13th EAEEIE Annual Conference on Innovations in Education for Electrical and Information Engineering (EIE), York 8-10April 2002, CD-ROM, ISBN-85911-008-8, paper 71EAEEIE2002.pdf

Teaching object oriented software engineering with UML

Zbigniew MROZEK[1],Bogumiła MROZEK[1], Osei ADJEI[2]

[1]*Cracow University of Technology (Politechnika Krakowska) ,*
PL 31-155 KRAKÓW,Poland, Warszawska 24,
zbigniew.mrozek@pk.edu.pl, bmrozek@usk.pk.edu.pl
[2]*University of Luton, Park Square, LUTON, Bedfordshire, LU1 3JU, England,*
osei.adjei@luton.ac.uk;

Abstract

The Unified Modelling Language (UML) is a language that helps to visualize, design and document models of software systems. It represents a collection of the best engineering practices that have proven successful in modelling large and complex systems. UML is widely used in designing large and reliable software systems required by banks and other corporate bodies. Students are encouraged to work in groups to learn the terminology, notation and use of UML and by so doing learn the task of communicating with clients and with other members of the project team. This effectively leads to the students' understanding of main factors associated in the project's success or failure and complexities involved in software development and project management.

Keywords
UML, software development, project management, CASE tool, MATLAB, Simulink, Stateflow diagram

1 Introduction.

A university professor should consider some quality objectives [5] according to the mission and circumstances of the educational institution that he/she belongs to. Such objectives must include:

1. student satisfaction consistent with professional standards,
2. continuous improvement of service,
3. giving consideration to the requirements of industry, commerce and the public sector,
4. efficiency in providing the service.

On the other hand, he/she must be able to identify clear opportunities for the delivery and improvement of quality lectures. This means that he/she must give interesting, useful, clear and state of art lectures on a chosen topic. This is very important when teaching UML programming since fast and continuous progress in extending UML specification is to be targeted. The current UML version specification is 1.4 and proposals for version 2.0 are under discussion. Therefore, an intended lecturer for a UML course must verify related materials (handouts, slides, PowerPoint presentations) and books before and during the semester in which the subject is being taught.

2 What is UML language

The Unified Modeling Language provides the means to visualize, document and model software applications before coding. It is independent of any programming language that will be used for coding the final application software. UML represents a collection of the best engineering practices that have proven successful in the modeling of large and complex systems [1-4, 6, 8, 10]. Many successful attempts have been considered to extend the application of UML to areas beyond informatics [6,8]. The advantage of UML over other systems analysis tools is that, it reveals gaps and inconsistencies in the requirement's specification at very early

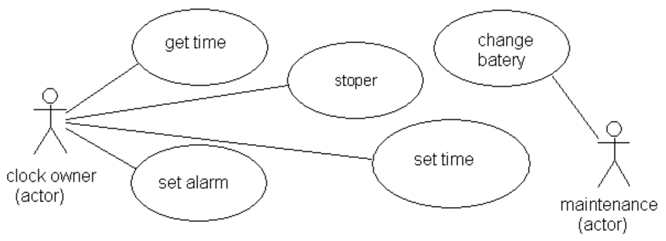

Figure 1: Use Case Diagram

stages of the design as well as providing the ease of understanding and the ability to modify visual modeling diagrams.

UML is derived from three other products: ***OMT** (object modelling technique)* by James Rumbaugh, ***Booch method*** by Grady Booch and ***OOSE** (object oriented software engineering)* by Ivar Jacobson. Some ideas of SDL (*Specification and Design Language, 1976 CCITT*) and E-R (*Entity Relationship*) model of databases are also taken into account. Since its introduction, UML had undergone continuous improvements several times until version 1.3 was proposed as the accepted standard in year 1999 [4].

Any complex system can be presented by a set of nearly independent views of a model. A single view is not sufficient. *Use case* and *class* diagrams, an example of which is shown in figure 1 (these are described later), are used in all UML supported projects. The choice of diagrams created depends on how a problem is to be solved. In addition to *use case* and *class* diagrams one can create behavioural diagrams:

- *state-chart* diagram
- *activity* diagram
- interaction diagrams*: sequence* and *collaboration*

and implementation diagrams:

- *component* diagram
- *deployment* diagram

It is interesting to note that some CASE tools supporting UML design may also offer their own extensions and diagrams which are not included in current UML specification [11-12].

2.1 Study program quality plan

Prerequisites for a UML module must include some practical knowledge of any object oriented language such as Java or C++. The balance of knowledge between C++, Java and software packages (e.g. MATLAB) and UML is an important factor to be taken into consideration when planning a curricula. A modular credit scheme with partial or full freedom in choosing modules may be essential to attract students and to get feedback on actual students' preferences [5]. The module should be validated using officially approved validation procedures. Once objectives are agreed, proper activities should be defined and documented.

The objective of a UML module is to provide the student with the visualization and understanding of software development skills that are achieved by using modern **CASE** tools. Before the module starts, some resources such as free and almost unlimited access to computer laboratory and installed **CASE software** must be guaranteed or made available to students. Regardless of these resources, students registered on the UML module may be asked to seek alternative CASE software advertised on the Internet. Students may install evaluation copies of such software and use them within limited period of time. Comparing different tools and discussing different approaches of solving problems help students to achieve skills in UML and to understand limits supported by different tools.

2.2 Class and object diagram.

When *use cases* or *scenarios* are analyzed, object activities are described although classes are not defined at all. This may be very strange to students with some background knowledge in C++ or Java languages and who, in general, accept that an object is an instance of class. Here in visual programming using

UML a class may be defined and drawn later, as generalization of chosen similar objects. A class diagram may contain classes and objects. If there is no other class in such a diagram, the diagram is named an object diagram.

There is no officially recognized methodology to identify object and classes. Therefore, different programmers may define a different set of classes and objects for the same problem.

It is not a good idea to try and design a complete class or object diagram when *scenarios* and *use cases* are ready. This is because some changes in object hierarchy, naming, methods and attributes may be inevitable, when other diagrams are designed. This is true especially when sequences or state diagrams are under preparation. In general, when a good case tool supporting UML programming is used, the class diagram is corrected automatically when objects or class names, types, methods, attributes or parameters are changed in other diagrams.

2.3 Behavioural diagrams

There is no need to use all existing types of behavioral diagrams in applications. That is, it is sufficient to have a deep knowledge of a small subset of carefully chosen diagrams. As Brugge [3] points out; “You can model 80% of most problems by using about 20% UML”. It means that a UML module should present only a part of carefully selected elements of the UML language. In the author’s opinion, *use case diagrams, scenarios, class* and *object diagrams,* two or three behavioural diagrams (e.g. *sequence* and *state), component* and *deployment diagrams,* OCL (*Object Constraint Language*) should be considered when designing a UML module.

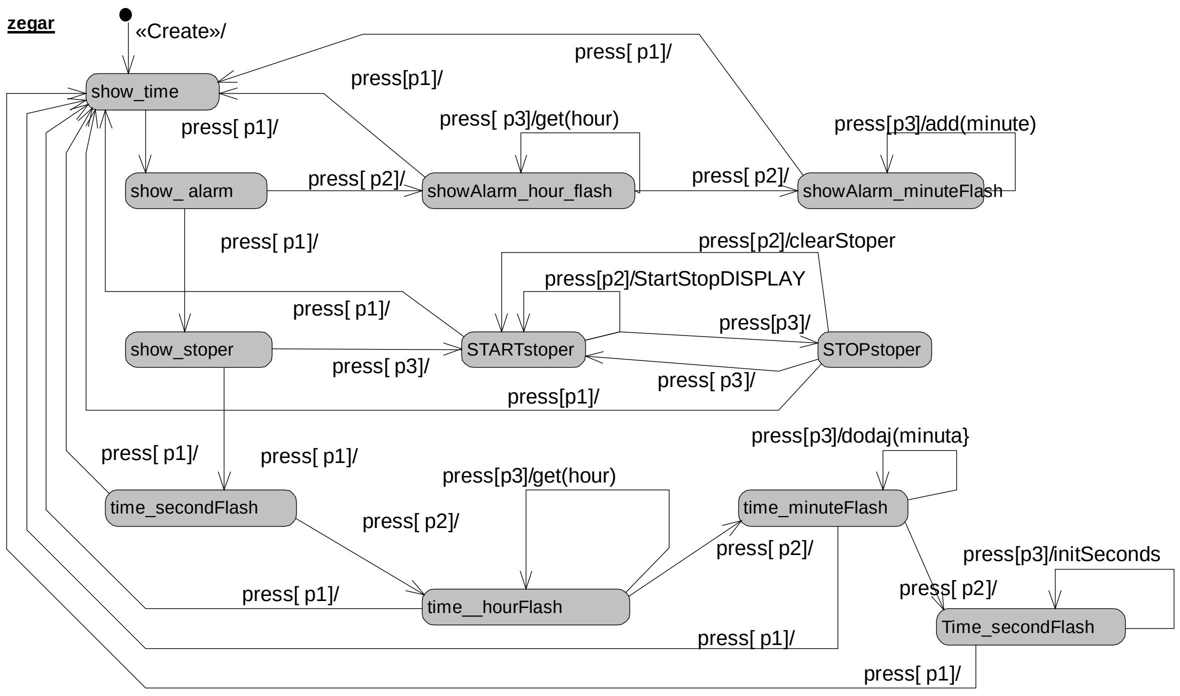

Figure 2: State Diagram example

2.3.1 Sequence diagram

In the preparation of sequence diagrams one can verify requirement’s specification and scenarios against omissions and inconsistencies. Similarly, one can verify existing classes and objects, in case class diagrams have been created before. Missing classes and objects should be defined at this stage. Methods and attributes defined here should be identified automatically by the CASE package and used immediately to update existing classes and objects.

2.3.2 State diagram

If an object's behavior is more complicated, a sequence diagram is not suitable and thus a state-chart diagram, shown in figure 2, should be used. By using a brainstorm session, a group of students may:

- identify major categories of possible events, including actors, methods and procedures involved,
- define the effect of each condition or event clearly and concisely, and
- begin to construct the diagram using a box to describe new state and transitions to connect different states.

When building diagrams, students will verify if already defined objects are useful for tasks and services described with the new state or sequence diagrams. They have to decide what methods (its name, type and parameters) and attributes (its name and type) are needed. Most CASE tools keep a database with objects and attributes defined within the project. In case of any inconsistency, the student is immediately alerted to correct the error. This speeds up programming and helps to avoid many errors

2.4 Brainstorming

Brainstorming is used to identify possible solutions to problems and potential opportunities for improvements. This technique is used for tapping creative thinking of a team to generate and clarify a list of ideas, problems and issues. We use brainstorming to find all *use cases* and *actors* necessary to build a *use case diagram*. Brainstorming is very useful to identify states and transitions for state diagram. There are two phases in the brainstorming procedure:

- during the **generation phase**
  - the purpose (target) of the brainstorming session is clearly stated
  - each team member takes a turn in a sequence, stating a single idea
  - where possible, new ideas are build on others ideas
  - all ideas are recorded and should be seen by all the participants (using whiteboard or overhead is recommended)
  - at this stage, ideas are neither criticized nor discussed
  - the process continues until no more ideas are generated.
- the **clarification phase**
  - list of ideas should be reviewed to make sure that each person understands all the ideas
  - evaluation of ideas will occur after the brainstorm session is completed.

2.5 Presentation tools

Power Point is a very useful tool at any level of the preparation and implementation phases. **Netscape** and **Internet Explorer** (using HTML language) are completely free for educational purposes and seem to be even better than MS Word for documenting, network communication and data collection phases. Power Point is widely used as presentation tool at any level of the preparation and implementation phases. HTML presentation may be used as well. **MATLAB (with SIMULINK and STATEFLOW tools)** can be used for the visualization and animation of state diagrams.

3 Conclusions

The UML has the advantage that it reveals gaps and inconsistencies in the requirement’s specification at earlier stages of software design, as well as making it easy to understand and modify visual modelling diagrams. Unification and precision of notation is important for large and interdisciplinary projects. UML tools keep track of used class and object names, its attributes and methods. Users may transfer already defined classes and other elements between different diagrams and reuse them. This accelerates work progress and helps to keep all parts of project in a consistent manner.

Using commercially available specialised CAD tools and CASE packages visual UML programming may greatly improve productivity of a software design team by cutting down development time and improving final product quality (in accordance with ISO 9000 standards). UML is a dedicated software design language for large software projects, and its applicability can be extended to other domains.

4 Acknowledgements:

Authors wish to express their gratitude to *ARTiSAN Software Tools, Inc* (GB); *Rational Software Corporation* (USA) and *Premium Technology Sp zoo* (Poland) for free evaluation license for following software: *Real-time Studio, Rational Rose Suite* and *Rational Rose RT*.

References

[1] Arief L.B, Speirs N.A, *Automatic generation of distributed system simulations from UML* in Modelling and Simulation, A tool for next millennium*, 13-th European Simulation Multiconference*, Vol.II pp 507-509, Warsaw, June 1-4,1999.

[2] Booch, G. Rumbaugh, J. Jacobson I *The Unified Modelling Language User Guide*, Addison Wesley, 1999

[3] Bruegge B, Dutoit A, *Object-Oriented Software Engineering: Conquering Complex and Changing Systems*, Prentice-Hall, 1999.

[4] Kobryn Cris, *UML 2001 A Standardization Odyssey*, Communications of the ACM Vol.42,No.10, pp.28-37, October 1999.

[5] Mrozek Z , Adjei Osei, Mansour Ali Quality Assurance In Higher Education, Proc. of 4-th Int. Conf. Computer Aided Engineering Education, ed. M. Chrzanowski, E.Nawarecki (CAEE'97), vol.2, pp.156-164, AGH Krakow.

[6] Mrozek Z, *UML as integration tool for design of the mechatronic system,* in Second Workshop on Robot Motion and Control, pp 189-194, ed. Kozlowski K, Galicki M, Tchoń K, Oct 18-20, 2001, Bukowy Dworek, Poland.

[7] Mrozek B,. Mrozek Z, MATLAB 6, poradnik użytkownika, ISBN 83-7101-449-X, PLJ Warszawa 2001.

[8] Mrozek Z, Metodyka wykorzystania UML w projektowaniu mechatronicznym, pp.25-28, Pomiary Automatyka Kontrola 1, 2002.

[9] OMG Unified Modeling Language Specification (draft), Version 1.4, February 2001. and other OMG (Object Management Group) standards, http://www.omg.org:

[10] Pristley, Practical object-oriented design with UML, 2000.

[11] Rational Rose Suite, Rational Rose RT and other software from Rational Software Corporation.

[12] Real-time Studio, ARTiSAN Software Tools, Inc. July 2001.