Umple: a Model-Oriented Programming Language

Omar Badreddin
SITE, University of Ottawa
800 King Edward Ave, Ottawa
obadr024@uOttawa.ca

ABSTRACT
Our research tool, Umple, has the objective of raising the abstraction level of programming languages by including modeling abstractions such as UML associations, attributes and state machines. My research focuses on the syntax and semantics of state machines in Umple and the empirical validation of Umple as a whole.

Categories and Subject Descriptors
D.3.3 **[Programming Languages]**: Integrated environments – *UML, Modeling, coding.*

1. The Problem
It is widely accepted that modeling is considered a good software development practice [8]. Forward et al, however, show that modeling practices are not as widely adopted as might be desirable [4]. They attribute the low adoption to the heavyweight nature of tools, poor generated code, and difficulty synchronizing code and models. The key problem we are solving then is how to create lightweight capabilities that allow modelers and programmers to seamlessly build applications.

2. Prior Work
There is considerable prior work related to textual modeling and code generation. Textual UML modeling research tends to focus on the usability of working with visual models textually. Examples include MetaUML [10], yUML [7], and TextUML [3]. These approaches, however, do not function as a complete development tool, nor do they integrate with programming languages seamlessly.

Executable textual modeling tools support automated code generation. For example, Ragel [12] supports a number of high-level languages but is targeted towards text parsing and input validation. Ruby on Rails recently added built-in support for state machines. The state Machine Compiler (SMC) [2] is targeted towards the specification of event driven systems. Microsoft is also developing a textual specification language, AsmL [6], based on state machine concepts. Those approaches do not incorporate class diagram abstractions and do not support development of complete applications. Executable UML [9] supports a subset of UML textually but misses key features of UML and does not integrate with programming languages.

ICSE '10, May 2-8, 2010, Cape Town, South Africa

3. Research Hypothesis
To address the problem, our research group has developed a model-oriented programming language called Umple. Umple integrates modeling concepts and code in the same artifact, allowing developers to model and write code at the same time.

Our hypothesis is that enabling the unification of coding and modeling will both enhance modeling and enable those who don't model to adopt it painlessly. Software engineers, especially those that are code-centric, will be able to model in a way similar to their programming habits. Unifying models and code should help alleviate several challenges:

- *Models become quickly out of date and therefore obsolete.* Because our approach unites models and code, models never become outdated since they are always maintained as long as the code is maintained.
- *Round-tripping between code and models is challenging* [11]. The need for roundtrip engineering is eliminated because code and model become integrated.
- *Most modeling tools prevent developers from using purely textual approaches even if they prefer them over editing diagrams. Creating nice looking models becomes time consuming*. With our approach models can be quickly created textually making use of text editing features like context assist and syntax-support editing. Diagrams can then be unambiguously generated.

4. The Umple Modeling/Coding Platform
Umple enables the programming of associations, attributes, and state machines into a base programming language. It supports Java and Php, but can easily be extended to support any object-oriented programming language. My doctoral work involves the development of the state machine syntax and semantics. It builds on the work of several other students.

We provide a simple example composed of three classes with associations and a two-state state machine. In this example, *Registration* is an association class; a *Student* has many *Registrations*, and the *CourseSection* also has many *Registrations*. The *CourseSection* class has a state machine attribute with two states, *Closed* and *Open*. While in state *Closed*, event *initiateRegistration* triggers a transition to state *Open*. The event *reqToRegister* triggers self-transition with action *notifyInvalid()*. Transitions may have guards, like the transitions triggered by the *drop* event. Note that the drop event also illustrates nested state machines; it applies to both *Closed* and *Open*. On entry to state *Open*, an entry action occurs; the code for actions can be arbitrary code. Umple also supports exit actions, and 'do' activities running in a separate thread that can be interrupted. Reusing state machine definitions in different

classes is also supported as is inheritance and refinement in subclasses.

```
class Student{
 1 -- * Registration;
 immutable studentID; }

class Registration{
 * -- 1 CourseSection;
 Integer grade; }

class CourseSection{
 Boolean deadlinePassed;
 Integer maxSize;
 sectionID;
 openClosedStatus {

  Closed{  // Initially course closed
   initiateRegistration -> Open;
   reqToRegister /{notifyInvalid();}->Closed;}

  Open {
   Entry / {notifyOpen();};
   reqToRegister / {registerStudent()};
   (Registration.size > maxSize) -> Closed;
  }
  drop [!deadlinePassed]-> Open;
}}
```

5. Research Accomplishments

I have designed and implemented the first version of state machine constructs in Umple; this has involved extensive analysis of alternatives for syntax and semantics, as well as experimentation by developing test examples. I intend to refine this work as the research progresses. In parallel, I have been conducting a grounded theory study of Umple to learn in an iterative manner what works for programmers and what does not. I have also been administering a survey [1] on the types of modeling notations being used by professionals and academics. The objective of this is to guide Umple enhancement efforts towards the most useful modeling notations.

6. Upcoming Research Activities

I aim at improving Umple and making it available for students and professionals. Specifically, we intend to implement and support arbitrarily deep nested and concurrent states and timed transitions. We also need to work carefully on the semantics of reusable state machines, and refinement in inheritance.

We will validate our work using two approaches:

6.1 Grounded theory study of Umple users

By continuing to making Umple available, the number of Umple users is continuously increasing. We are conducting an empirical study of Umple using a grounded theory approach to evaluate Umple technology. We have completed a first iteration of this study. We expect it to help us understand how a software engineer perceives and utilizes Umple features and also to guide us in improving Umple.

6.2 Analysis of open source applications

We have built all recent Umple versions using Umple itself. In a similar manner, we intend to refactor some existing systems so they use Umple. This refactoring will enable us to assess the characteristics of the refactored Umple code in terms of its complexity, quality, and cohesion. We also intend to build a significant library of examples to demonstrate how incorporating state machines in the manner proposed can result in both better models and better programming. This empirical assessment also enables us to understand the level of coverage Umple provides for typical industrial software applications.

7. Expected Contribution

At the end of my research activities I expect to have provided a modeling tool that enables software developers to write code at a higher abstraction level by embedding state machine abstractions into their code. I also expect to have provided an extensive empirical evaluation of the work.

8. How Our Research is Distinct

Umple is distinct from prior work because it integrates class diagrams and state machines into a full fledged model-oriented programming language that simultaneously acts as a modeling tool. Class diagram elements can be controlled by state machines, and state machines can be instantiated within class diagrams. As mentioned, we modeled and implemented Umple using Umple itself. Executable code generation is seamless and, like any high level language compiler, eliminates round tripping from code to model and back. Umple integrates with a variety of tools such as Eclipse, Rational Software Architect, UMLet, and also can be used stand-alone on the command line. A simplified version is also available online [5].

9. References

[1] Badreddin, O.. "Online modeling survey". August, 2009. http://spreadsheets.google.com/viewform?formkey=cFZqeVg0cGZzZkZpSFl1SVNjaFI2UEE6MA..

[2] Charles, R. 2009. "The state machine compiler". http://smc.sourceforge.net/

[3] Chaves, R. 2009. "TextUML". http://abstratt.com/

[4] Forward, A. 2007. "A Survey of Software Practitioners". http://www.site.uottawa.ca/~tcl/gradtheses/aforwardphd/.

[5] Forward, A. "UmpleOnline". 2009. http://cruise.site.uottawa.ca/umpleonline/.

[6] Gurevich, Y., Rossman, B. and Schulte, W. 2005. "Semantic essence of AsmL". *Th Computer Science*, 343, 370-412.

[7] Haris, T. "yUML". 2009. http://www.yuml.me/

[8] Kleppe, A. 2003. " MDA Explained: The Model Driven Architecture: Practice and Promise". ISBN:032119442X.

[9] Miliciv, M. "Model-Driven Development with Executable UML". Worx, ISBN: 978-0-470-48163-9.

[10] Gheorghies, O. 2009. "MetaUML". http://metauml.sourceforge.net/

[11] Selic, B. 2003. "The pragmatics of model-driven development". *IEEE Software*, 20, 19-25.

[12] Thurston, A. 2009. "Ragel state machine compiler". http://www.complang.org/ragel/