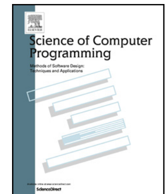
Science of Computer Programming 208 (2021) 102665

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico



Original software publication

Umple: Model-driven development for open source and education

Timothy C. Lethbridge *, Andrew Forward, Omar Badreddin, Dusan Brestovansky, Miguel Garzon, Hamoud Aljamaan, Sultan Eid, Ahmed Husseini Orabi, Mahmoud Husseini Orabi, Vahdat Abdelzad, Opeyemi Adesina, Aliaa Alghamdi, Abdulaziz Algablan, Amid Zakariapour

University of Ottawa, Canada

A R T I C L E I N F O

Article history:
Received 9 August 2020
Received in revised form 14 April 2021
Accepted 15 April 2021
Available online 21 April 2021

Keywords:
Model-driven development
Code generation
Compiler

A B S T R A C T

Umple is an open-source software modeling tool and compiler. It incorporates textual language constructs for UML modeling, including associations and state machines. It includes traits, aspects, and mixins for separation of concerns. It supports embedding methods written in many object-oriented languages, enabling it to generate complete multilingual systems. It provides comprehensive analysis of models and generates many kinds of diagrams, some of which can be edited to update the Umple code. Umple runs on the command line, in a web browser or in integrated development environments. It is designed to help developers reduce code volume, while they develop in an agile, model-driven manner. Umple is also targeted at educational users where students are motivated by its ability to generate real systems from their software models.

© 2021 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

* Corresponding author.
E-mail address: timothy.lethbridge@uottawa.ca (T.C. Lethbridge).

https://doi.org/10.1016/j.scico.2021.102665
0167-6423/© 2021 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

Software metadata

(executable) Software metadata description	
Current software version	*1.30.2*
Permanent link to executables of this version	*Github Jar for command line interface (CLI) use:* *https://github.com/umple/umple/releases/latest* *GUI: https://try.umple.org and via Docker at http://docker.umple.org*
Legal Software License	*MIT*
Computing platform / Operating System	*MacOS, Linux, or Windows with Java Virtual Machine (JVM) version 8 and higher). Alternatively runs in a Docker container on any Docker-supported platform.*
Installation requirements & dependencies	*No required dependencies other than either JVM or Docker. Runs as a Command-Line-Interface (CLI) tool with the JVM, or through a web browser with Docker. Other options are available to run in Eclipse or Microsoft Visual Studio Code.*
Link to user manual	*https://manual.umple.org*
Support email for questions	*umple-help@googlegroups.com but also questions may be asked on Stack Overflow, with the Umple tag.*

Code metadata

Code metadata description	
Current Code version	*1.30.2*
Permanent link to code / repository used of this code version	*https://github.com/ScienceofComputerProgramming/SCICO-D-20-00148*
Legal Code License	*MIT*
Code Versioning system used	*Git*
Software Code Language used	*Umple (i.e. self-compiling) with embedded Java for the compiler. PHP and Javascript for the GUI*
Compilation requirements, Operating environments & dependencies	*Compilation operating environment: MacOS, Linux or Windows* *Dependencies for compilation: Ant (or Gradle), Java 8+, and the previous version of Umple (as Umple is self-compiling).* *Additional dependencies for testing generated code as part of the build: Ruby and PHP 7+*
Link to developer documentation / manual	*http://architecture.umple.org*
Support email for questions	*umple-dev@googlegroups.com*

1. Introduction

In this Original Software Publication paper, we present Umple [22,21], an open-source technology that blends high-level abstractions, widely referred to as models, with widely-used programming languages. By generating code for the abstractions, it allows creation of software with many-fewer lines of code than would otherwise be possible.

Umple consists primarily of:

- A compiler that processes Umple code and generates code in one of several programming languages, as well as diagrams and other artifacts. The input Umple code can contain modeling constructs such as UML associations and state machines; code written in other programming languages, and constructs that allow separation of concerns. The compiler provides extensive analysis of the Umple code and generates many warnings and error messages. The compiler can work on systems with thousands of files, as demonstrated by the fact that Umple is compiled in itself.
- A web-based front-end called UmpleOnline [31] that allows editing of Umple code either textually, or rendered as diagrams, and enables interaction with the compiler running as a server.

Umple was developed to target two audiences: The first is open-source developers who overwhelmingly work with textual languages but can benefit from Umple features to improve quality and reduce code volume. The second audience is educators who want to teach modeling in UML, but can improve motivation and engagement if the students can build complete systems with their models.

Umple can be run on the command line, in Eclipse, or in a web environment called UmpleOnline [31].

Umple is an important technology for several reasons: It is the only modeling technology that is written in itself, and that can generate and analyse large-scale systems. It is also one of a very small number of feature-rich modeling and code-generation technologies that is completely open source. Finally it has been designed to be easily integrated into open source toolchains, without dependencies on any technology other than the Java Virtual Machine.

2. Problems and background

The main problems Umple addresses are:

- Making it possible to develop software simultaneously with a textual and visual representation of high-level abstractions, so as to gain the advantages of either representation.
- Reducing the volume of code that needs to be written due to generation of code from high-level abstractions.

For decades, developers of all types have organized their software primarily in terms of sets of textual files containing source code. At the same time, they have often drawn diagrams representing certain views of their software so as to design it at a high level of abstraction, to explain it to others, and to understand its design. For the most part, these two types of representation have remained separate, with the source code being the *gold master*, and the diagrams being primarily found in documentation that is often outdated [26].

There are many reasons for the primacy of textual code: Generations of developers are comfortable with it, there are numerous editors available, it can be commented easily, and version-management tools work well with it. Diagrams can also be a bother to lay out in an informative way, even with automated algorithms, whereas indenting of text is generally easy. Text also melds well with modern approaches to software engineering, particularly agile approaches. At the same time, the two-dimensional nature of diagrams can be very helpful in ensuring quality and communicating. If the diagrams can be edited, with textual code generated, then the diagrams can make redundant a lot of error-prone typing of boiler-plate code.

Some people have promoted *round-trip engineering* [29], wherein code is reverse-engineered to diagrams, which are then edited and converted back into source code. However, this tends not to work well in practice, according to our experience and those of developers with whom we have worked. This alternating between diagrammatic and textual programs is really the opposite of what we have been trying to achieve with Umple. The biggest problems with round-trip engineering are these: Firstly, the model is no longer the single master to be edited, leading developers to have to mentally keep track of both model diagrams and generated code. Secondly the developers tend to modify the generated code in inconsistent ways, resulting in the technology not being able to regenerate the diagrams.

We consider all representations of software to be models, although models that capture the highest levels of abstraction are the more powerful ones. Examples of such abstractions include state machines and UML associations. The notion of *model-driven development* (MDD) envisions generating software from these high-level abstractions, avoiding the need to write repetitive boiler-plate code.

In 2006, we set out to develop a technology that would provide the best of both worlds: We wanted to achieve the editability, documentability, familiarity, version-tracking, and agility of textual code. At the same time, we wanted to incorporate high-level abstractions (modeling) directly into that code, allowing the code to be directly manipulated by a diagrammatic editor simultaneously as it is edited by a text editor. The resulting software is called Umple.

Umple is explicitly *not* a single new programming language; instead, it consists of a set of features that extend *multiple* existing programming languages. The extensions add various kinds of abstractions (state machines, associations, mixins, traits and so on) consistently to those languages, while allowing use of these features both diagrammatically and textually. Umple enables people to develop the abstract structure or behavior of software quickly, and understand the implications of the so-called models, and to generate functioning systems from them.

The modeling language UML, with several types of diagrams, was developed starting in the mid 1990's, and there are many UML tools. The most recent generation of these tools includes Papyrus [18], and Astah [7]. However, research has shown that UML is not widely used in practice, at least for actual generation of systems [27]. We studied over 20 such tools as used by professors and students, [4,5] and found them to have numerous weaknesses, including being overly complex, not generating good quality code, and not giving good feedback about models. The Umple project seeks to overcome these limitations.

3. Software framework

3.1. Software architecture

The core component of Umple is the compiler. An Umple program is processed by the Umple compiler to generate various outputs including diagrams as well as complete systems in C++, Java, PHP, and other target languages. The compiler is a Jar file and runs under the JVM. The compiler itself is generated from Java code, which is in turn generated from code written in Umple. Thus, the Umple compiler is self-hosted (written in itself). The compiler can be run on the command line, embedded in IDEs, or used in server mode in the Umple graphical user interface, as described below.

The second component of Umple is the UmpleOnline graphical user interface for editing Umple either diagrammatically or textually. This is a website that uses PHP to intercept requests from the front-end, which contains extensive JavaScript code. The PHP backend establishes socket connections to the Umple compiler (as in the last paragraph), which runs continually as an internal server, providing compilation microservices. UmpleOnline is live on the web [31] and is also distributed for local use via Docker.

3.2. Software functionality

The generated code is designed to be readable to allow for inspection if necessary, but it is expected that for the most part it will *not* be read. In other words, normal practice would be to compile Umple directly to an executable system, using the 'target' language only as a hidden intermediate form.

The Umple language is designed to look and feel like a C-family language, in that it uses curly brackets for blocks, and has a data typing scheme compatible with such languages. An Umple program includes the following top-level entities, with those marked with an asterisk also being able to be specified internally to classes.

- Classes and interfaces.
- Separation of concerns mechanisms: Traits [1] that allow classes to be built from parts; aspects *, to inject code into generated or user-specified methods; and mixsets * [23], a capability to structure the system using groups of named mixins to enable feature-oriented or product-line development.
- Use statements to incorporate multiple files or mixsets, and require statements to specify dependencies among mixsets or files.
- Associations [9], the standard UML constructs. *
- State machines [10], with unlimited nesting, concurrent regions, guards, entry actions, exit actions and concurrent do-activities (as per UML). *
- Enumerations. *

Classes and traits can include the following, in addition to those items marked with an asterisk above:

- Generalization statements (using the keyword 'isA') to specify the use of traits, interfaces or super-classes.
- Attributes as in UML, subject to constraints and various stereotypes controlling features such as uniqueness or immutability.
- Methods, whose bodies can have preconditions and postconditions specified in Boolean logic, as well as test code, and can have bodies in multiple programming languages (Java, C++, PHP) so they can be used to generate systems in multiple target languages.
- Constraints, which specify class invariants in Boolean logic, and which can refer to attributes and associations.
- Generation templates, describing patterns of textual output [16].
- Dependency declarations for external Umple code or the use of libraries in target languages.
- Directives specifying various class properties (and hence controlling what code is generated) such as immutability, being a singleton, and distributability.

Umple also makes extensive use of mixins [23] that allow additional constructs to be added to previously-defined classes, traits, state machines and other constructs. All of Umple's separation-of-concerns constructs (traits, mixins, mixsets, aspects) synergistically work together.

The core of the language is hence a textually-represented subset of UML But, as indicated above, Umple goes beyond UML in many ways. Its facilities for traits, aspects, mixins, and mixsets [23] depend on its textual nature, and could not be easily represented in a purely diagrammatic language.

A tool is available to reverse engineer (umplify [12,24]) systems to Umple, to reduce code volume and allow visualization.

Programs developed in Umple need to be compiled on the user's local machine, even if the public or Docker-based UmpleOnline platforms are used. There have been many requests to be able to execute programs on UmpleOnline servers, but this poses a security challenge. There is, however, a capability that allows a user using one keystroke to take a program in UmpleOnline, transfer it to their local computer, and compile it there, ready for running.

Users of Umple are supported by an online user manual [32] that has over 440 examples, each of which can be dynamically loaded into UmpleOnline. UmpleOnline also has many examples to help the user to learn Umple.

4. Related literature

There is extensive literature on the research and technologies that underpin Umple, mostly written by the seven PhD students and three masters students who have completed their theses focusing on Umple, or by the students whose thesis is still underway. The present paper is designed to bring it all together into a single citable paper covering the software itself as a whole; all the contributing graduate students are co-authors. Here we briefly mention only a subset of the literature, due to space limitations.

Early work focused on representing and generating good quality code for UML associations [9] and attributes [8]. Later work focused on state machines [10] (including queuing and pooling), tracing [6,41], traits [1], formal methods (Alloy and nuXmv) generation [3], concurrency [14], component-based modeling [15], text generation [16,17], distribution, and mixsets for product lines [23].

There is also literature about reverse engineering to Umple (umplification) [12,24] and the use of Umple in modeling education [19,25].

Umple is not the only textual modeling technology with full executability. TextUML [2], USE [13], SDL [28] and the many so-called formal methods languages are also well-known modeling technologies with a textual form. Executable UML is designed specifically to allow executability. These tools tend, however, to be limited to a certain IDE such as Eclipse, to be narrow in focus (e.g. USE focuses on associations and constraints), to not blend with traditional programming languages, to

be complex to use, and/or to not be free and open source. Umple has been designed to blend some of the best aspects of these technologies, and to overcome many of their key limitations.

5. Implementation and empirical results

5.1. Implementation details

The following gives details of Umple's design. More details, including dynamically generated diagrams, can be found online [33].

As discussed in Section 3.1, Umple has several distinct components, the most important being the compiler and the web-based user interface (UmpleOnline) [31]. There is also an Eclipse plugin and an extension for Microsoft Visual Studio Code.

The Umple compiler is written in Umple; its first version was in Java, but it was umplified [24] as soon as the first version was complete. Umple incorporates its own parsing engine (replacing Antlr in the original version), and its own text generation templating feature [17] for code generation.

The compiler follows a fairly standard compiler architecture. It is deployed as Java JAR file. The grammar [35] is parsed when the system is built, to create the parsing engine for instances of Umple code. The same grammar file is also used to generate documentation in the user manual, ensuring that the documentation is always 100% accurate.

The Umple compiler parses a set of Umple files (with suffix '.ump') as input and transforms them into an internal abstract syntax tree (AST), whose nodes and arcs are the instances of the classes and associations in Umple's metamodel [39].

Analysis is performed during and immediately after parsing, and numerous errors and warnings can be raised as a result. Each of these is listed in a key file called en.error [37], and they include both syntax errors and semantic errors (e.g., creating a cycle in the inheritance hierarchy). Documentation, with executable examples, is provided describing each of the errors and warnings [38]. The analysis phase results in some post-processing of the AST, converting it into an Abstract Syntax Graph (ASG).

Following creation of the ASG, the compiler invokes one of many user-selected generators that transform the ASG into outputs including diagrams of various types (class diagrams, state machines, entity-relationship diagrams, and others), code in various target languages (Java, C++, PHP, Ruby, nuXmv, and Alloy), as well as various other outputs such as diagrams, documentation, sample state machine execution sequences, and metrics. Selection of the generator can be performed through UmpleOnline, by specifying the generator in the Umple code itself, or using a command-line argument.

Both the Umple parser and the Umple textual generation technology (UmpleTL) are native to Umple and can be used by other programs written in Umple.

The compiler can also accept both a program and a set of edit operations that originate from a diagram editor.

In addition to accepting input via standard-input, the compiler can also operate as a server. In this case the compiler accepts compilation commands and returns results through a socket. This mechanism is used by UmpleOnline to sustain the high throughput demanded by its peak usage (thousands of compilation requests an hour).

The second Umple component is its graphical user interface. This takes the form of a website, but can be run locally on any computer using Docker [34]. UmpleOnline enables simultaneous editing of text and diagrams as well as generation and display of any of the compiler's many outputs. It supports several diagramming plugins; for example, for class diagrams there is a native diagramming tool, but it also supports a GraphViz generator and a joint.js editor.

The front-end of UmpleOnline is written using Javascript and JQuery. This makes web-service calls to a PHP backend. The PHP backend routes commands to a running Umple server. Although many people run UmpleOnline locally on their machines using Docker, the main UmpleOnline website [31] hosted in the cloud has proved capable of handling over 150,000 commands in some 24-hour periods. These peaks tend to occur when very large classes of students at several universities are using Umple for assignments and projects with tight deadlines. UmpleOnline hosts over 200,000 user sessions per year, and executes over 2 million commands per year. This does not include personal usage of the downloaded Jar, or usage in Docker or Eclipse; these forms of usage are not tracked.

5.2. Development process, testing and build system

The Umple codebase is hosted on Github. Umple follows standard agile open-source best practices with a few modifications to account for model-driven development using itself reflectively.

New developers wishing to contribute to Umple are asked to follow the instructions for developer setup in the Umple Wiki, and are then trained on small enhancements or bug fixes before they may contribute significant changes. Each proposed change must be documented in an issue, where discussion occurs about any proposed design. Developers create a branch for each issue, and when they have solved the issue they create a pull request for the set of commits that solve the issue (or part of it). Each pull request for the compiler or generators must include enhancements to the automated tests.

A full build involves the following:

- Compiling the Umple code (as changed), including code for the various generators and the parser, and then generating a new compiler.
- Compiling the code again with the newly-generated compiler (the compiler hence performs a self-test).
- Running tests in several categories: a) Tests of parsing all Umple constructs, including generating any error messages and constructing the internal abstract syntax tree; b) tests for generating outputs (e.g., code or diagrams) that appear as expected; c) tests that generated code executes as expected in languages such as Java and PHP; d) tests that failures of generated code can be debugged as expected; e) tests of that all 450 examples in the user manual compile as expected. Overall, over 6000 tests are run.

The developer is expected to achieve a zero-failure build before submitting any pull request. All branch commits and pull requests are tested on Travis and Appveyor in the MacOS, Linux and Windows environments before merging. Pull requests must also undergo a code inspection. Once all these checks pass, a new version of Umple is created, and becomes immediately available for use. Stable builds are also released a couple of times a year.

Umple has been developed by over 60 students. In addition to the graduate students (who are co-authors), 60 undergraduate students from many universities participated in their capstone projects through the UCOSP program [30,20]. The undergraduate students are listed in the Acknowledgments section.

5.3. Empirical results

A variety of empirical studies have been performed with Umple. We summarize them here.

In a paper dating back to 2011 [25], we showed that students found Umple useful in the classroom, and that it improved students' comprehension of modeling and their grades: Final exam grades for questions regarding drawing UML class diagrams increased from 76.4% to 83.6%.

Badreddin and Lethbridge [11] performed a study comparing the comprehensibility of systems written in Umple (as text), UML (as diagrams), and in Java. Systems in Umple were as easy to understand as the same system in UML diagrams, and much easier to understand than the same systems written in Java.

Between 2016 and 2017 we surveyed 125 professors [4] and 117 students [5] in many countries regarding their use of software modeling tools including Umple. Umple scored well, as compared to other tools, with respect to features such as simplicity and support for code generation.

In 2017, Leibel, Badreddin, and Heldal [40] compared the use of Papyrus [18] and Umple in student modeling projects. They report, "There is a clear endorsement for Umple ... regarding its impact on the project, with 65% agreeing that the tool affected their project positively and about 20% disagreeing." They then say, "65% agree and 19% disagree that Umple is easy to learn. ... 39% agreeing and 48 % disagreeing that Papyrus is easy to learn."

6. Illustrative examples

We give two examples, each with a sample of Umple code and some of the generated output. In both cases, the Umple keywords are in `red` and raw target language code (here Java) that will be transferred to the generated code without change (or as-is), is in `dark blue`.

The first example corresponds to the class diagram shown in Fig. 1, which is generated as the text is edited. The code for this can be found in the Umple user manual in the 'Hello World' page [36] where the user can interact with it. If the user clicks 'Generate Java', 425 lines of Java appear, 325 of which would need to be written by hand if Umple were not used (the remaining 100 are generated to facilitate debugging of the original Umple code, as opposed to the generated code).

There are three classes, starting on lines 1, 7, and 10. Line 16 illustrates Umple's mixin capability, whereby multiple class definitions are combined. Lines 8 and 11 show how generalization is specified in Umple. Lines 13-15 show the representation of a UML association. The code generated from associations properly manages referential integrity (i.e., Students know their Mentor, and Mentors know their Students).

The second example shows a UML state machine for a canal, which is one of the live examples in UmpleOnline [31]. The State machine name lockState is shown on line 7. State names are in `blue-green`, and event names (which generate methods to be called by hardware or user interface) are in `brown`. Arrows (−>) indicate which event leads to which resulting state. Guards, controlling whether or not a transition will be taken are shown in square brackets. The entry keyword specifies actions to be taken on entering a state, and the after keyword specifies a event to be triggered after a time delay in milliseconds. It can be seen from this example that many features of Umple can be interwoven.

A total of 483 lines of Java are generated from these 50 lines of Umple.

Fig. 2 shows a state table, generated from the Umple code. This table provides a way for developers to visualize and manually validate their model, and is one of a variety of potential outputs of UmpleOnline. Fig. 3 shows the UmpleOnline user interface enabling editing the model, with part of the generated code visible at the bottom, and the generated state diagram on the right.

```
1  class Person {
2    name; // Attribute, string by default
3    String toString () {
4      return(getName());
5    }
6  }
7  class Student {
8    isA Person;
9  }
10 class Mentor {
11   isA Person;
12 }
13 association {
14   0..1 Mentor -- * Student;
15 }
16 class Person {
17   // Notice that we are defining more contents for Person
18   // This uses Umple's mixin capability
19   public static void main(String [ ] args) {
20     Mentor m = new Mentor("Nick The Mentor");
21     Student s = new Student("Tom The Student");
22     s.setMentor(m);
23     System.out.println("The mentor of " + s  + " is " +  s.getMentor());
24     System.out.println("The students of "+  m  + " are " +  m.getStudents());
25   }
26 }
```

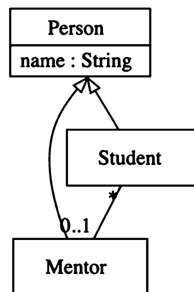


Fig. 1. Class diagram generated from the Student-Mentor example using the GraphViz generator.

7. Conclusions

Umple is a software tool designed to improve the productivity of software engineers and the quality of their products. It is unique in several ways: It is the only open-source software modeling tool written in itself; it embraces synergies between textual notations, diagrams, and other representations of systems; it is cross-platform and can be used without dependence on any particular integrated development platform (IDE); it embeds code in multiple target languages and has multiple inter-operating separation-of-concerns capabilities such as traits, mixins, mixsets, and aspects. Umple also has many other features such as comprehensive analysis of models.

Umple is targeted to agile open-source developers and software engineering education, but can be used for development of any type of software. For developers, it is designed to work with their best practices such as test-driven development, change management, and continuous integration. For professors and students, it enables learning of software modeling, facilitated by an extensive manual and examples, and motivated by the ability to create fully functional systems of any size.

Umple is used in education in universities in North America, Europe, Australia and New Zealand. We are also aware of its use to develop a variety of closed-source industrial software in such domains as statistical analysis, scheduling, contact management, and learning management.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

```
1   class Lock
2   {
3     Boolean boatGoingDown = false;
4     Boolean boatGoingUp = false;
5     Boolean boatBlockingGate = false;
6   
7     lockState {
8       BothDoorsClosedLockFull {
9         // Waiting for boat
10        boatRequestsToEnterAndGoDown -> OpeningUpperGate;
11        boatRequestsToEnterAndGoUp -> LoweringWater;
12      }
13      OpeningUpperGate {
14        upperGateFullyOpen -> UpperGateOpen;
15      }
16      UpperGateOpen {
17        entry / {setBoatGoingUp(false);}
18        boatInLockRequestingToGoDown -> / {setBoatGoingDown(true);}
19                                                  ClosingUpperGate;
20        after(180000) [!boatBlockingGate] -> ClosingUpperGate;
21      }
22      ClosingUpperGate {
23        upperGateFullyClosed [boatGoingDown] -> LoweringWater;
24        upperGateFullyClosed [!boatGoingDown] -> BothDoorsClosedLockFull;
25      }
26      LoweringWater {
27        waterLevelMatchesDownStream -> OpeningLowerGate;
28      }
29      BothDoorsClosedLockEmpty {
30        // Waiting for boat
31        boatRequestsToEnterAndGoUp -> OpeningLowerGate;
32        boatRequestsToEnterAndGoDown -> RaisingWater;
33      }
34      OpeningLowerGate {
35        lowerGateFullyOpen -> LowerGateOpen;
36      }
37      LowerGateOpen {
38        entry / {setBoatGoingDown(false);}
39        boatInLockRequestingToGoUp -> / {setBoatGoingUp(true);} ClosingLowerGate;
40        after(180000) [!boatBlockingGate] -> ClosingLowerGate;
41      }
42      ClosingLowerGate {
43        lowerGateFullyClosed [boatGoingUp] -> RaisingWater;
44        lowerGateFullyClosed [!boatGoingUp] -> BothDoorsClosedLockEmpty;
45      }
46      RaisingWater {
47        waterLevelMatchesUpStream -> OpeningUpperGate;
48      }
49    }
50  }
```

	BothDoorsClosedLockFull	**OpeningUpperGate**	**UpperGateOpen**	**ClosingUpperGate**
BothDoorsClosedLockFull		boatRequestsToEnterAndGoDown		
OpeningUpperGate			upperGateFullyOpen	
UpperGateOpen				boatInLockRequestingToGoDown after(180000) [!getBoatBlockingGate()]
ClosingUpperGate	upperGateFullyClosed [!getBoatGoingDown()]			
LoweringWater				
BothDoorsClosedLockEmpty				
OpeningLowerGate				
LowerGateOpen				
ClosingLowerGate				
RaisingWater		waterLevelMatchesUpStream		

Fig. 2. Part of the state table generated by Umple from the Canal example. Rows are states; cells are events that cause a state transition from that state; columns are resulting states. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

Fig. 3. High-level view of the UmpleOnline user interface showing the state machine example, with the diagram generated by GraphViz, and some generated code.

Acknowledgements

The authors of this paper are the lead professor and graduate students who have worked on the Umple compiler since its inception. However, contributions have also been received from numerous others, including masters student Julian Solano, the following 4th year students through the UCOSP program [30,20] (listed in Chronological order of contribution): Joshua Horacsek, Joel Hobson, Alvina Lee, Jordan Johns, Sonya Adams, James Zhao, Adam Dzialoszynski, Luna Lu, Song Bae Choi, Thomas Morrison, Sacha Bagasan, Andrew Paugh, Stuart Erskine, Russell Staughton, Christopher Hogan, Geoffrey Guest, Gabriel Blais Bourget, Robin Jastrzebski, Quinlan Jung, Blakeley Quebec Desloges, Tianyuan Chu, Fiodar Kazhamiaka, Greg Hysen, Jean-Christophe Charbonneau, Kenan Kigunda, Adriaan Cody Schuffelen, Marc Antoine Gosselin-Lavigne, Pedro Augusto Vincente, Ellen Arteca, Alexi Turcotte, Karin Ng, Mark Galloway, Alexander Ringeri, Eric Telmer, Charles Wang, Chan Chun Kit, Nabil Maadarani, John Zweip, Kevin Brightwell, Warren Marivel, Ashley Merman, Xinxin Kou, Aymen Ben Rkhis, Curtis Meerkerk, Adam Kereliuk, Matthew Fritze, Michael Mkicik, Victoria Lacroix, Morgan Redshaw, Matthew Rodusek, Shikib Mehri, Marc de Niverville, Alex Hochheiden, Noah Murad, Katharine Cavers, Jackie Lang, Adam Bolding Jones, Chang Ding, Joshua McManus, Balaji Venkatesh, Runqing Zhang, Finn Hackett, Daniel Mitchell, Richard Hugessen, Bowei (Bernard) Yuan, Gloria Law, Yiran Shu, Evgeniya Vashkevich, and Paul Wang.

These 4th year students have attended the Universities of Ottawa, Guelph, British Columbia, Regina, Saskatchewan, Sherbrooke, Waterloo, New Brunswick, Alberta, Toronto, Windsor, Lethbridge and Illinois, as well as Laurentian, Bishops, Simon Fraser, Dalhousie, Western, Wilfrid Laurier, Brock, Carnegie Mellon and Cornel Universities plus the Massachusetts Institute of Technology.

Several postdocs, visiting professors, employees and interns have also contributed to Umple: Ali Fatolahi, Antonio Resende, Julie Filion, Jesus Zambrano, Tiago Nascimento, Craig Bryan, Jason Canto, Zainab Al Showely, Firas Jribi and Jingyi Pan.

Funding for the travel of the undergraduate students was provided by Google and Facebook through the UCOSP and Facebook Open Academy programs. Funding for the graduate students was from NSERC under grants 453224, 483509, 569913, and 657301 as well as from the Ontario Research Fund under grant RE-05-044.

References

[1] V. Abdelzad, T.C. Lethbridge, Promoting traits into model-driven development, Softw. Syst. Model. 16 (4) (2017) 997–1017. Springer.
[2] Abstratt, TextUML, http://abstratt.github.io/textuml/readme.html, visited January 2021.
[3] O. Adesina, T.C. Lethbridge, S. Somé, Optimizing hierarchical, concurrent state machines in Umple for model checking, in: 16th Workshop on Model Driven Engineering, Verification and Validation (MoDeVVa), Models Companion Volume, Munich, September, IEEE, 2019, pp. 523–531.
[4] L.T.W. Agner, T.C. Lethbridge, A survey of tool use in modeling education, in: 2017 ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS), IEEE, 2017, September, pp. 303–311.
[5] L.T.W. Agner, T.C. Lethbridge, I.W. Soares, Student experience with software modeling tools, Softw. Syst. Model. 18 (2019) 3025–3047. Springer.
[6] H. Aljamaan, T.C. Lethbridge, O. Badreddin, G. Guest, A. Forward, Specifying trace directives for UML attributes and state machines, in: 2nd International Conference on Model-Driven Engineering and Software Development (MODELSWARD), January, IEEE, 2014, pp. 79–86.
[7] Astah, Visualize your ideas: UML and Data Modeling and diagramming tools for the enterprise, https://astah.net, visited January 2021.
[8] O. Badreddin, A. Forward, T.C. Lethbridge, Exploring a model-oriented and executable syntax for UML attributes, in: Software Engineering Research, Management and Applications, Springer, 2014, pp. 33–53.
[9] O. Badreddin, A. Forward, T.C. Lethbridge, Improving code generation for associations: enforcing multiplicity constraints and ensuring referential integrity, in: Software Engineering Research, Management and Applications, Springer, 2014, pp. 129–149.
[10] O. Badreddin, T.C. Lethbridge, A. Forward, M. Elaasar, H. Aljamaan, M.A. Garzon, Enhanced code generation from UML composite state machines, in: 2014 2nd International Conference on Model-Driven Engineering and Software Development (MODELSWARD), IEEE, 2014, January, pp. 235–245.
[11] O. Badreddin, T.C. Lethbridge, Combining experiments and grounded theory to evaluate a research prototype: lessons from the Umple model-oriented programming technology, in: First International Workshop on User Evaluation for Software Engineering Researchers, IEEE, 2012, June, pp. 1–4.
[12] M.A. Garzón, T.C. Lethbridge, H. Aljamaan, O. Badreddin, Reverse engineering of object-oriented code into Umple using an incremental and rule-based approach, in: 24th Annual International Conference on Computer Science and Software Engineering, ACM, 2014, November, pp. 91–105.
[13] M. Gogolla, F. Büttner, M. Richters, USE: a UML-based specification environment for validating UML and OCL, Sci. Comput. Program. 69 (2007) 27–34. Elsevier.
[14] M. Husseini Orabi, A. Husseini Orabi, T.C. Lethbridge, Concurrent Programming Using Umple, MODELSWARD, 2018, pp. 575–585. SCITEPRESS.
[15] M. Husseini Orabi, A. Husseini Orabi, T.C. Lethbridge, Component-Based Modeling in Umple, MODELSWARD, 2018, pp. 247–255. SCITEPRESS.
[16] M. Husseini Orabi, A. Husseini Orabi, T. Lethbridge, Umple as a Template Language (Umple-TL), in: Proceedings of the 7th International Conference on Model-Driven Engineering and Software Development, SCITEPRESS, 2019, February, pp. 96–104.
[17] M. Husseini Orabi, A. Husseini Orabi, T.C. Lethbridge, Umple-TL: a model-oriented, dependency-free text emission tool, Commun. Comput. Inf. Sci. 1161 (2020) 127–155. Springer.
[18] A. Lanusse, Y. Tanguy, H. Espinoza, C. Mraidha, S. Gerard, P. Tessier, F. Terrier, Papyrus UML: an open source toolset for MDA, in: Proc. of the Fifth European Conference on Model-Driven Architecture Foundations and Applications (ECMDA-FA 2009), 2009, June, pp. 1–4.
[19] T.C. Lethbridge, Teaching modeling using Umple: principles for the development of an effective tool, in: 2014 IEEE 27th Conference on Software Engineering Education and Training (CSEE&T), IEEE, 2014, April, pp. 23–28.
[20] T.C. Lethbridge, Capstone software engineering students can develop a high-quality complex system: a case study with Umple, in: Canadian Engineering Education Conference, 2019, https://doi.org/10.24908/pceea.vi0.13730.
[21] T.C. Lethbridge, A. Algablan, Umple: an executable UML-based technology for agile model-driven development, in: IGI Global, Advancements in Model-Driven Architecture in Software Engineering, IGI Global, 2020, pp. 1–25.
[22] T.C. Lethbridge, V. Abdelzad, M. Husseini Orabi, A. Husseini Orabi, O. Adesina, Merging modeling and programming using Umple, in: International Symposium on Leveraging Applications of Formal Methods (Isola), Springer, 2016, pp. 187–197.
[23] T.C. Lethbridge, A. Algablan, Using Umple to synergistically process features, variants, UML models and classic code, in: International Symposium on Leveraging Applications of Formal Methods, Springer, 2018, November, pp. 69–88.
[24] T.C. Lethbridge, A. Forward, O. Badreddin, Umplification: refactoring to incrementally add abstraction to a program, in: 2010 17th Working Conference on Reverse Engineering, IEEE, 2010, October, pp. 220–224.
[25] T.C. Lethbridge, G. Mussbacher, A. Forward, O. Badreddin, Teaching UML using Umple: applying model-oriented programming in the classroom, in: 2011 24th IEEE-CS Conference on Software Engineering Education and Training (CSEE&T), IEEE, 2011, May, pp. 421–428.
[26] T.C. Lethbridge, J. Singer, A. Forward, How software engineers use documentation: the state of the practice, IEEE Softw. 20 (6) (2003) 35–39.
[27] M. Petre, UML in practice, in: 2013 35th International Conference on Software Engineering (ICSE), IEEE, 2013, May, pp. 722–731.
[28] A. Rockstrom, R. Saracco, SDL-CCITT specification and description language, IEEE Trans. Commun. 30 (6) (1982) 1310–1318.
[29] S. Sendall, J. Küster, Taming model round-trip engineering, in: Workshop on Best Practices for Model-Driven Software Development, 2004, pp. 1–23.
[30] E. Stroulia, et al., Teaching distributed software engineering with UCOSP: the undergraduate capstone open-source project, in: 2011 Community Building Workshop on Collaborative Teaching of Globally Distributed Software Development, ACM, 2011, pp. 20–25.
[31] UmpleOnline, https://try.umple.org, visited January, 2021.
[32] Umple User Manual, https://manual.umple.org, visited January 2021.
[33] Umple Architecture, http://architecture.umple.org, visited January 2021.
[34] Umple DockerHub, http://docker.umple.org, visited January 2021.
[35] Umple Grammar, http://grammar.umple.org, visited January 2021.
[36] Umple Hello World Examples, https://manual.umple.org?HelloWorldExamples.html, visited January 2021.
[37] Umple Master Error File, http://errors.umple.org, visited January 2021.
[38] Umple Messages, http://manual.umple.org?Umplemessages.html, visited January 2021.
[39] Umple Metamodel, http://metamodel.umple.org, visited January 2021.
[40] G. Liebel, O. Badreddin, R. Heldal, Model driven software engineering in education: a multi-case study on perception of tools and UML, in: 30th Conference on Software Engineering Education and Training (CSEE&T), Savannah, GA, Ipp, 2017, pp. 124–133.
[41] H. Aljamaan, T.C. Lethbridge, MOTL: a textual language for trace specification of, in: State Machines and Associations, 25th Annual International Conference on Computer Science and Software Engineering (Cascon), ACM, 2015, pp. 101–110.