JOURNAL OF OBJECT TECHNOLOGY

Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2004**

Vol. 3, No. 1, January-February 2004

On the Contribution of UML Diagrams to Software System Comprehension

Irit Hadar and Orit Hazzan, Technion – Israel Institute of Technology, Israel

Abstract

Program comprehension has been researched extensively ever since software systems became complex and longer than a few hundred code lines. At the same time, the way in which people comprehend visual models of software systems has received much less attention. This paper focuses on the comprehension of UML diagrams. During the research presented in this paper, data was gathered from the work of two groups. Group 1 consisted of 13 senior computer science students who worked in five teams. The students were asked to trace and analyze the process by which they retrieved information from UML diagrams of a given system. Group 2 consisted of 42 senior computer science students who were requested to complete a questionnaire in which they were asked to rank different types of UML diagrams according to their importance. The section on data analysis discusses strategies adopted by the novices in their attempt to reveal the meaning of a set of UML diagrams, as well as their attitudes towards the different diagrams. One of the interesting observations is that although each team had its own preferences with respect to the usefulness of each specific type of diagrams, the overall use of each diagram type is very similar across the teams.

1 INTRODUCTION

UML has attained the status of a de facto modeling language standard ([Booch, 1999], [Kobryn, 1999], [OMG, 1999]). According to our literature review, however, only few studies have examined how people write or comprehend object-oriented visual models in general, and UML diagrams in particular. At the same time, the writing and comprehension of computer programs have been studied extensively over the last few decades. Research in these areas looks at how novices and experts cope with these two activities. Research about the writing of computer programs examines, among other topics, how programmers battle the complexities involved in software development; while research on program comprehension looks mainly at strategies adopted by programmers while attempting to comprehend a computer program. Specifically, research about program comprehension deals mainly with topics such as mental models and cognitive processes ([Brooks, 1983], [Littman, Pinto, Letovsky and Soloway, 1987], [Letovsky, 1987]); program reading techniques and strategies ([Francel and Rugaber,

Cite this article as follows: Irit Hadar, Orit Hazzan: "On the Contribution of UML Diagrams to Software System Comprehension", in *Journal of Object Technology*, vol. 3, no. 1, January-February 2004, pp. 143-156. http://www.jot.fm/issues/issue_2004_01/article3

2001]); influence of computer programs presentation on program comprehension ([Bentley, 1986], [Knuth, 1984]); and the comprehension of computer programs by novices vs. experts ([Soloway and Ehrlich, 1984], [Pennington, 1987]). In general, these papers acknowledge the complexities involved in program comprehension. Moreover, it is repeatedly stated in the aforementioned studies that *no* unique theory for describing program comprehension has been proposed so far and that such a theory should take into consideration that "programming behavior can be understood only with reference to the interactions between multiple knowledge sources" ([Davies, 1993], p. 265).

The objective of our research was to identify and describe strategies applied in the process of comprehending visual models of software systems. We view this research as a new branch of the extensive research on program comprehension. In analogy to research about the comprehension of computer programs, UML diagrams have been selected to be the focus of the study described in this paper. Visual models are the medium by which the specification and design of software systems are expressed, as computer programs are the medium by which objects and algorithms are expressed. In this analogy, the UML corresponds to a specific programming language.

The study described in this paper was conducted with the participation of senior computer science students. The added value of this research which investigates the process of understanding a software system documented by the UML, is two-folded: First, the research outlines the manner in which students use and integrate information they retrieve from several types of UML diagrams; Second, it addresses the question of the relative importance of the different types of diagrams during that specific process.

Section 2 of the paper describes the research setting. Section 3 presents conclusions derived from the data analysis. In Section 4 the research results are discussed and directions for future research are presented.

2 RESEARCH SETTING

Methodology

The data collection and data analysis in this research were based both on quantitative and qualitative research approaches. The qualitative research does not intend to prove a quantitative theory or to develop a solution to a specific problem, but rather to add knowledge and insights regarding a phenomena or problem identified by the researcher ([Bassey, 1999]). This approach is appropriate for this research since there is no intention of statistically proving a certain hypothesis, but rather of investigating the approach taken by students while trying to understand a software system, as well as the ways in which they use the different UML diagrams to fulfill this objective. The quantitative data gathered and analyzed were integrated into the qualitative analysis.

The research tools used in this study included observations, students' learning materials and position questionnaires.

Population

Two groups of senior computer science major students participated in our research. The first group worked on a comprehension task based on UML diagrams; the second group completed a questionnaire that addressed the relative importance of different types of UML diagrams.

Group 1:

The participants in this group were 13 students, majoring in either Software Engineering or Information Systems, who were taking the course entitled Human Aspects of Software Engineering, taught by the second author at the Department of Computer Science at the Technion – Israel Institute of Technology.

The above course was taught parallel to a course entitled Methods in Software Engineering. The emphasis of the later course was on the entire life cycle of software, from the initial requirements, through analysis, design, implementation, integration and testing. In addition, the course provided complementary material on support activities, such as software maintenance and quality assurance. The formal requirement of the course was a team project, carried out by the students throughout the semester according to the progress of class lectures. At the beginning of the semester, the students received a "client document" (RFP) containing the requirements for a software system. The project was submitted in six stages: requirement document, analysis and specifications (UML), design (UML), test plan, code (in Java) and documentation of test results.

The Human Aspects of Software Engineering course was based, in part, on the Methods in Software Engineering course and some of the activities carried out during the course were associated with the project developed by the students in the Methods in Software Engineering course. The examination of this software development process was, however, carried out from a different angle. More specifically, while the focus in the Methods in Software Engineering course was on the *software* development process, the emphasis in the Human Aspects of Software Engineering course was placed on the *people* developing the project and mental and social processes were examined.

In general, the Human Aspects of Software Engineering course encourages a reflective mode of thinking in the spirit of [Schön, 1983]. Most of the students' tasks were based on the students' examination of their own way of thinking and working. Thus, one of the purposes of the task that provided the main data for our research (Cf. Table 1) was to encourage student reflection.

In addition, the work of two pairs of students on the task was used for its validation, as described in the next sub-section. These four students took the Methods of Software Engineering course, but did not participate in the Human Aspects of Software Engineering course.

Group 2:

The 42 participants in this group participated in the capstone course entitled 'Software Engineering Project' offered by the Department of Computer Science at the Technion. The course aims to train students in the entire process of software system development,

including requirement definition, conceptual and detailed design, unit implementation, integration and testing.

The Task

Table 1 presents the task that constitutes the basis for the first stage of our research. Students were given three weeks to complete this homework assignment. The 27 diagrams in this task were taken from [Paltor and Lilius, 1999]. Prior to handing out the task, one lesson of two hours was devoted to the topic of program comprehension in which several program comprehension theories were presented ([Brooks, 1983], [Littman *et al.*, 1987], [Fjeldstad and Hamlen, 1983], [Vans *et al.*, 1999]). The objective of the said lesson was to introduce students to the cognitive complexity of program comprehension as well as to the variety of heuristics available for program comprehension.

Table 1. The Task.

Homework Assignment # 3 (in groups of two or three students)

Software documentation may be lost occasionally due to maintenance problems. In this homework assignment, a collection of UML diagrams is presented. The documentation of the computer system has been lost. Your task is to re-create the system description based on diagram analysis. In addition, you are asked to document and analyze the process you went through (a detailed description is presented below).

This task focuses on program comprehension, which is a central topic in the human aspect of software engineering. The two main objectives of the task are the improvement of computer program comprehension and the enhancement of the understanding of mental processes.

More specifically:

a. You are given a collection of UML diagrams that describe a Digital Audio Recorder. The specification documentation has been lost. Your task is to analyze the diagrams, write a general system description, and specify particular use cases.
b. During your work, you are requested to trace and document the analysis process and the way in which information is retrieved. In order to support the documentation process, it is recommended to mark the diagrams using some notation.
c. After completing Stage b, please go back and analyze the process you went through and characterize your work process.

You are asked to submit the following:

- A description of the Digital Audio Recorder (general description as well as use cases).
- Follow-up documentation:
  i. The order of diagrams processed in your work.
  ii. The way in which you set to work on the task.

iii. Miscellaneous documentation (such as sketches on the UML diagram pages, etc).

- An analysis of the process of diagram comprehension and information retrieval:
  i. Process characterization and the rationale behind it.
  ii. A graphical model representing the process.
  iii. Suggestions for the process improvement.

Additional comments:

- It is recommended to dedicate at least 3-4 consecutive hours to diagram comprehension.
- The diagrams are given in random order and are not stapled together. We recommended that you start off by checking which diagrams were given.
- At the diagram analysis stage, one of the team members should focus on diagram comprehension and the other(s) on process documentation.

The task was performed by students of Group 1 (See 'Population') as a homework assignment, and thus it was highly important to validate not only the task itself, but the process of its execution as well. This was performed in three stages, using three teams that solved the task in our presence, as described below:

- Prior to Group 1 performing the task, a pilot study was conducted with one pair of students, who were not course participants. In this pilot, we observed the way in which the students confronted and interpreted the task, noted which parts of it attracted their attention, and how they faced the documentation requirement (Section b in Table 1). Following this session, we refined the task formulation and further clarified some of the instructions.
- During the time in which Group 1 performed the task, one of the five teams participating in the course solved the task in our presence. The aim of this observation was to ensure students worked on the task and understood it as intended.
- After Group 1 completed its work on the task, we asked another pair of students, who did not participate in the course, to solve the task while observing their work. The aim of this observation was to check the general consistency in the performance of students, who are not course participants, with the work performed by the students in the course. Although no generalization can be reached based on this one additional team, it is interesting to note that their strategy and work process were consistent with our findings from the analysis of the work of Group 1 students.

The Questionnaire

The questionnaire was developed during the second stage of the research. Its aim was to examine the results obtained from the first stage (the task) from a different perspective. Forty-two Group 2 students completed the questionnaire, which is presented in Table 2.

Table 2. The Questionnaire.

Questionnaire UML

Name: _____

Occupation (in addition to computer science studies): _____

Experience in object-oriented development: _____

The following table presents 9 types of UML diagrams. Please rank the contribution of the various diagrams to the **software development process** according to their importance (1 – highest contribution, 9 – lowest contribution, 0 – not familiar).

Type of Diagram	Rank
Use Case	
Activity	
Class	
Sequence	
Collaboration	
State Chart	
Object	
Package	
Deployment	

Please re-rank the diagrams. This time refer to the contribution of the different types to the comprehension of a software system in whose development you did not take part, that is, to the **comprehension of an unfamiliar software system**.

Type of Diagram	Rank
Use Case	
Activity	
Class	
Sequence	
Collaboration	
State Chart	
Object	
Package	
Deployment	

What are the reasons for the difference between the two rankings (if there is a difference)?

Thank you for your cooperation!

If you are willing to further contribute to this research, please specify the following details:

Tel.: _____ e-mail: _____

3 DATA ANALYSIS

Based on an analysis of the students' work and responses, we present three findings that refer to different stages of the comprehension process. One idea is common to these observations: Despite the small number of teams, relatively many different strategies were observed for each of the findings. In other words, no unique strategy can be identified in the different stages of the execution of the task. This finding clues about the nature of visual model comprehension.

a. Diagram sorting:

Although sorting of the diagrams was not part of the task, it is interesting to note that all teams sorted the diagrams according to some criterion before starting the actual examination of the diagrams. Specifically, five kinds of sorting were identified:

- By title (referred to by students as "the development phases"): This sorting process resulted in grouping all class diagrams together, all sequence diagrams together, and so on.
- By static vs. dynamic information: The students sorted the diagrams according to the kind of information they provide: static information (e.g., class diagrams) vs. dynamic information (e.g., sequence diagrams).
- By use cases: In this case, the diagrams were grouped according to the use cases presented in the use case diagram (cf. Appendix). Thus, all diagrams that contribute to the description of a specific use case were grouped together (e.g., all diagrams that might contribute to the description of the use case "delete a message" were grouped together).
- By objects: Students first identified the main entities of the system. Then, according to these entities they grouped the diagrams (e.g., all diagrams describing the "Message" object were grouped together).
- Informative vs. less-informative diagrams: Some diagrams were valued by students as having a major contribution to system comprehension (such as class diagrams), while other types of diagrams (notably collaboration diagrams), have, according to students' opinions, only a minor contribution to system comprehension.

Looking at the second, third and fourth methods of categorization, it can be noted that the systems were examined from dynamic and/or static perspectives: The second categorization examines the diagrams from both perspectives; the third categorization examines the diagrams from a dynamic perspective; and the fourth examines the diagrams from a static perspective. As mentioned above, this shows that the different teams preferred different perspectives.

b. Pivotal diagrams:

This finding presents three diagrams that the students kept returning to in the process of reviewing the diagrams. In some way or another, these three diagrams present the system in an abstract manner. Thus, these diagrams provided the students with a global view of

the system at times when they felt overwhelmed with details. Specifically, the three diagrams that served as pivotal diagrams were (as presented in the Appendix):

- Use case diagram
- Subsystems in sound recorder (a package diagram)
- MenuUserMode statechart

Once again, it can be observed that the students used both a static perspective (the package diagram) and a dynamic perspective (the use-case diagram and the statechart).

c. UML as a multifaceted perspective of a system:

This observation focuses on the diagrams reviewed by the students in order to retrieve relevant information. Table 3 presents the number of "visits" made by each team in each type of diagram. This table does not include the general diagrams (such as use case and deployment diagrams) and thus refers to only 23 of the 27 research diagrams.

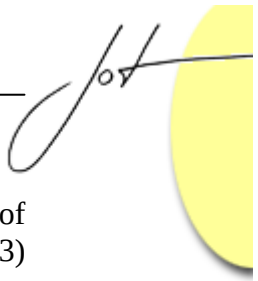
Table 3. The number of "visits" to each type of diagram per team.

Diag. Type	Team #1	Team #2	Team #3	Team #4	Team #5	Total "visits"	No. of given diagrams
Class	5	10	8	7	18	**48**	8
Collaboration	2	2	10	3	3	**20**	3
Sequence	4	2	7	3	10	**26**	4
State	10	8	9	9	16	**52**	8
Total	**21**	**22**	**34**	**22**	**47**	**146**	**23**

Since there were a different number of diagrams in each diagram type (right-hand column), comparison of the absolute number of "visits" to each type of diagram would not be appropriate. Furthermore, since the total number of "visits" made by each team was different, a comparison of the number of "visits" made by the different teams to each type of diagram is not appropriate either. In order to be able to make these comparisons and reach reasonable conclusions, the data must first be normalized. Normalization was performed in the following manner (Formula 1): first, the number of "visits" per team per diagram type (each cell in the table) was divided by the total number of "visits" made by the team; second, the resulting fraction was divided by the ratio between the number of diagrams of the specific type and the total number of diagrams.

Formula 1: Data normalization.

$$\frac{\text{No. of "visits" per team per type}}{\text{Total no. of "visits" per team}} \Big/ \frac{\text{No. of diagrams per type}}{\text{Total no. of diagrams}}$$



For example, the normalization of the upper left-hand cell in Table 3, the number of "visits" made by Team #1 to the Class diagrams, is performed as follows: (5/21) / (8/23) = 0.68. Table 4 presents the outcome of this normalization.

Table 4. Normalized data per team.

Diag. Type	Team #1	Team #2	Team #3	Team #4	Team #5
Class	0.68	1.30	0.67	0.91	1.10
Collaboration	0.73	0.69	2.25	1.04	0.49
Sequence	1.09	0.52	1.18	0.78	1.22
State	1.36	1.04	0.76	1.18	0.97

Table 4 shows that, in fact, there is no one dominant type of diagram that *all* teams relate to more than others. The conclusion from this finding is that *different students prefer different perspectives in the process of UML diagram comprehension*. Interestingly, although different students preferred different types of diagrams while revealing the information, no significant differences were observed in the students' descriptions of the system. This could imply that *consistent information was reflected in the various diagrams and was revealed independent of the specific strategy adopted by the students* (at least with respect to the specific set of diagrams presented to the students in this study).

Another interesting outcome, particularly with regard to the above, is the normalized total number of "visits" to each diagram (Table 5).

Table 5. Total no. of "visits" by all teams – normalized data.

Diag. Type	Total no. of "visits"	No. of diagrams	"Visits" per diagram (=Total "visits"/No. of diagrams)	"Visits" per diagram per team
Class	48	8	6.00	**1.20**
Collaboration	20	3	6.66	**1.33**
Sequence	26	4	6.50	**1.30**
State	52	8	6.50	**1.30**

From Table 5 we may conclude that, although each team focused on, and found different types of diagrams to be more useful than others, the total number of "visits" made by the five teams to the different types of diagrams is similar. In other words, although different personal preferences surfaced during the process of UML diagram comprehension, in an overall perspective, all of the diagrams were in fact equally important and useful.

These observations are supported by results obtained from the questionnaire (Table 6). As can be observed, the four diagram types were ranked in the same order in both parts of the questionnaire. More specifically, the order in which the four diagram types were ranked (from high to low importance) was: class, sequence, collaboration and

finally, state. The differences, however, in the level of importance were smaller in the comprehension section of the questionnaire compared to the development section. Based on the answers to the open question at the end of the questionnaire about the perceived reasons for the differences between the two sections, our impression is that this reduction can be explained by the fact that the success of the comprehension phase is highly dependant on the *integration* of information retrieved from the different diagrams. At the same time, in the development phase, especially when performed by a single developer, there is less need to integrate information from different perspectives, hence the greater differences. It can thus be concluded that the multifaceted description provided by the UML diagrams not only contributes to software development, but that its contribution is increased in comprehension tasks.

Table 6. Questionnaire results – relative importance of the four diagrams (from high to low).

Diagram Type	Development phase	Comprehension phase
Class	2.76	3.19
Sequence	3.17	3.51
Collaboration	4.46	4.12
State	5.32	4.92

4 DISCUSSION

Our observations show that UML was utilized by the students as a multifaceted expression tool. The way in which the different teams sorted the diagrams in preparation for the comprehension process, the different pivotal diagrams that they leaned on, and the number of "visits" made to each of the different diagram types, all indicate that the process of comprehension and information extraction from UML diagrams varies between different people. It was also found that, when taken together, no one diagram type was globally less or more important than the others for the performance of the comprehension task. In other words, the differences in preference between the various teams canceled out each other.

The above conclusions are based on the work of senior computer science students. In the future, we intend to conduct a similar study on senior computer professionals. In parallel, research is being conducted by the authors on the construction of UML diagrams.

ACKNOWLEDGEMENTS

We would like to thank Professor Uri Leron from the Department of Education in Technology and Science for his comments on earlier versions of this paper and to Dr. Yossi Gil from the Department of Computer Science for his cooperation and good will.

REFERENCES

[Bass1999] Bassey, M., "Case study research in educational settings. Chapter 7: Methods of enquiry and the conduct of case study research", UK: Open University Press, 1999.

[Bent1986] Bentley, J., "Programming pearls", *Communications of the ACM*. 29(6) (1986), 26-28.

[Booc1999] Booch, G., "UML in Action", *Communications of the ACM*. 42(10) (1999), 26-28.

[Broo1983] Brooks, R., "Towards a Theory of the Comprehension of Computer Programs", *International Journal of Man-Machine Studies*. 18 (1983), 543-554.

[Davi1993] Davies, S. P., "Models and Theories of Programming Theory", *International Journal of Man-Machine Studies*. 39 (1993), 237-267.

[Fjel1983] Fjeldstad, R. K. and Hamlen, W. T., "Application Program Maintenance Study – Reports to Our Respondents", In Parikh, G. and Zvegintzov, N. (eds.), "Tutorial of Software Maintenance", Silver Spring, MD: *IEEE Computer Society Press*, 1983.

[Fran2001] Francel, M. A. and Rugaber, S., "The Value of Slicing while Debugging", *Science of Computer Programming*. 40 (2001), 151-169.

[Kobr1999] Kobryn, C., "A Standardization Odyssey", *Communications of the ACM*. 42(10) (1999), 29-37.

[Knut1984] Knuth, D. E., "Literate Programming", *The Computer Journal*. 27(2) (1984), 97-111.

[Leto1987] Letovsky, S., "Cognitive Processes in Program Comprehension", *The Journal of Systems and Software*. 7 (1987), 325-339.

[Litt1987] Littman, D. C., Pinto, J., Letovsky, S. and Soloway, E., "Mental Models and Software Maintenance", *The Journal of Systems and Software*. 7 (1987), 341-355.

[OMG1999] OMG Object Management Group, "UML Notation Guide", Version 1.3, 1999.

[Palt1999] Paltor, I. P. and Lilius, J., "Digital Sound Recorder: A Case Study on Designing Embedded Systems Using the UML Notation". (1999), URL: http://users.evitech.fi/~tk/rt_design/uml_sound_rec.pdf.

[Penn1987] Pennington, N., "Comprehension Strategies in Programming". In Olson, G., Sheppard, S.B., Soloway, E. (eds.), *Empirical Studies of Programmers*, Albex, Norwood, N. J., 1987, 100-113.

[Schö1983] Schön, D. A., *The Reflective Practitioner*, BasicBooks, 1983.

[Solo1984] Soloway, E. and Ehrlich, K., "Empirical Studies of Programming Knowledge", *IEEE Trans. Software Engineering*. 10(5) (1984), 595-609.

[Vans1999] Vans, A. M., von Mayrhauser, A. and Somlo, G., "Program Understanding Behavior During Corrective Maintenance of Large-Scale Software", *Int. Journal Human-Computer Studies*. 51 (1999), 31-70.

About the authors

Irit Hadar is a doctorate student in the Department of Education in Technology and Science of the Technion – Israel Institute of Technology. Both her Master and Bachelor degrees are from the Faculty of Industrial Engineering and Management of the Technion. Her doctorate thesis examines how software development experts understand concepts and principles in Object Oriented Design. UML is used in her research as an expression tool. She can be reached at hadari@techunix.technion.ac.il.

Orit Hazzan is a senior lecturer in the Department of Education in Technology and Science of the Technion - Israel Institute of Technology. Her research focuses on teaching human aspects of software engineering. Specifically, she examines the application of the studio teaching method into software engineering education and the teaching of software development methodologies in undergraduate curricula. She can be reached at oritha@techunix.technion.ac.il.

Appendix: Three Pivotal Diagrams

Source: Paltor and Lilius, 1999.

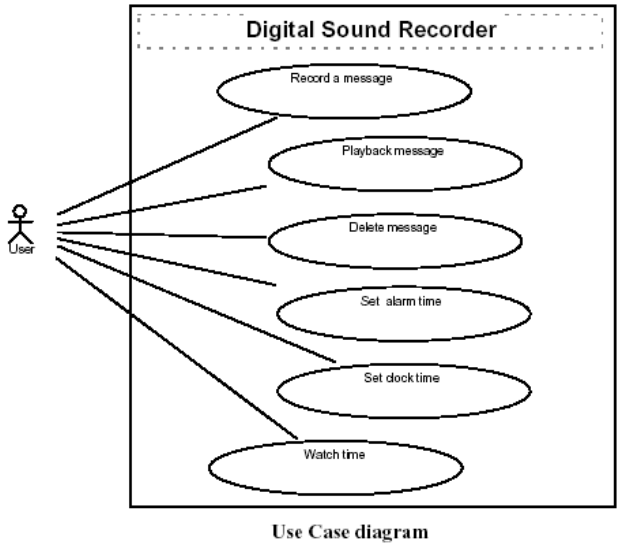


Fig 1. Use Case Diagram used in the task

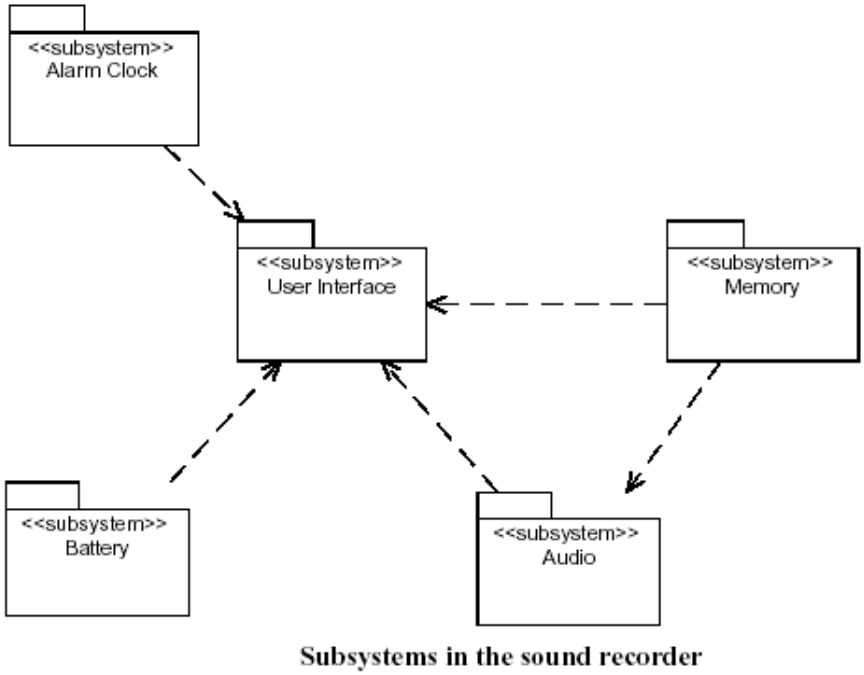
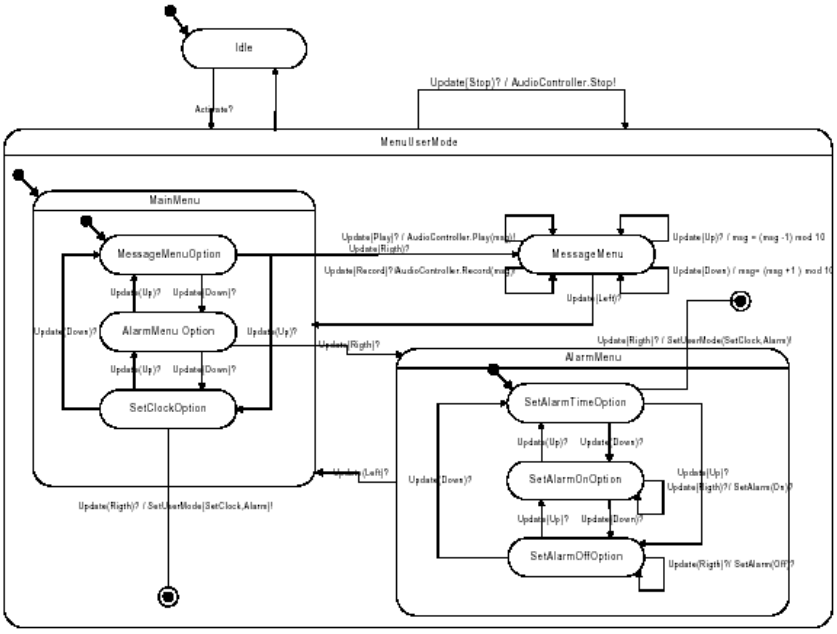


Fig 2. Package Diagram used in the task



MenuUserMode statechart

Fig 3. Statechart describing the user menu used in the task