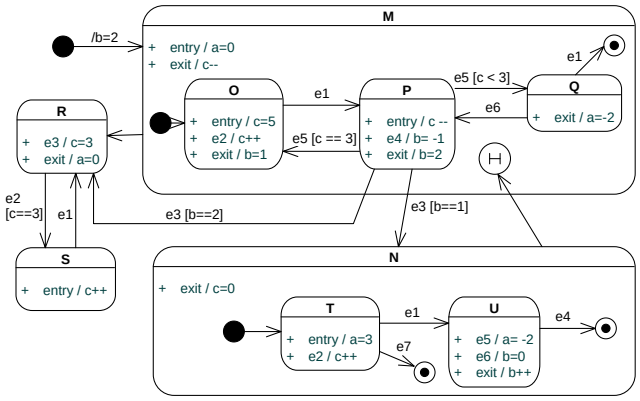
5. UNDERSTANDING SEQUENCES

In State Diagrams, an object changes its state depending on certain events and under certain circumstances (guards), Sequence Diagrams illustrate the possible ways of communication between interaction partners and activity diagrams show possible sequences of actions.

Understanding this sequences is the same as understanding the underlying concepts of the given diagram type, so assessing sequences seems obvious. In Exercise 8, the students are given a state diagram and a corresponding chain of events. They are asked to complete the table to show which event triggers which action. This kind of question is language-independent and can be graded without any knowledge about UML, but it needs the students to be very concentrated while completing the task. It is of great importance that the sequences are not too long.

Exercise 9 shows an exercise where the examinee is given a Sequence Diagram and a set of possibly corresponding traces.

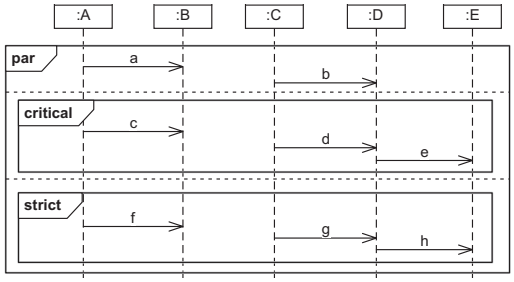
Exercise 8. You are given the following UML 2 State Diagram. Fill in the given spreadsheet to show the values the variables take and the states entered during the following event chain.



value of each variable

event	state	a	b	c
beginning				
e1				
e3				
e2				
e1				
e4				
e5				
e1				
e3				
e2				

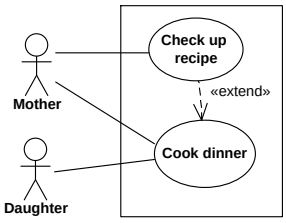
Exercise 9. You are given the following (simplified) UML 2 Sequence Diagram with 8 messages (`a-h`) and five examples of possible traces. Do the given traces correspond to the Sequence Diagram?



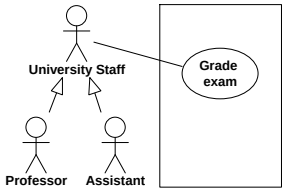
a → b → c → d → e → f → g → h	☐ true ☐ false
a → b → d → c → e → f → g → h	☐ true ☐ false
a → b → d → c → e → g → h → f	☐ true ☐ false
a → c → d → e → f → g → h → b	☐ true ☐ false
a → c → d → f → g → h → e → b	☐ true ☐ false

Exercise 10. Model the following facts with Use Case Diagrams—strictly according to UML 2 standard.

- A mother and a daughter cook dinner together. It is possible that the mother has to check up the recipe during cooking.
  possible solution:

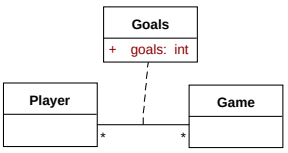


- An exam is graded by a professor or an assistant.
  possible solution:

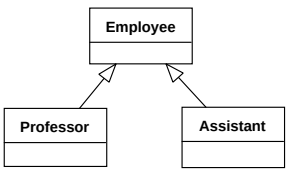


Exercise 11. Model the following facts with Class Diagrams.

- During a football season, several football players play several games. Each player scores a (different) amount of goals in each game.
  possible solution:



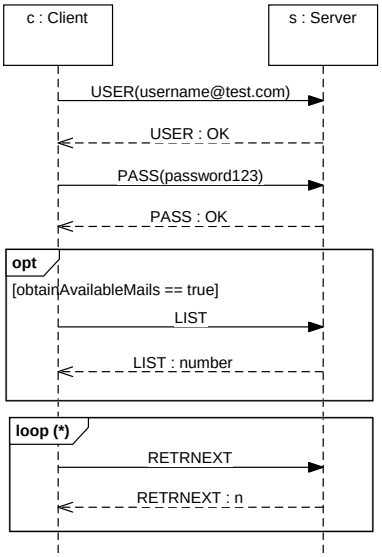
- There are two different kinds of employees, professors and assistants.
  possible solution:



Exercise 12. Model the communication occurring during a session between a mail server and a client according to the following protocol with a UML 2 Sequence Diagram.
In order to connect to the server, the client sends its username (`username@test.com`) and its password (`password123`) to the server. Assume that username and password are correct and the server replies with OK (error situations do not have to be considered). Optionally, the user may retrieve the number of available mails (condition `obtainAvailableMails == true`). Then the client receives all unread emails. Finally, the client closes the session.
The following commands are available:

- `USER xxx` ... sends the user name to the server indicating that a new session will be started
- `PASS xxx` ... sends the client's password in plain text
- `LIST` ... returns the number of available mails
- `RETRNEXT` ... returns the next unread mail
- `QUIT` ... terminates the session

possible solution:



6. MODELING DIAGRAMS

The most obvious way to assess modeling skills is to ask the students to derive a certain diagram from a given textual description, code clipping, etc. Even though those exercises require much more time and skilled staff to be marked, we will not fully refrain from this type of questions for didactic purposes: we have to ensure that the students are able to derive models themselves.

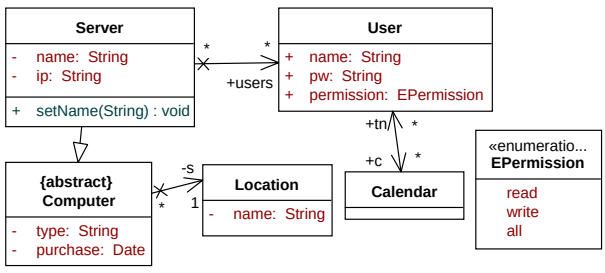
Open modeling tasks have to be designed carefully to keep the correction effort as low as possible. The question specification has to be as precise as possible, allowing just a small set of different solution approaches—the discussion about different approaches to one problem space depending on the goal the modeler follows is done during the discussion of the

lab exercises. Often it is also useful to think about a scheme for grading the exercise while designing it—for example "-3 points for a missing association". This helps to find potential problems occurring while marking the tests later on. We also always ask two other staff members to solve the tasks trying to find out if the specification is clear enough.

Exercise 13. Model an UML 2 Class Diagram compliant to the given pseudo-code. If possible, model references as associations. Include the navigation directions, visibility symbols, types, role names and multiplicities which result from the pseudo-code.

```
abstract class Computer {
    private String type;
    private Date purchase;
    private Location s;
 }

class Server extends Computer {
    private String name;
    private String ip;
    public User[] users;

    public void setName(String name) {...}
}

class Location {
    private String name;
}

class User {
    public String name;
    public String pw;
    public Calendar[] c;
    public EPermission permission;
}

class Calendar {
    public User[] tn;
}

enumeration EPermission {
    read;
    write;
    all;
}
```

possible solution:



6.1 Modeling Diagrams Based on a Text Describing the Problem Space

If the question specification is textual, besides the difficulties about precise wording mentioned before, one also has to be careful to choose a problem domain which is well known enough by all the students and which does not require too much "special" words. Small clippings of diagrams can be corrected with reasonable effort—provided that the text is formulated clearly enough to get just a small amount of possible ways to model the given task. Exercises 10 and 11 show very small modeling tasks which still are sufficient to test some of the basic concepts of Use Case diagrams (namely associations and inheritance) and Class Diagrams (namely association classes and inheritance). Exercise 12 shows an example for a bigger task, resulting in more—but still reasonable—correction effort.

6.2 Modeling Diagrams Based on Pseudo Code

Giving the students pseudo-code and ask them to derive a diagram overcomes the problems of language. Furthermore knowledge of the problem domain is not absolutely necessary (assuming the examinee understands the pseudo-code). The pseudo-code may only contain basic programming constructs, for most students do not have broad programming skills yet. Exercises 13 and 14 are examples for such tasks.

7. CONCLUSION

Several papers(e.g., [1, 2, 4]) propose ways to structure and organize courses on modeling, but they touch one didactic instrument, namely the tests, only very marginally. Unfortunately, many students tend to learn only for passing tests following a minimal effort strategy. Hence, we try to design our tests in such a manner that the students have to acquire sustainable knowledge in order to finish the course. Over the years we have collected a wide range of exam questions for assessing object-oriented modeling with special emphasis on the Unified Modeling Language and we have gained much experience in the organization of exams. This allows us to deal with the huge amount of students in an effective manner. Besides the assessment of theoretical content, also practical exercises are given which demand not only a deep knowledge of the facts but also a profound understanding of the teaching material for their solution. Our exercises vary in their degree of freedom from restricted multiple choice questions to open modeling exercises.

Currently, our tests are still done on paper sheets, hence a next logical step would be to conduct them electronically. Although repeatedly discussed, we currently abstain from this idea, because electronic assessments would introduce numerous new problems. Besides the technical realization and need for an adequate infrastructure, we would need questions which can be corrected automatically what we consider as the main advantage in electronic testing. Currently, the actual execution and correction of one test takes about one half of a working day if we rely on the complete manpower of our department (student tutors, pre- and post-doc assistants, and our secretary). If we would carry out the tests electronically, we would block the hugest laboratory of the faculty for at least a week and probably would still have the correction effort for the open modeling questions from which we will not refrain for didactic purposes.

Overall, we have found a way to effectively assess modeling, in particular UML. Over the years we collected a huge number of test questions some of which we revealed in this paper. We presented a wide spectrum of modeling assessments with various degrees of freedom. We introduced these by giving examples from our "Introduction to Object-Oriented Modeling" course and we shortly reported on our experiences.

8. REFERENCES

[1] M. Brandsteidl, M. Seidl, M. Wimmer, C. Huemer, and G. Kappel. Teaching Models @ BIG: How to Give 1000 Students an Understanding of the UML. In *Promoting Software Modeling Through Active Education, Educators' Symposium MoDELS 2008*, pages 64–68. Warsaw University of Technology, 2008.

[2] G. Engels, J. Hausmann, M. Lohmann, and S. Sauer. Teaching UML is Teaching Software Engineering is Teaching Abstraction. In *Satellite Events at the MoDELS 2005 Conference, Educators' Symposium MoDELS 2005*, volume 3844 of *Lecture Notes in Computer Science*, pages 306–319. Springer, 2006.

[3] M. Hitz, G. Kappel, E. Kapsammer, and W. Retschitzegger. *UML@Work, Objektorientierte Modellierung mit UML 2*. dpunkt.verlag, Heidelberg, 2005.

[4] L. Kuzniarz and M. Staron. Best Practices for Teaching UML Based Software Development. In *Satellite Events at the MoDELS 2005 Conference, Educators' Symposium MoDELS 2005*, volume 3844 of *Lecture Notes in Computer Science*, pages 320–332. Springer, 2006.

[5] C. Rupp, S. Queins, and B. Zengler. *UML Glasklar. Praxiswissen für die UML-Modellierung*. Hanser Fachbuch, 2007.

Exercise 14. Extend the given UML 2 Sequence Diagram to visualize the message interchange when the given pseudo-code is executed. Also model return messages if necessary. Assume that all variables which are not explicitly declared in the pseudo-code are already declared and initialized.

```
class Main {
  ...
  Worker w = s1.getConnection(user, pw);

  if(w==null) {
    print("Error");
    exit;    // program terminates
  }

  status = w.sendMail("abc", "test");

  do {
    m = w.getMail();
    print(m);
  } while (m != null);
  ...
  private void print(String m) {...}
}

class Server {
  ...
  public Worker getConnection(
    String user, String pw) {
    Worker w = new Worker();
    w.start();
    return w;
  }
  ...
}

class Worker extends Thread {
  ...
  public void run() {...}

  public boolean sendMail
      (String msg, String receiver) {...}

  public String getMail() {...}
}
```

possible solution: