StudentUML: An Educational Tool Supporting Object-Oriented Analysis and Design

Ervin Ramollari [1] and Dimitris Dranidis [2]

[1] South East European Research Centre (SEERC)
17 Mitropoleos Str., 54624 Thessaloniki, GREECE
erramollari@seerc.info

[2] Computer Science Department
CITY LIBERAL STUDIES
Affiliated Institution of the University of Sheffield
Tsimiski 13, 54624 Thessaloniki, GREECE
dranidis@city.academic.gr

Abstract

The Unified Modeling Language (UML) is commonly used in Computer Science curriculum in order to teach object oriented analysis, design and programming. In this context, UML CASE tools are useful to assist in modeling and automating routine tasks. However, available tools are generally intended for use by professional developers to improve productivity and are not suitable for educational purposes. Tools are generally difficult to learn and use, are confusing to beginners, and ignore educational aspects. Existing educational tools also have shortcomings, which are discussed in detail. Finally, we present a new UML tool, *StudentUML*, which specifically addresses these issues. Emphasis is placed on the educational nature, simplicity, and ability of the tool to ensure correctness and consistency of the models.

Keywords: Education, Teaching, UML, CASE tools, Object-oriented analysis and design.

1. Introduction

Modeling is a powerful technique that helps in managing the complexity of software systems and communicating ideas. The dominating modeling notation in object oriented development is the Unified Modeling Language (UML) [OMG (2006a)], which has become the de facto standard in the industry and academia. UML, by means of diagrams, describes the static structure of software in terms of objects and classes, the dynamic behavior in terms of object interactions, and the functionality in terms of use cases.

Besides a notation, developers need assistance from Computer-Aided Software Engineering (CASE) tools. Such tools emerged as an alternative to software engineering techniques to improve on productivity [Mynatt et al (1989)]. They are also used in academic environments for teaching object orientation and the UML language itself. A number of CS courses use UML as the modeling language of choice. Instructors are faced with the decision of choosing a suitable tool to support such courses. However, most of the existing tools are designed for use by professionals and not by students [Auer et al (2003), Buck et al (2000), Crahen et al (2002), Turner et al (2005), Iivari (1996)]. They are usually overloaded with features and UML syntax, thus causing confusion among beginners. Since their aim is to improve productivity, professional tools assume their users have adequate experience, and disregard educational features.

In this paper we introduce a new tool, StudentUML, which specifically addresses the needs of students. Drawing from our experience of teaching object oriented analysis and design and UML at both undergraduate and postgraduate level, we have recognized the benefits of a tool tailored to the needs of the taught modules, which could greatly assist students in their learning process. StudentUML is aiming to be easy to learn and use through its simplicity, to maintain correctness and consistency of models, and to support a student software development process.

This paper first reviews general UML CASE tools and then focuses on educational tools. It critically evaluates these tools and presents the motivation for a new tool that is specifically designed for educational purposes. Next, the general aims and requirements are defined, and the paper proceeds with a short demonstration of StudentUML. Conclusions and further research directions close the paper.

2. Background

2.1 UML CASE Tools

Numerous CASE tools supporting UML modeling have been developed by different companies, which are commercial, free, or open source. The interested reader can refer to [OD (2005a)] and [Godfrey (2006)] to view a comprehensive list of tens of existing UML tools, along with a short description and a link to the official web site for each one of them. In [OMG (2006b)], OMG publishes a list of companies that produce or distribute UML 2.0 compliant tools.

There is a wide spectrum of UML tools that serve different purposes, have different degrees of sophistication and portability, and incorporate various features. Smith (2004) roughly divides UML-based tools into three categories: UML Drawing Tools, UML Code-Centric Tools, and UML Framework Tools. In brief, UML drawing tools are those that enable drawing of diagrams with limited restrictions [Smith (2004)]. An example is Microsoft Visio. UML code-centric tools go one step further in

sophistication and enable connection between static aspects of the UML model and source code, usually through forward and reverse engineering [Smith (2004)]. Most tools fall into this category. Two well-known examples are IBM Rational Rose, and Borland Together. UML framework tools are the most sophisticated and provide a complete code generation, including behavioral aspects of the UML model [Smith (2004)]. Examples include IBM Rational Rose Real-Time, and I-Logix Rhapsody. Generally speaking, when moving from drawing tools to framework tools, flexibility decreases, while automation increases [Smith (2004)].

Features that modern UML CASE tools commonly support include: drawing diagrams, repository support, forward/reverse engineering, round-trip engineering, navigation, multi-user support, integration with other tools, exporting models to other formats, versioning, printing support, scripting, pick lists for classes and methods, and many others [Eriksson et al (1998), OD (2005a)]. On the other hand, they differ regarding the portion of UML syntax and diagrams that are supported, compatibility with UML 2.0, portability, programming language or IDE dependence, proprietary extensions to standard UML, and so on.

2.2 Educational UML Tools

A number of studies [Burton et al (2004), Mynatt et al (1989), Ho (1992)] have shown that the use of modeling CASE tools in education facilitates teaching of concepts and improves productivity. We found that a very small number of UML tools are available for educational purposes. Those that exist are usually personalized for specific usages or academic programs. Below, we briefly present the following tools: QuickUML, minimUML, UMLet, and Ideogramic UML™.

QuickUML [Crahen et al (2002), Alphonce et al (2003)] is a simple UML drawing tool written specifically for beginners in object oriented programming. It supports only design class diagrams, and only one at a time. It allows some limited forward engineering to Java and C++ code, and reverse engineering from Java code to class diagrams. QuickUML lacks validation and consistency maintenance capabilities, so that the user is allowed to draw incorrect diagrams. It does not support working with whole development projects, which consist of a number of diagrams, as most modern UML tools do. The main strengths of QuickUML are its interface simplicity and ease of learning.

MinimUML [Turner et al (2005)], similarly to QuickUML, follows a minimalist approach [Carroll (1997)] by covering a small subset of the UML notation. Its authors regard this subset as sufficient for the purpose of introductory OO classes. MinimUML has support only for design class diagrams. Diagrams consist of two basic elements: classes (design classes or interfaces), and connections (associations, aggregations, realizations), which are treated in a generic way. The user is allowed to specify their types in later phases during an iterative process. In addition, the tool

introduces some other features that facilitate its usability, such as multiple selection, undo/redo, cut and paste, flexible printing, and drag and drop. Finally, it provides limited support for forward engineering to Java and C++. This tool has a number of strengths that make it more appropriate for students. It has a simple and intuitive user interface through which it is easier to learn and use. Also, some features, such as labels explaining the type of UML elements, user notations, flexibility, and generic treatment of classes and connections, make this tool appealing to beginners. However, like QuickUML, minimUML does not support whole development projects and lacks correctness and consistency capabilities.

UMLet [Auer et al (2003)] is a simple UML tool with a number of characteristics that are useful for teaching and learning object orientation and the UML notation. It covers a significant portion of the UML notation found in different types of UML diagrams, but does not distinguish between the diagram types. It has a large and intuitive drawing palette where various UML elements are displayed in the same way they appear in the diagram. To increase drawing speed and avoid user distraction, dialog boxes for editing UML constructs are avoided and replaced with a separate text panel in the application window, where UML constructs are specified in the form of simple markup text. These features make this tool easy to use for quickly drawing diagrams, although not as easy to learn. UMLet does not go beyond mere sketching of diagrams, however. It gives the user the flexibility to draw any diagram consisting of any of the supported UML constructs, but without checking for consistency. The tool does not distinguish between the different types of diagrams, and elements found in different UML diagram types can be put in a single one. Diagrams are sketched separately one at a time, and are not managed as part of a whole development project. Finally, the tool does not have support for forward or reverse engineering.

Ideogramic UML [Hansen et al (2002), Ideogramic (2006)] is a more sophisticated educational UML tool from Ideogramic. Unlike the previously discussed tools, this one is commercial, not free, and not open source. Its scope is limited to design class diagrams that are drawn one at a time and not as part of development projects. Ideogramic UML introduces the useful paradigms of user gestures and collaborative modeling. Collaboration is achieved with electronic whiteboards or light pens where users draw objects that are intelligently recognized and transformed to UML constructs. This capability engages students in active learning through visual collaboration. Ideogramic UML has a simple and intuitive interface. The complexity of the user interface and the number of menus is drastically reduced through the use of context-sensitive pie menus that appear in the drawing area. As a result, the tool is very easy to use and allows quick drawing of diagrams. However, the use of gestures requires learning of different gestures corresponding to different UML elements, which makes the tool learning curve slightly steeper. Finally, Ideogramic UML does not have any support for forward or reverse engineering.

Table 1 summarizes and compares the most relevant features of these four educational tools.

Table 1. *Comparison of Four Available Educational UML Tools*

	QuickUML	**minimUML**	**UMLet**	**Ideogramic UML**
Support for projects	-	-	-	-
Class Diagrams	●	●	●	●
Interaction Diagrams	-	-	-	-
Use Case Diagrams	-	-	●	-
Diagram Correctness	-	-	-	Limited
Project consistency	-	-	-	-
UML syntax	Limited	Limited	Most	Most
Forward engineering	● (Java/C++)	● (Java/C++)	-	-
Reverse/Round-trip engineering	● (Java)	-	-	-
Ease of learning	Easy	Easy	Moderate	Moderate
Ease of use	Easy	Easy	Easy	Easy
Cost	Free	Free	Free	Not free
Source code	Available	Available	Available	Unavailable

3. Motivation behind StudentUML

Most available UML-based tools are designed for professional use. Smith (2004) advocates the use of industry standard *commercial-off-the-shelf* (*COTS*) tools in undergraduate courses. According to him, it is to the benefit of the student to have hands-on experience with the tools of the trade. In addition, many vendors offer their tools at a discount or for free for academic programs [Smith (2004)]. However, a number of other studies have shown that professional tools are too complex to be suitable for educational purposes [Auer et al (2003), Buck et al (2000), Crahen et al (2002), Turner et al (2005), Iivari (1996)]. One reason is that they usually aim to be fully compliant with the latest UML syntax and to offer a large number of features [Flint et al (2004), Auer et al (2003), Crahen et al (2002), Turner et al (2005)]. According to some authors [Flint et al (2004), Turner et al (2005), Carroll (1997)], only a simplified subset of UML notation is needed to teach object oriented software engineering courses. This subset includes the basic elements of class diagrams [Turner et al (2005)], and interaction diagrams. Also, a large number of features and a cluttered interface confuse students, who frequently have to face and ignore functionality that is not needed [Hansen et al (2002)]. Therefore, professional tools have steep learning curves and are unsuitable for use in education.

In addition, some useful educational aspects including consistency checking, inter-diagram conversions, educational hints, illustration of learned concepts, and so on, are usually ignored as they are not required in professional development. Other issues include high tool costs, proprietary UML syntax, high requirements on computing resources, programming language dependency, and others [Auer et al (2003)].

Educational tools, on the other hand, have some useful characteristics, including ease of learning, ease of use, limited subset of UML, limited illustration of learned concepts (minimUML), collaborative learning with gestures (IdeogramicUML), as well as basic forward and round-trip engineering. However, they have serious shortcomings. None of them supports development projects with several diagrams, interaction diagrams, consistency and validation capabilities, and important educational features.

Relating to our past experience, we have identified other limitations with available tools, and student needs that are not fulfilled by existing professional or educational tools. In our curriculum a number of object orientation modules are taught, based on UML, both at the undergraduate and postgraduate level. In "Object-Oriented Programming" taught at the first undergraduate year, students are introduced for the first time to objects. In this module a tool is required that is simple to learn and use, without distracting students with complex UML syntax and functionality that is not needed. Two other modules – "Object Oriented Analysis and Design" (undergraduate, 3rd year) and "Software Development" (postgraduate) impose other needs. Students of these modules are exposed to iterative development of projects consisting of different UML diagrams during design phase. Thus, in support of these courses a tool is required that can handle complete development projects, maintain a consistent UML model project-wide, and support the process that is followed.

Regarding the process that students follow, we have found that existing tools hardly support it. Furthermore, existing tools do not distinguish diagrams to analysis and design diagrams, a necessity when teaching object-oriented analysis and design.

Another shortcoming with existing tools is their poor diagram validation and consistency checking capabilities. A UML project consists of a set of diagrams that together describe a single system. There may be overlap between different diagrams and the overlapping parts contain the risk of defects (inconsistencies). A study by Lange et al (2006) has shown that a large number of such defects are frequently not detected. A more alarming finding is that the defects that are not detected cause serious misinterpretations later in the development lifecycle [Lange et al (2006)]. We are familiar with such problems based on our experience. UML artifacts produced by students during coursework usually have significant inconsistencies. These result not only from poor group coordination and lack of attention, but also from lack of knowledge. It is therefore essential that such defects be discovered and reported by tools, and preferably be automatically repaired. Existing tools have little support for

detecting such inconsistencies, probably because they are not considered as errors. It would also be desirable to have inter-diagram conversion functionality and exporting of elements from one diagram to another, keeping in mind that diagrams of different types are related to one another.

As a result, we have recognized the need for a UML CASE tool tailored specifically to support object orientation modules. It is envisioned to assist in lectures and labs, or be used independently by students as a learning assistant.

4. Aims of StudentUML

The major aims of StudentUML result from the previous discussion of shortcomings of existing tools, and student and module needs. The most important aims considered in the development of StudentUML are its simplicity, correctness and consistency, and its support for the student software development process. All these features aim to promote the educational nature of StudentUML

4.1 Simplicity

One aspect of simplicity is the tool learning curve and ease of use. Towards this goal, the tool provides an intuitive and user-friendly graphical user interface with minimum cluttering. There are a limited number of menus and ways of interaction, so that the user can learn the tool and draw diagrams in a minimum amount of time. Tooltips explain the functions of different menus and buttons. Another aspect of simplicity is the amount and complexity of UML syntax that is supported. We have tried to limit this amount of UML to what is needed in the courses.

4.2 Consistency

Consistency is seen from two different points of view: internal diagram correctness and inter-diagram consistency.

Regarding diagram correctness, the tool restricts the user from specifying incorrect diagrams, without compromising flexibility. For instance, it does not allow the user to add incorrect relationships in the design class diagram, such as a class realizing another class instead of an interface, or an interface being an aggregation of classes.

Inter-diagram consistency, is a more important issue and at the same time neglected issue. We address it by providing automated consistency checking between different types of diagrams (for example, interaction diagrams against class diagrams). The results of validation are given in the form of errors and warnings, and the user has the option of automatically repairing the errors. Besides the benefit of finding defects and correcting them, this feature has the desirable side effect of acting as a student advisor. It provides hints about how different diagrams are related to each other and how to avoid inconsistencies.

Another way to address this issue is by supporting automatic conversion from one type of diagram to another in the same project, by exchanging UML elements between related diagrams, or by promoting analysis elements to design elements.

4.3 Process Support

One main requirement is that StudentUML should support the process that is taught to or followed by students, although not enforce it.

The supported process is largely based on the process suggested by Craig Larman in his popular textbook [Larman (2005)]. In this process various UML diagrams are created iteratively. The developer starts with a use case diagram and use cases. From these, system sequence diagrams (SSDs) that correspond to the use cases are created. In parallel, domain analysis is performed, where concepts and relationships are identified and presented in conceptual class diagrams (CCDs). During a transition from OOA to OOD, the system messages in a SSD and the concepts from CCDs inspire the identification of objects and their collaborations in an interaction diagram, such as a sequence diagram (SD). Also, messages from SDs, and concepts and attributes from CCDs, serve to determine software classes with attributes and methods and their relationships in detail. They are represented in one or more design class diagrams (DCDs). Finally, software classes from DCDs and object collaborations from SDs serve to write code in an object-oriented programming language, such as Java or C++.

The features of inter-diagram consistency checking, repairing, and conversions assist the student process. Also, analysis artifacts should be promoted, on user's discretion, to design artifacts in the project. Finally, a desirable feature of StudentUML is forward engineering to code, and reverse/round-trip engineering with code-model synchronization.

5. Tool Functionality

StudentUML is written in Java™ and therefore is platform-independent. The graphical user interface provides a main project toolbar with functionality for opening, closing, and saving complete projects. The user can also export diagrams to different image formats. Each UML diagram is displayed in its own internal window, consisting of a simple drawing palette and a drawing area.

Also, the tool offers the option of automatically checking consistency between existing diagrams. For example, in Figure 1 the application requests input about which diagrams to validate in the context of SD-DCD validation. Next, the application presents the validation results in the forms of warnings and errors (Figure 2). Notice that the option of automatically fixing the errors (though not the warnings) is provided.

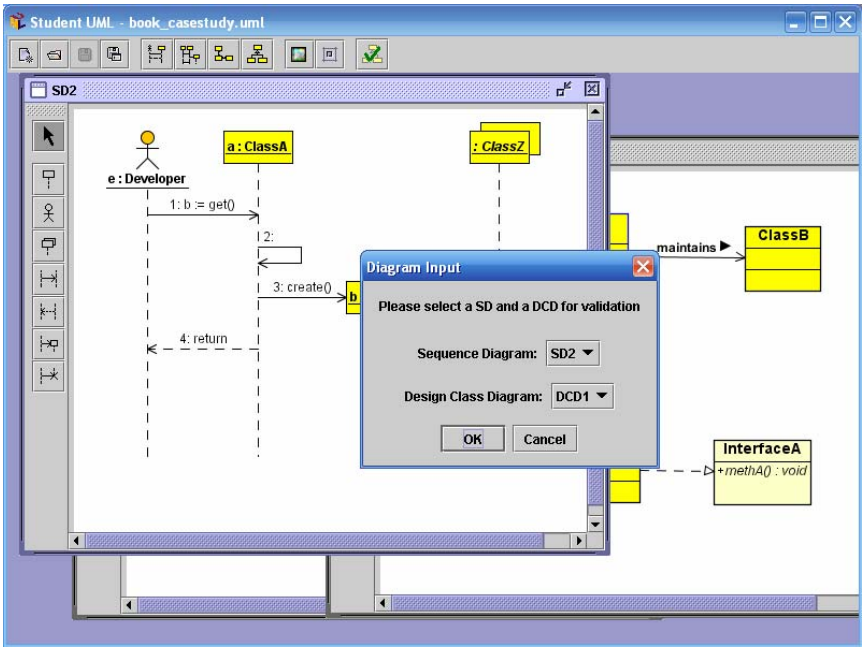

Figure 1. *Selecting a DCD and a SD for Inter-Diagram Validation*

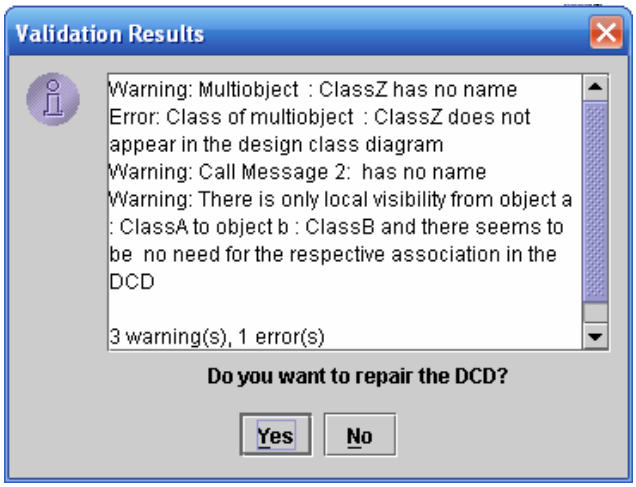

Figure 2. *Validation Results - Warnings and Errors*

So far we have implemented full support for drawing system sequence, sequence, conceptual class, and design class diagrams, project persistence and exporting, inter-diagram consistency validation, and repair of consistency errors.

6. Conclusions

In this paper we have identified a number of requirements that a UML tool suitable for educational purposes should satisfy. In general terms, this tool should be easy to learn and use by students, support correctness and consistency, provide educational features, and support the process taught to students. After a review of professional and educational tools that are available, we concluded that in general they do not satisfy these requirements and have a number of shortcomings. This motivated the need for a new tool, StudentUML, which we have developed. This report described the features, design, and functionality of this new tool.

Work is ongoing to make StudentUML a fully-fledged CASE tool for students. Features planned to be added are forward and reverse engineering (round-trip engineering). These features are important, since they illustrate the relationship between models and code and support iterative development. Another feature is diagram conversion and exporting of UML elements between related diagrams in the same project. Finally, some additional types of UML diagrams, such as use case diagrams, will be added in the future. Extension of the tool should be relatively easy, as the tool has been designed for modifiability and extensibility.

References

Alphonce C., Ventura P. (2003), "QuickUML: A tool to support iterative design and code development", in *OOPSLA '03*, Anaheim, California, USA.

Auer M., Tschurtschenthaler T., Biffl S. (2003), "A flyweight UML modeling tool for software development in heterogeneous environments", in *EUROMICRO'03*.

Buck D., Stucki D. J. (2000), "Design early considered harmful: Graduated exposure to complexity and structure based on levels of cognitive development", in *Proceedings of the 31-st SIGCSE Technical Symposium on Computer Science Education*, Austin, Texas, USA.

Burton P. J., Bruhn R. E. (2004), "Using UML to facilitate the teaching of object-oriented systems analysis and design", *JCSC*, vol. 19, no. 3, pp. 278-290.

Carroll J. M. (1997), "Reconstructing minimalism", in *Proceedings of the 15th annual international conference on Computer documentation*, Salt Lake City, Utah, United States, ACM Press.

Crahen E., Alphonce C., Ventura P. (2002), "QuickUML: A beginner's UML tool", in *OOPSLA '02*, Seattle, Washington, USA.

Eriksson H.-E., Penker, M. (1998), *UML Toolkit*, 1st edn, John Wiley & Sons, ISBN 0-471-19161-2.

Flint S., Gardner H., Boughton C. (2004), "Executable/Translatable UML in computing education", in *Proceedings of Sixth Australasian Computing Education Conference* (ACE2004), Dunedin, New Zealand.

Godfrey M. W. (2006), "My little UML (tools) page", http://plg.uwaterloo.ca/~migod/uml.html, [accessed 24/10/2006].

Hansen K. M., Ratzer A. V. (2002), "Tool support for collaborative teaching and learning of object-oriented modeling", in *Proceedings of ITiCSE '02*, Aarhus, Denmark.

Ho Y.-C. (1992), "To what extent will CASE tools assist users in the systems development – A case study in academic environment", in *Proceedings of the 1992 ACM SIGCPR conference on Computer personnel research*, Cincinnati, Ohio, USA, ACM Press, pp. 93-96.

Ideogramic (2006), "Ideogramic UML", http://www.ideogramic.com/products/uml/pervasive-uml.html, [accessed 01/03/2006].

Iivari J. (1996), "Why are CASE tools not used", *Communications of the ACM*, vol. 39, no. 10, pp. 94-103.

Lange C. F. J., Chaudron M. R. V. (2006), "Effects of defects in UML models – An experimental investigation", in *ICSE'06*, Shanghai, China.

Larman C. (2005), *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*, 3rd edn, Prentice Hall.

Mynatt B. T., Kleventhal L. M. (1989), "A CASE primer for Computer Science educators", in *Proceedings of the twentieth SIGCSE technical symposium on Computer science education*, Louisville, Kentucky, USA, ACM Press, pp. 122-126.

Object Management Group (OMG) (2006a), *OMG Unified Modeling Language Specification. Version 2.0*, Document formal/05-04-01, accessible at http://www.omg.org/technology/documents/formal/uml.htm.

Object Management Group (OMG) (2006b), "UML resource page", http://www.uml.org/, [accessed 24/10/2006].

Objects by Design (OD) (2005a), "Choosing a UML modeling tool", http://www.objectsbydesign.com/tools/modeling_tools.html, [accessed 24/10/2006].

Objects by Design (OD) (2005b), "UML modeling tools", http://www.objectsbydesign.com/tools/umltools_byCompany.html, [accessed 24/02/2006].

Smith H. H. (2004), "On tool selection for illustrating the use of UML in system development", *JSCS*, vol. 19, no. 5, pp. 53-63.

Turner S. A., Perez-Quinones M. A., Edwards S. H. (2005), "minimUML: A minimalist approach to UML diagramming for early Computer Science education", *Computing Research Repository*, http://arxiv.org/abs/cs.HC/0603121.