A Study of Applying a Research Prototype Tool in Industrial Practice

Omar Badreddin
University of Ottawa
800 King Edward
Ottawa, Ontario, Canada
001 757-644-4494
obadr024@uottawa.ca

Timothy C. Lethbridge
University of Ottawa
800 King Edward
Ottawa, Ontario, Canada
001 613-562-5800x6685
tcl@site.uottawa.ca

ABSTRACT
Our research tool, Umple, has the objective of raising the abstraction level of programming languages by adding modeling abstractions such as UML associations, attributes and state machines. My research focuses on the syntax and semantics of state machines in Umple plus the empirical validation of Umple. The latter consists of a grounded theory study of Umple users, and action research involving the use of Umple in an industrial sitting. The grounded theory study is guiding our research and development of Umple. The action research is enlarging the pool of Umple users and contributing to the validation of Umple's approach.

Categories & Subject Descriptors: D.2.6
[Programming Environments]: Integrated environments – *UML, Modeling, coding*.

General Terms: Experimentation, Verification

1. INTRODUCTION
Umple is a model-oriented programming language that incorporates modeling abstractions into common object-oriented languages. Umple enables developers to take advantage of the power of modeling, while maintaining the textual development environment, an environment familiar to most developers. This is achieved without compromising the communication advantage of visual models.

Umple has the following high level goals:

- Enhance the adoption of modeling practices among software development teams.
- Reduce, or eliminate, the need for model/code synchronization.
- Provide an environment most appealing to developers that enables them to take advantage of modeling abstractions.

In the course of developing Umple, we have observed the following benefits of using textual modeling in software development activities:

FSE-18, November 7–11, 2010, Santa Fe, New Mexico, USA.

- In executable modeling environments, textual models blend better with UML Action Languages.
- Versioning and merging of textual models are significantly more efficient, results in fewer conflicts, and leverages the infrastructure already in place for code version control.
- Textual models enable developers to rapidly create prototypes to better understand their modeling implications on the developed system.

The reader is referred to [1] for an over view of Umple approach, [2] for Umple rapid prototyping benefits. We are maintaining an online version of Umple [3] that provides an easy forum with examples to demonstrate many of the concepts available in Umple, with zero install footprint.

2. THE PROBLEM
Research prototypes, particularly those in the software engineering discipline like Umple, face challenges in deployment and validation in industrial practices. Reasons for such challenges may include:

1. Research prototypes are frequently not ready for industrial deployment;
2. Research tools do not integrate well with other tools, and lack industrial level of reliability and support;
3. Risk aversion on the part of industrial users;
4. Time constraints and other business commitments;
5. Legal regulations or conflict with business objectives;

We have initially opted for a grounded theory study of Umple users as an empirical approach to evaluating Umple. To date, Umple is mainly used by a small research team, and by a small number of undergraduate and graduate students at the University of Ottawa.

Anticipating the potential value of Umple in industrial projects, we have therefore complemented our Umple evaluation effort with an *action research* methodology. Such a methodology will help deal with many of the challenges of adoption of Umple into real industrial practice, and will provide potential professional participants for the grounded theory study.

3. PRIOR WORK
Umple textual modeling approach is not new. There is a growing amount of work that adopts textual modeling, either to complement or to replace mouse-centric visual modeling approaches. Examples of such textual modeling work include

MetaUML [4], yUML [5], and TextUML [6], State Machine Compiler (SMC) [7] , AsmL [8], and Executable UML.

In addition to the highly abstract human readable textual notation, Umple is unique in its approach to integrating such textual notations in object-oriented programming languages like Java and PHP. This approach effectively raises the abstraction level of object-oriented programming languages. Ruby is adopting a similar approach, where UML abstractions are incorporated within Ruby's textual notation [9]. Such approaches are promising; Umple however implements abstractions that are language independent. Umple currently supports Java, PhP, Ruby, and work is on going to support other languages, such as C++.

There has been little work on empirical evaluation of textual modeling and even less in industrial sittings. The work of Aschauer and others [10] is an example of a Domain Specific Language (DSL) that is developed within tight integration with industry. In this work, the DSL requirements and directions are driven by immediate industrial needs, and was therefore, easily put into industrial production. Umple, on the other hand, has been independently developed within a research institute.

4. MOTIVATION

Empirical methodologies have emerged in the social sciences, and are growingly finding adoption in other sciences and engineering disciplines. Grounded Theory methodology, for example, was first conceived by Glasser and Strauss in 1967 [11]. However, it was not until 1993 that we could find the first significant grounded theory work applied in software engineering [12].

Software Engineering is increasingly adopting empirical methodologies. This is reflected in the number of publications, conferences, and journals that focus on empirical work. Nevertheless, such methodologies, having emerged from the social sciences, are not readily implementable without modifications in software engineering discipline.

For example, social sciences do not face the same types of challenges in identifying subjects for their studies. Other than following approved ethical standards, and subjects consenting to participate, there are little to no additional challenges. In software engineering, recruiting appropriate participants is more challenging.

It is widely accepted that professional subjects are preferred to student subjects [13] ; however, it is significantly harder to find professional subjects for the following reasons:

1. Professional subjects have competing business obligations and are less likely to be willing to participate than a typical subject from the general public.
2. Management is added layer of complexity for selecting professionals as subjects; management may not approve of the time required from subjects or the study as a whole. In addition, professional employees are considered a volatile group, addressing additional ethical issues becomes a must.
3. For a professional subject to qualify for a study, like Umple's, the professional may need to have spent significant amount of time using Umple to build sufficiently complex or real life solutions using Umple technology.

5. AN OVERVIEW OF UMPLE

Umple enables the programming of associations, attributes, and state machines into a base programming language. My doctoral work focuses on the development of the state machine syntax and semantics as well as the empirical evaluation of Umple as a whole. It builds on the work of several other students.

Figure 1 shows a simple UML state machine, where 'E' is the event to which S1 responds to; 'G' is any Boolean expression, or a Boolean function, or a code segment; and, 'A' is a function call, or any arbitrary implementation code.

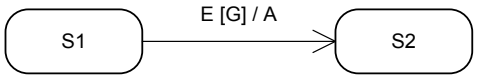

Figure 1: state machine transition

Figure 2 shows Umple code in which two attributes, one association and one state machine are declared. The state machine is the same one that appears in Figure 1.

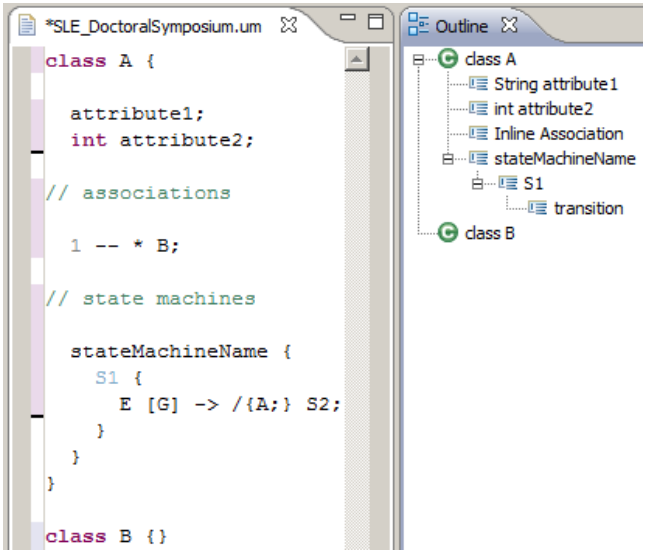

Figure 2: Textual editor and Outline view

Implementation code of any of the supported languages can be embedded within Umple modeling abstractions shown in Figure 2. Umple's outline view treats both implementation code and modeling abstractions uniformly. The text editor provides context assist and syntax highlighting. Not shown here are the Umple problems view, where errors and problems in Umple are reported to the user with recommendation for resolution, and the Umple visual view, where the modeling elements can be manipulated.

6. RESEARCH ACCOMPLISHMENTS

I have completed the implementation of the state machine related Umple features, and conducted a comprehensive quality assurance to make sure that Umple core features function at a quality level matching levels expected in an industrial project.

The quality assurance was implemented using a test driven approach [14]. In summary, Umple development infrastructure implements an integrated test suite of about 1400 test cases that automatically runs whenever a new feature or enhancement is developed. The test cases cover all phases of compilation and

code generation, including parsing, tokenization, meta-modeling population, and code generation.

We are evaluating Umple using two approaches; a grounded theory study of Umple users, and an action research that involves using Umple in an industrial project. The following two sections present each study, and a summary of the initial findings.

7. GROUNDED THEORY STUDY OF UMPLE USERS

Grounded theory (GT) is a systematic qualitative research methodology, originating in the social sciences, and emphasizes the generation of theory from qualitative data in the process of conducting research. The main objective of this Umple study is to understand the perception of Umple's users, to provide input for Umple development, and to generate theories about the patterns and characteristics of Umple use in developing complete systems.

In addition to performing classic grounded theory research, the study also involves a questionnaire of users. The questionnaire includes 21 questions that focus on the participants' user experience, their perception of the syntax, the generated code, and Umple's role in past, present, or future software development activities. The responses are rated on a Likert scale from strongly agree, to strongly disagree.

The classic GT aspect of the study involves interviews with open-ended questions; it allows the interviewee to guide the flow of the interview. The interviews are analyzed qualitatively to 'code' the information gathered; i.e. to arrange it into categories

We have so far recruited 8 participants in this study. All participants have been undergraduate and graduate students who have used Umple in their course work. We anticipate being able to have professional participation once the action research work is near completion.

7.1 Findings

The questionnaire is targeted towards specific aspects of Umple. Table 1 presents the two highest and lowest weighted average scoring questions.

Table 1: Sample questionnaire questions

#	Question	Weighted Average
1	Umple code is easier to understand than the equivalent Java code?	3.5
2	I am satisfied with how Umple implements associations in the generated Java code?	3.4
3	Error messages that appear when I compile Umple are very useful?	1.3
4	Syntax errors in Umple are easy to identify?	1.2

The first two questions are in line with our hypothesis; adding modeling abstractions to common object oriented programming language enhances program comprehension, reduces complexity, and minimizes boilerplate code. Low averages for Q3 and Q4 are expected; early releases of Umple did not provide textual editing features, such as context assist. We have addressed such low ratings by building a syntax-driven editor that enhances Umple's error messages and code-assist aspects (Figure 2).

The coding process for the trascripted interviews was performed using a word processor. Shading and colors were used to apply codes to portions of the transcripts. The process included two phases, initial coding, and coding phases. Figure 3 illustrates the top two level codes.

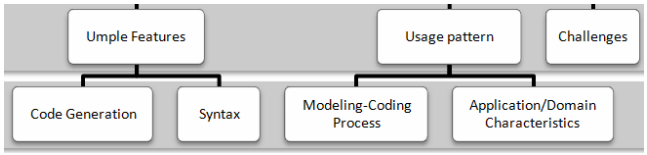

Figure 3: Two-level codes

The "code generation" code included sub-codes that address user's perception of the Umple generated code. For example, few users expressed negative sentiments with what they perceived to be unnecessary repeated code. The "Syntax" code included perception on the usability of the syntax and recommendations for improvements. For example, we observed that some of the users prefer to have the guard before the event, and others preferred the guard to come after the event (

Table 2).

Table 2: Guard and event syntax

```
[G] E -> S1;
E [G] -> S1;
```

Umple was updated to support both preferences.

The modeling and coding process provided insights on how Umple users go about modeling and developing systems. We have observed different patterns where some users prefer to write pure Umple, others preferred to start modeling visually, and later add implementation code in Umple textual editor.

Because all participants in this study were students, the levels of complexity of the development tasks were relatively simple, resulting in rather simple modeling/development patterns. This aspect of the study provided further motivation into conducting the action research work.

8. APPLYING UMPLE IN AN INDUSTRIAL PROJECT USING AN ACTION RESEARCH APPROACH

Action research involves the researcher actively immersing himself in a task while researching that task.

The target industrial setting is a large Business Process Management (BPM) project, where business operations are modeled using BPMN notation [15]. Incremental subsets of the BPM are iteratively selected for automation. The client is a large fund granting agency, and the timeline to complete the project is estimated to be 5 years.

Project stakeholders identified the need to model system entities using state machines. The project involves a number of stakeholders from different organizations, and therefore, a strict process of reviews and approvals are in place. Figure 4 illustrates the review and approval process for approving the state machine models.

Having a strict review process in this project is particularly attractive for Umple because:

1. We want to evaluate Umple effectiveness as a way to version and merge models. The review and approval process in place

creates several model versioning and merging scenarios ideal for evaluating versioning and merging of models using Umple;

2. The review process ensures that the models created go through typical development scenarios and reflect real life modeling practices;
3. Many of the project stakeholders are already using and are familiar with modeling notations, making the adoption of Umple easier;

Fifty data entities have been identified as candidates for state machine modeling, of which 30 are targeted for modeling using Umple. The remaining entities will be modeled visually using other modeling tools. This mix of tooling is interesting from a research perspective, since it provides more data on the characteristics of textual and visual modeling.

At the time of writing, 15 state machine models have gone through the first four steps of the review cycle (Figure 4).

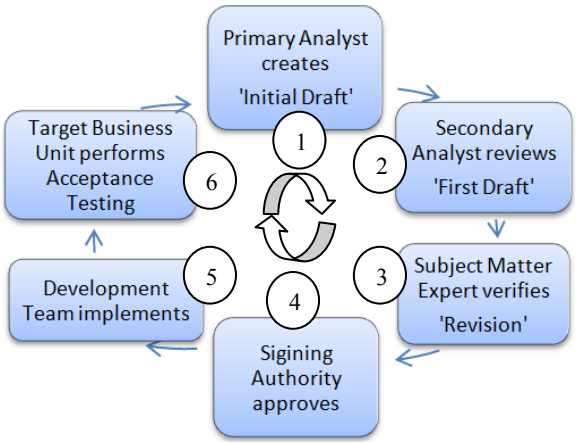

Figure 4: Artifact review process

8.1 Findings

We present here a summary of the findings related to the characteristics of the state machine models and the versioning of Models using Umple.

We used a central repository to manage model versions. On average, each state machine model went through 14 versions before being approved by the signing authority. We are anticipating more versions when the state machines models are implemented by the development team.

On average, the state machines consist of 15 states, 21 transitions, 18 transition actions, 10 guards. Entry and exit actions, nested and concurrent states were not used. We attribute this to the project manager's objective of keeping models simple, particularly since the state machine models were reviewed and approved by stakeholders not always having strong familiarity with UML state machines notation.

Because Umple models were represented textually, we were able to version and merge all model revisions in SVN, a lexical repository already in place for managing code versions. We were also able to track model changes and automatically reproduce all model history. Team members relied on automatic conflict resolution when merging versions.

9. FUTURE WORK

More extensive analysis of the results of the action research is pending. Particularly, we need to empirically investigate evidence of model versioning and merging of models that are internally stored in a textual format. In addition, since Umple is put in use in an industrial project, we expect to be able to recruit professional participants to our grounded theory study. This will further enhance the depth of analysis we can perform in the grounded theory study.

We are conducting a workshop on the challenges and opportunities associated with research prototype adoption in industry, and we intend to use Umple as a case study.

10. CONCLUSION

This research presents Umple, a technology for adding language-independent textual modeling abstractions to high level object oriented programming languages.

This doctoral research involves developing syntax and semantics for state machines in Umple, and also its empirical evaluation in industrial sittings using two empirical research methodologies, a grounded theory study, and an action research study. The grounded theory is guiding the research and development effort of Umple; the action research is widening the pool of Umple users to include professionals. During the course of this research we hope to uncover general insights into patterns of adoption of research prototypes into industrial practices.

11. REFERENCES

[1] Badreddin, O. "Umple: A model-oriented programming language," in Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2, 2010, pp. 337-338.

[2] A. Forward, O. Badreddin and T. C. Lethbridge. "Umple: Towards Combining Model Driven with Prototype Driven System Development". 2010. RSP

[3] "UmpleOnline," accessed 2010, http://cruise.site.uottawa.ca/umpleonline/

[4] UML MetaUML. Diagrams for La T E X, Internet

[5] "yUML," accessed 2009, http://yuml.me/

[6] "TextUML," accessed 2009, http://abstratt.com/

[7] Charles W. Rapp. (December, 2009., "The state machine compiler". vol. 6.0.1, December, 2009.

[8] Y. Gurevich, B. Rossman and W. Schulte. "Semantic essence of AsmL". 2005. Theor.Comput.Sci. vol 343, pp. 370-412.

[9] Ruby. (2009, "Rails state machine". vol. V5.0, 2009.

[10] Thomas Aschauer, Gerd Dauenhauer, Wolfgang Pree. "A Modeling Language Evolution Driven by Tight Interaction between Academia and Industry". May, 2010. ICSE, 2010

[11] "The Discovery of Grounded Theory: Strategies for Qualitative Research". New York: Aldine de Gruyter, 1977, pp. 257.

[12] W. J. Orlikowski. "CASE tools as organizational change: investigating incremental and radical changes in systems development". 1993. MIS quarterlypp. 309-340.

[13] M. Höst, B. Regnell and C. Wohlin. "Using students as subjects—a comparative study of students and professionals in lead-time impact assessment". 2000. Empirical SE vol 5, pp. 201-214.

[14] "Test Driven Development: By Example". Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc, 2002,

[15] "Business Process Modeling Notation, V1.1," accessed 2008, http://www.bpmn.org/