Teaching Modeling using Umple: Principles for the Development of an Effective Tool

Timothy C. Lethbridge
Electrical Engineering and Computer Science, University of Ottawa
800 King Edward Ave, Ottawa, ON Canada
tcl@eecs.uottawa.ca

Abstract

Umple is a model-oriented programming technology designed to teach modeling while at the same time being practical for industrial application. In this paper we discuss six principles we followed in order to ensure Umple can be effective as a learning resource: being highly usable, facilitating incremental learning, providing an experience of positive reinforcement in learners, convincing learners about the value of the material, broadening learning opportunities, and being inexpensive.

1. Introduction

Teaching modeling can be a challenge. If taught abstractly, or with tools that merely allow drawing of diagrams, students often miss the point. They learn syntax, but leave without understanding the value of modeling and without being able to model with reasonable competence [1]. Until recently, the only way to ensure students can create real systems from models was to purchase expensive or complex tools. But both expense and complexity run counter to the needs of educators and learners.

In this paper we describe the principles we use to ensure that the Umple [2] model-oriented programming technology is effective for teaching and learning modeling.

1.1 Quick overview of Umple

Umple is a modeling technology designed to help people learn to model well. The lessons embodied in Umple were derived from years of teaching first OMT [3], and then UML [4], and also observing modellers working in industry [5].

Umple has several aspects that work together:

- It adds textual renderings of modeling concepts to programming languages, allowing programmers to increase their level of abstraction, and write less code [6]. It supports UML class and state diagrams, various patterns, aspects, constraints and concurrency, generating Java, PHP and C++. An Umple program can be seen as a base language program with Umple syntax added, or an Umple program with base language syntax embedded for such things as algorithmic methods. Figure 1 gives a sample of Umple. Many more examples are available in the Umple user manual [7].
- It allows models to be either written textually, or drawn as diagrams, and keeps the two in synchronization. Umple can be run on the web using a website called UmpleOnline [8] (Figure 1), in Eclipse or on the command line.
- It performs extensive analysis of models. Figure 1 shows an example of the display of warnings (each of which has a link to an explanation in the user manual).
- It generates high quality code in Java, PHP and C++ that fully supports the semantics of the modeling constructs. For example, it allows nested state machines at arbitrary levels of depth, with concurrent activities run in threads. It generates code that respects multiplicity constraints and the need for referential integrity.

Umple is not only a tool for educational use; it is designed for use in industry too, on projects of all scales. In fact, the developers hope that by making a learnable tool, they are also likely to foster more and better modeling in industry.

1.2 Use of Umple in Teaching

Umple has been used in teaching since 2009 by several professors. It has been used in courses ranging from the high-school level to graduate studies.

We analysed the grades of students in the years before and after Umple was introduced and discovered that students performed much better at creating basic models after Umple was introduced [1]. Students also showed significant satisfaction with the use of Umple.

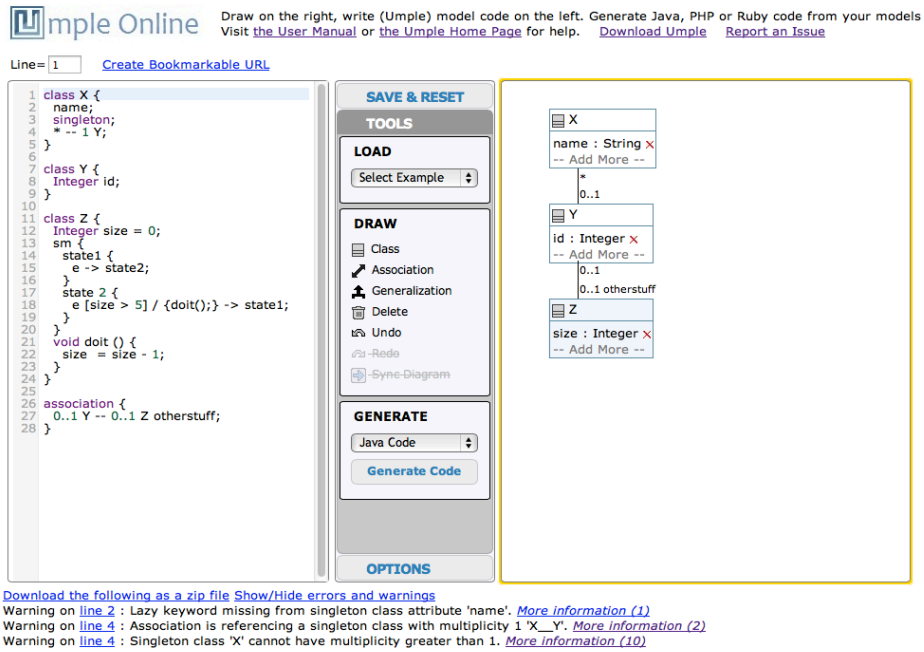

Fig. 1. UmpleOnline, showing Umple text on the left. Lines 4 + 26-28 represent UML associations. Lines 13-23 is a state machine. Lines 21-23 show a method from a base language embedded in Java code. This is called in the state machine on line 18. The UML diagram on the right is always in synch with the text.

2. Educational principles deployed in Umple

The following sections discuss the principles deployed in the design of Umple to make it more suitable for education.

2.1 Principle 1: Focus on high usability

A well-established concern about modeling tools is that they are not as usable as they could be [9]. Choosing a modeling tool poses a challenge for the educator because she or he must either choose a tool so simple that it doesn't do enough, or else too complex.

Usability can be decomposed into related properties [10]. We chose the following three subcategories on which to focus, based on our experiences with other tools.

2.1.1 The tool is simple (easy to install, learn and remember): A key objective was to keep Umple simple, at least for students and most general usage. As features are added, it is essential that they be only exposed as needed and as expertise builds.

The first aspect of simplicity is easy of installation. UmpleOnline exposes most of the power of Umple directly in the browser, hence requiring zero installation. No login is required to start using it for modeling, allowing the professor to instantly illustrate points, and for students to try them out with no delay. Umple also provides a command-line compiler as a Java jar – also easy to install and run for those who prefer, as well as a standard Eclipse plugin in a single file. Only the Eclipse plugin is non-trivial to install, although it is no more complex than other Eclipse plugins.

Ease of learning implies that the functionality must be 'discoverable', and once discovered can be remembered. This is exemplified in UmpleOnline in several ways: When first launched, there is a central menu that makes clear the three most important actions are 'Load', 'Draw' and 'Generate' (Figure 1). The 'gestalt principle' is invoked by placing each of these groups of functionality within boxes. There are two other simple groups of items ('Options' and 'Save and Reset') that can be selected too. We fine tuned this interface while working with students, and observe that students find this minimalist interface self-explanatory and also quick to use.

Ease of learning implies 'obviousness' as above, but for a modeling language there also needs to be readily accessible and well-organized documentation with lots of relevant examples. Both the Umple user manual [7] and UmpleOnline [8] have many live examples. User manual examples can be loaded with one click into UmpleOnline.

Several other notable tools that also focus on simplicity in learning how to develop software are GreenFoot [11] and the related BlueJ [12] platform. Greenfoot is discussed in some detail by DeReamer [13].

2.1.2 The tool guides users away from errors and gently helps the learner to correct them: As Figure 1 illustrates, Umple gives a large number of warnings and other messages that help to guide the user to create models that will result in valid programs. It is possible to continue to informally model in the presence of warnings, however. The Umple user manual has a page for each error or warning, with examples that cause each message, and examples of how each problem can be solved.

2.1.3 Response time is fast: Students are busy, and professors in lectures are even more concerned about lack of delay, so response time in Umple was a key concern. Umple's web-based implementation is zero-footprint; it loads much faster than Eclipse for example. It also quickly updates the diagram or text when the other is edited, and code is generated quickly too. The command-line compiler works as fast as typical compilers.

2.2 Principle 2: Build knowledge and skills incrementally

We anticipate most students will experience their first modeling lessons after learning the basics of programming. For these students, a key Umple objective was to initially add 'a touch of modeling' to their repertoire of skills, building on their programming background, but not replacing it. We hence chose to provide modeling concepts in a textual form, that parsimoniously and synergistically work with other C-family languages (Java, C++, PHP), yet dynamically generate UML diagrams. Once the learner understands the diagrams, they can edit them instead, and gradually increase the modelling content embedded in their code.

On the other hand, some learners might have their first programming experience using Umple as a modeling tool. It is therefore possible to use Umple purely graphically, sketching

the structure or behaviour of a system. The learner can then be taught to fill in details (e.g. a main method) to create a program that executes.

2.3 Principle 3: Give positive reinforcement to learners

In psychology [14], one of the key modes of 'operant conditioning' for learners is positive reinforcement, in which learners are given a reward for achieving something. In computer programming, the reward is a working program: A usable language will facilitate learning because students repetitively achieve success at getting increasingly sophisticated programs to work. Programming hence becomes exciting and enjoyable. The shorter the compile-run-debug cycle, the faster and more intense the learning process.

Learning to model should be the same, but this is only possible if models can actually be created simply, and executed without delay. As a result, Umple learners get the same exciting experience of rapidly seeing their models turned into functioning systems. The user manual [7] has executable examples, which can be used as given or adapted.

Vihavainen et al. [15] describes other approaches to giving positive reinforcement in learning how to develop software, in their Extreme Apprenticeship approach.

2.4 Principle 4: The tool convinces learners of the value of what they learn

Prior to the introduction of Umple, many students openly wondered why they shouldn't just write code. What was the point of drawing the diagrams?

With Umple, students can rapidly see that they can generate large amounts of code in their target language from very small amounts of Umple text, or UML diagrams. Furthermore, the generated code is written in a manner so as to be as readable as possible, and documentation (in Javadoc format) is also generated at the same time, so users see yet another benefit of modeling and the Umple approach.

It is necessary, however, to go further: To convince students that this approach scales up. To do this, we wrote Umple in itself and made it open source. The entire compiler (99 files, 38000 lines of Umple code including embedded Java) is available for examination and modification. The generated class diagram is also available and allows navigating about the code [16]. Figure 2 shows a screen image of part of this large diagram.

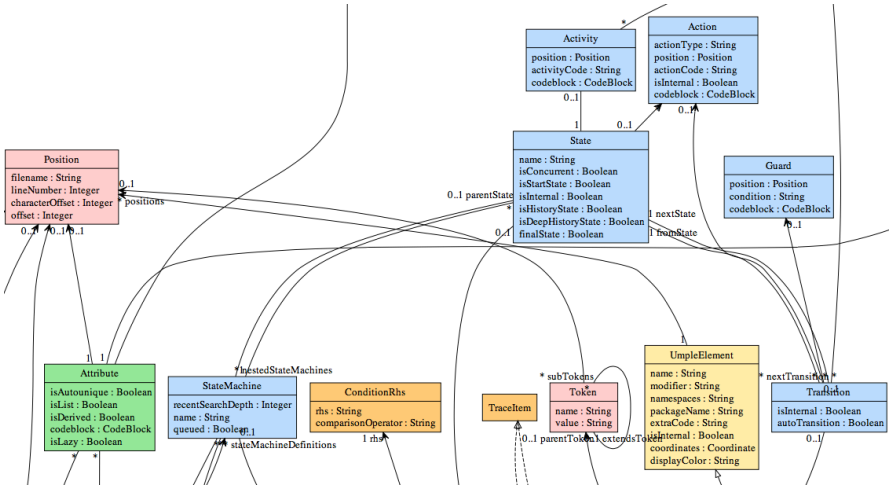

Fig. 2. Part of the dynamically generated class diagram of Umple itself (metamodel.umple.org) [16], used to help students appreciate the practicality of what they are learning (Principle 4), and also to help them actively contribute to Umple development (Principle 5).

The fact that Umple has been, and continues to be sponsored by industry (IBM, Ericsson, GM, the Canadian Defense Department) also helps to convince learners of its usefulness: Students see that they are learning transferrable skills.

2.5 Principle 5: The tool gives maximum opportunities for learning

Many UML tools either do not cover UML comprehensively, or generate weak code [17]. Umple was designed to be strong in both aspects, while remaining simple. Umple was also designed with a variety of additional features to expand what can be taught and learned beyond pure modeling: It has aspect-oriented features, concurrency, constraints (including method preconditions for example), and has several built-in design patterns.

The compiler itself also can be used to demonstrate many aspects of both model-driven software engineering (since it is written in itself) and agile software development. In particular, it uses test-driven development and continuous integration.

The Umple compiler has been used to give students active development experience in not just modeling, but also all other aspects of software engineering. Over 30 students from 12 Canadian universities [18] and also from Brazil have contributed to Umple as part of their fourth-year undergraduate 'capstone' courses. Two PhD theses have been completed on Umple [17] [19], as have several Masters theses. Five PhD students and are in progress. Through these experiences, students come to see modeling as a 'second-nature' part of their software engineering activity. Modeling is no longer 'just for documentation;' students come to see it as just as essential as programming.

2.6 Principle 6: The tool is inexpensive

Last but not least, in the educational community neither students nor institutions can afford very many expensive tools. Umple is free.

3. Conclusions

We have discussed six principles that guide the development of the Umple modeling technology, such that it can be effectively used to educate about modeling. These principles were derived from our experiences with other tools and our desire to do better.

The principles are: 1) Focus on usability; 2) Build on students' knowledge incrementally; 3) Provide positive reinforcement; 4) convince students of the value of modeling; 5) Maximize opportunities for learning, and 6) Make it inexpensive.

Umple has been demonstrated to help students improve their modeling skills [1]. We recommend others creating modeling tools consider a similar approach.

References

[1] Lethbridge, T. C., Mussbacher, G., Forward, A., Badreddin, O., May 2011. Teaching UML using Umple: Applying model-oriented programming in the classroom. In: 2011 24th (CSEE&T). IEEE, pp. 421-428. http://dx.doi.org/10.1109/cseet.2011.5876118

[2] Umple home page, University of Ottawa, http://www.umple.org

[3] Rumbaugh, J., Blaha, R., Lorensen, W., Eddy, F., Object-Oriented Modeling and Design, Prentice Hall, 1990

[4] Object Management Group, UML, http://www.omg.org/spec/UML/

[5] Farah, H. and Lethbridge, T.C., 2007, Temporal Exploration of Software Models: A Tool Feature to Enhance Software Understanding, WCRE 2007, Vancouver, IEEE Computer Society.

[6] Forward, A., Lethbridge, T. C., Brestovansky, D., May 2009. Improving program comprehension by enhancing program constructs: An analysis of the Umple language. In: 2009 WCRE. IEEE, pp. 311-312. http://dx.doi.org/10.1109/icpc.2009.5090073

[7] Umple User Manual, http://manual.umple.ort

[8] Umple Online, University of Ottawa, http://try.umple.org

[9] Forward, A., and Lethbridge, T.C. (2008), “Problems and Opportunities for Model-Centric Versus Code-Centric Software Development: A Survey of Software Professionals”, MSE Workshop, ICSE 2008, pp. 27-32

[10] ISO, ISO 9241 Ergonomics of Human Computer Interaction

[11] Greenfoot, http://www.greenfoot.org

[12] BlueJ – Teaching Java – Learning Java, http://www.bluej.org

[13] DeReamer, S., Teaching Computer Science: A Neumont Philosophy, 2010, ACM Sigsoft Software Engineering Notes, 35, 1 pp. 31-34

[14] Walker, H., and Buckley, N., 1968, The Use Of Positive Reinforcement In Conditioning Attending Behavior, Journal of Applied Behavior Analysis, 1, 3, pp. 245-250, http://dx.doi.org/10.1901/jaba.1968.1-245

[15] Vihavainen, A., Paksula, M. and Luukkainen, M. 2011. Extreme apprenticeship method in teaching programming for beginners. SIGCSE 2011 ACM, pp. 93-98. http://doi.acm.org/10.1145/1953163.1953196

[16] Umple Metamodel, http://metamodel.umple.org

[17] Forward, A., 2010. The convergence of modeling and programming: Facilitating the representation of attributes and associations in the Umple Model-Oriented programming language. Ph.D. thesis, University of Ottawa. http://www.site.uottawa.ca/~tcl/gradtheses/aforwardphd/.

[18] Undergraduate Capstone Open Source Projects, http://www.ucosp.ca

[19] Badreddin, O., “A Manifestation of Model-Code Duality: Facilitating the Representation of State Machines in the Umple Model-Oriented Programming Language”, PhD Thesis, University of Ottawa, http://www.site.uottawa.ca/~tcl/gradtheses/obadreldin/