Umple: A Framework for Model Driven Development of Object-Oriented Systems

Miguel A. Garzón*, Hamoud Aljamaan† and Timothy C. Lethbridge‡
School of Electrical Engineering and
Computer Science University of Ottawa
Ottawa, Canada
Email: *mgarzon@uottawa.ca, †hjamaan@uottawa.ca, ‡tcl@eecs.uottawa.ca

***Abstract*—Huge benefits are gained when Model Driven Engineering are adopted to develop software systems. However, it remains a challenge for software modelers to embrace the MDE approach. In this paper, we present Umple, a framework for Model Driven Development in Object-Oriented Systems that can be used to generate entire software systems (Model Driven Forward Engineering) or to recover the models from existing software systems (Model Driven Reverse Engineering). Umple models are written using a friendly human-readable modeling notation seamlessly integrated with algorithmic code. In other words, we present a model-is-the-code approach, where developers are more likely to maintain and evolve the code as the system matures simply by the fact that both model and code are integrated as aspects of the same system. Finally, we demonstrate how the framework can be used to elaborate on solutions supporting different scenarios such as software modernization and program comprehension.**

***Keywords*—UML; framework; Object Oriented; MDE; Umple; Reverse Engineering; Forward Engineering;**

I. Introduction

Model Driven Engineering (MDE) has become in recent years a promising approach to alleviate complexity in the development of modern software systems. In MDE, models are primary artifacts from which parts of the software can be automatically generated. With the increasing use of models for software development, the challenge becomes how to extract the model(s) from an existing system and how to maintain them during the complete development life cycle.

In this paper, we present a modeling framework named Umple [1, 2] that allows modelers to perform model driven engineering processes in a single framework. The Umple framework can be used to recover documentation and design from existing legacy systems, and/or can be adopted to perform forward engineering for modelers keen on following the MDE methodology. Case studies will be used to demonstrate how the framework can be used to increase modeler's comprehension of targeted systems and/or modernize their code base.

This tool demonstration paper flows in this sequence: Section 2 presents the Umple framework. Section 3 demonstrates the usage of Umple in open-source case studies. Section 4 concludes the tool demonstration.

II. Umple Framework

In this section, we provide an overview of the framework tools, and their underlying architecture, currently available to support the creation of systems modeled in Umple, and the analysis of systems in other base languages, converting them to Umple.

A. Architecture

The Umple toolset and language, which were originally written in Java, were fully rewritten in Umple in 2008, and have been developed and maintained in Umple since then. The Umple Framework has a layered and pipelined architecture as shown in Figure 1. The architecture is divided into two major parts corresponding to the main capabilities of the Umple framework. On one hand, Umple can generate base language code from Umple models. This process of generating code from an abstract model is known in the literature as forward model engineering. On the other hand, the Umple is also able to turn an existing system into an Umple system, a process called reverse model engineering. Both major components, for forward and reverse engineering are built and deployed separately using the Ant scripting language; resulting in several executable artifacts.

It should be noted that the intended use of Umple is forward engineering, with the Umple source being the master artifact. The reverse engineering process is intended to convert existing systems to Umple, whereupon they would be maintained in a forward engineering manner.

The development of Umple follows a test-driven development approach to provide a confident way of: 1) adding new features to the system 2) making minor modifications to an existing feature and 3) resolving defects.

B. Forward Engineering

The forward engineering components of Umple includes: a parser, an analyzer as well as several code-generators and model-to-model transformation engines. These are described below:

1) **Parser**: The parser receives an input model, written in Umple language, tokenizes it and passes it to the next component in the pipeline.
2) **Analyzer**: This component processes the tokens previously obtained and converts them into an internal representation consistent with Umple's metamodel.
3) **Code-Generator(s)**: The internal representation is then translated into other artifacts; either additional models like Papyrus XMI, EMF, Yuml, Xuml; various

diagrams, or source code such as Java, C++, PHP, Ruby or SQL. The compiler generates various types of methods including mutators (to control changes to a variable) and accessors (to return the value of a variable) from the various Umple features. Sophisticated code for managing state machines, tracing [3], generation templates, patterns, aspects and concurrency can also be generated from models.

Each component is tested independently to ensure that the input is processed correctly and the output produced is valid. Testing the Umple parser is centered on tokenization of Umple code. Testing the metamodel classes ensures that the analyzer component produces valid metamodel instances. Testing of generated systems is also performed.

In addition to the Eclipse plugin and command-line based compiler, UmpleOnline [4], a web-based application, allows to instantly experiment with Umple on the Web.

C. Reverse Engineering

The process of reverse engineering as performed by the Umple framework, involves recursively modifying the code to incorporate additional abstractions, while maintaining the semantics of the program. Furthermore, the reverse engineering is performed in multiple small steps that produce a new version of the system with additional modeling information. Our approach proceeds incrementally, performing additional transformations until the desired level of abstraction is achieved. The code comments and annotations are preserved. In fact, in the output source code, the comments and/or annotations are found at the same location (within the class body or a method body) as in the input source. The following Table gives a summary of the transformations currently implemented. Detailed information about the transformations and mapping rules implementing these transformations can be found in [5].

TABLE I. SUMMARY OF REVERSE-ENGINEER TRANSFORMATIONS

Transformation	**Description**
Initial :	Source files with language L (e.g. Java, C++) code are initially renamed as Umple files, with extension .ump.
Inheritance:	Transformation of generalization / specialization, dependency, and namespace declarations.
Attributes:	Transformation of variables to UML/Umple **attributes**.
Associations:	Transformation of variables in one or more classes to UML/Umple **associations**.
State Machines:	Transformation of variables to UML/Umple **state machines**.

The Reverse Engineering component, called the Umplificator, is available as a separate download. As with the Umple compiler (for forward engineering) the Umplificator works within Eclipse but it also operates as a command line tool to handle heavy workloads and make automated testing easier. The components of the Reverse Engineering part of the Umple Framework are described below:

1) **Parser**: The parser receives a set of source code files in the base language and/or Umple and creates an abstract syntax tree (AST) as representation of the code (Umple is allowed as input to allow repeated application to refine the model).
2) **Model Extractor**: The tree representation, previously obtained, is taken and a base-language model (input model) is built conforming to a base-language metamodel.
3) **Transformer**: The base-language model is transformed into an Umple model (output model) using a predefined set of mapping rules. If the input model is Umple code, the transformer produces an Umple model with additional modeling constructs (abstractions).
4) **Generator**: The Umple model is then validated and Umple code is generated from it.

The Umple Framework includes other subsidiary and internal tools such as:

- *Language validators* – A set of base language validators allowing validation of the base language code that is generated after compilation of the recovered/extracted Umple models.
- *Statistics* – A metrics-gathering tool to analyze certain aspects of a software system such as the number of classes and interfaces, the number of attributes and the number of associations (including the different types of associations: 1-to-1, 1-to-many, many-to-many, etc.).
- *Workflow* – A tool that guides the reverse engineering process within the Eclipse IDE.

III. DEMONSTRATION

Now, we demonstrate the Umple framework as a model driven forward and reverse engineering environment designed to ease the development and maintenance of Object oriented software systems. We discuss our experiences having two different goals in mind that will be introduced later. We have employed two different open source projects to achieve the anticipated goals. Demonstration goals are set as follows:

Goal 1. Program comprehension by Reverse Engineering and design recovery.

- Input: JHotdraw 7.5.1 [6]
- Output: An Umple System

We have reverse engineered JHotDraw, an open source graphics editor written in Java that makes extensive use of software patterns and has detailed documentation about its design. The analyzed version consists of 138 packages, 689 classes, 8720 methods, 1503 constructors and 81632 lines of code (LOC). For the purposes of this demo, we will show how Umple can be used as an aid in program understanding by extracting the model from JHotDraw. A slice of the model is presented below.

Figure 2 presents the reverse engineering output from package <org.jhotdraw.draw> of JHotDraw after performing the initial transformation. This package contains, as seen in the UML class diagram (which is the visual representation of the Umple model obtained), the Figure classes. Figure is a central abstraction of the drawing editor framework. The UML class

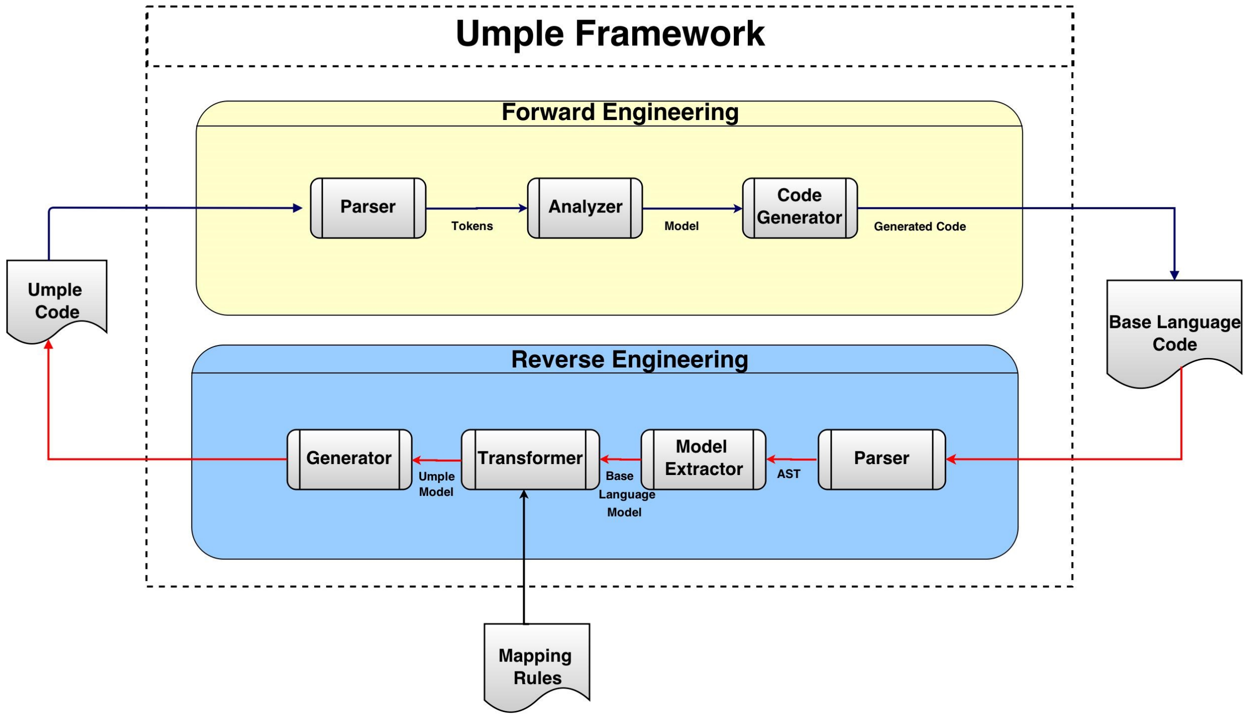


Fig. 1: Umple Framework Architecture

diagram in Figure 2 shows the Figure, CompositeFigure, DecoratorFigure, ConnectionFigure, Connector, and Handle classes but not the attributes of each class and/or their relationships with other classes in the package (associations).

Figure 3 shows a visual representation of the Umple model (a slice of it) obtained from the same package <org.jhotdraw.draw> but this time after transforming the variables into attributes or associations and refactoring the required code. This Umple model contains now more abstractions allowing us to get a better understanding of the classes in the package and how a refactoring or a change could affect them.

Goals 2: Modernization of an existing system into a different programming language.

- Input: args4j [7]
- Output: An Umple System having the modeling related code separated from the algorithmic and logic code.

We have reverse engineered args4j, a small library that enhances the parsing of command line, options and arguments in any Java application. For every class in args4j, our reverse engineering tool has produced two different files. The model design is expressed as an Umple model (i.e. Model.ump) and any algorithmic code is separated from the model as extra code (i.e. Model_code.ump). Developers can now use our forward engineering technology to generate their system in any chosen programming language from the list of available languages in Umple framework. If the modeler chooses to reproduce their system in the same language as it was before the reverse engineering process, we claim that the generated code will be of higher quality for a couple of reasons: First, we follow a rigorous test driven development approach in all of our framework components to ensure quality. Second, we have a state-of-the art code generator that respects associations multiplicity constraints and referential integrity [8], [9], and supports complex state machine code generation [10]. Further details on the Umple API generated from various Umple constructs can be found at [11]. In another scenario, if the modeler chooses another programming language, as a target language, to generate the system, then they need to manually convert the algorithmic code in (files of type Model_code.ump) to the new language.

We have instrumented our forward and reverse engineering framework components with timers to measure the time taken to process an input file and produce the target source code. More specifically, in the forward engineering module, the timer measures the time taken to 1) parse an input file, 2) to analyze and build an instance of the Umple metamodel and 3) to generate code which involves creating a file (.Java, .C++, etc.). In the reverse engineering module, the timer measures the time taken to 1) parse an input file, 2) to build (extract) an instance of the base language model, 3) to transform the input model based on a predefined set of mapping rules into an Umple model, and 4) to generate code which involves creating files with the .ump extension. Table 2 summarizes the execution times in milliseconds taken to reverse engineer JHotDraw and args4j.

Table 3 presents the execution times in milliseconds taken to generate Java code from the umplified versions of JHotDraw and args4j. The execution times have been split to reflect the main stages of the transformation process, as explained in Section 2. Note that the reverse engineering component uses Perf4J [12] for calculating (and displaying) performance statistics. Table 2 displays the average of multiple execution times as gathered by this utility.

TABLE II. REVERSE ENGINEERING EXECUTION TIMES

	Execution Time (in ms)	
Component	**JHotDraw**	**Args4j**
Parsing	50899	12500
Extractor	21025	3204
Transformer	339327	920
Generating Umple Code	1700	450
Total Time:	412951	14074

The tests were executed on a machine exhibiting the following characteristics:

- Intel Core i7-4500 CPU @ 3.10GHz
- RAM: 8.00 GB
- Windows 8 - 64 bits

TABLE III. FORWARD ENGINEERING EXECUTION TIMES

	Execution Time (in ms)	
Component	**JHotDraw**	**Args4j**
Parsing	23193	2333
Analyzing	14120	205
Generating Java Code	1430	430
Total Time:	38743	2968

The transformation (Table 2) and code generation (Table 2 and 3) results depend on the number of rules applied (based on the level of refactoring achieved) and on the size of input system (138 classes and 22 interfaces as for JHotDraw, for instance) and this explains the variations in the execution times from one system to another. Furthermore, for args4j where the code contains only a few instance variables that can become Umple constructs, the transformation stage has been performed very fast compared to the corresponding time for JHotdraw, a very rich system in terms of modeling abstractions.

The video demonstration can be found at this link http://screencast.com/t/hwbUqvFhPri5.

IV. CONCLUSION

In this paper, we presented a model-driven framework called Umple that allows performing both forward and reverse engineering. Using Umple, developers are provided with features that can help them:

1) **Create** UML models textually. It can often be faster to create UML class diagrams and state machines using Umple's textual format that looks just like programming-language code.
2) **Generate** high quality code from the UML/Umple models.
3) **Incrementally convert** an existing system into Umple.
4) **Modernize** an existing system.
5) **Recover** the documentation of an existing system. One direction for future work would be to apply the reverse engineering approach to other open source systems to improve the detection and analysis mechanisms of our reverse engineering tool. We also will improve Umple tools to support better IDE integration including model-level debugging.

One direction for future work would be to apply the reverse engineering approach to other open source systems to improve the detection and analysis mechanisms of our reverse engineering tool. We also will improve Umple tools to support better IDE integration including model-level debugging.

ACKNOWLEDGMENT

Hamoud Aljamaan would like to thank King Fahd University of Petroleum and Minerals (KFUPM) for their financial support during his PhD studies.

REFERENCES

[1] A. Forward, O. Badreddin, T. C. Lethbridge, and J. Solano, "Model-driven rapid prototyping with Umple," *Software: Practice and Experience*, vol. 42, no. 7, pp. 781–797, Jul. 2012.

[2] O. Badreddin, A. Forward, and T. C. Lethbridge, "Model oriented programming: an empirical study of comprehension," in *CASCON '12 Proceedings of the 2012 Conference of the Center for Advanced Studies on Collaborative Research*. IBM Corp., Nov. 2012, pp. 73–86.

[3] H. Aljamaan, T. C. Lethbridge, O. Badreddin, G. Guest, and A. Forward, "Specifying Trace Directives for UML Attributes and State Machines," in *2nd International Conference on Model-Driven Engineering and Software Development*, Jan. 2014, pp. 79–86.

[4] "UmpleOnline: Generate Java, C++, PHP, or Ruby code from Umple." [Online]. Available: try.umple.org

[5] M. A. Garzon, T. C. Lethbridge, H. Aljamaan, and O. Badreddin, "Reverse engineering of object-oriented code into umple using an incremental and rule-based approach," in *Proceedings of CASCON'14*, ser. CASCON '14. Toronto, Ontario, Canada: IBM Corp., 2014, pp. 91–105.

[6] E. Gamma and T. Eggenschwiler, "JHotDraw." [Online]. Available: http://www.jhotdraw.org/

[7] "args4j." [Online]. Available: http://args4j.kohsuke.org/

[8] O. Badreddin, A. Forward, and T. Lethbridge, "Exploring a Model-Oriented and Executable Syntax for UML Attributes," *Software Engineering Research, Management and Applications SE - 3*, vol. 496, pp. 33–53, 2014.

[9] O. Badreddin, A. Forward, and T. C. Lethbridge, "Improving Code Generation for Associations: Enforcing Multiplicity Constraints and Ensuring Referential Integrity," in *SERA 2013*. Springer, 2013, pp. 129–149.

[10] O. Badreddin, T. C. Lethbridge, A. Forward, M. Elasaar, and H. Aljamaan, "Enhanced Code Generation from UML Composite State Machines," in *Modelsward 2014 - Proceedings of the 2nd International Conference on Model-Driven Engineering and Software Development*, 2014, pp. 235–245.

[11] "Umple generated API." [Online]. Available: http://cruise.eecs.uottawa.ca/umple/APISummary.html

[12] Perf4j, "Perf4j - performance statistics for Java code." [Online]. Available: http://perf4j.codehaus.org/

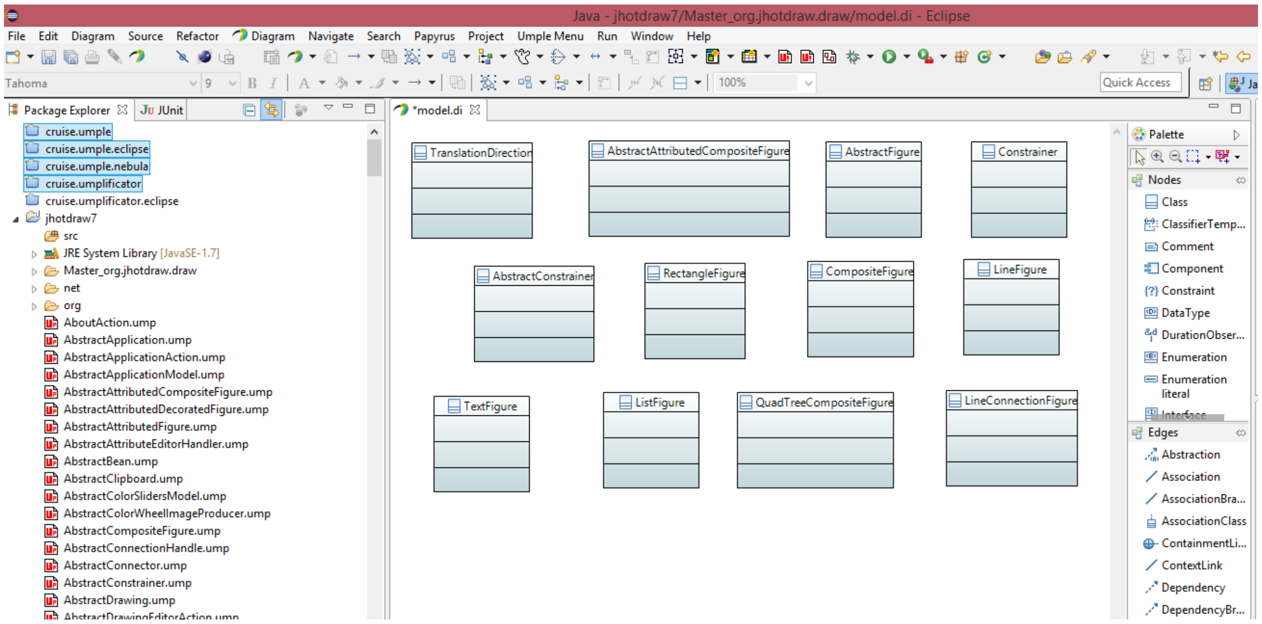
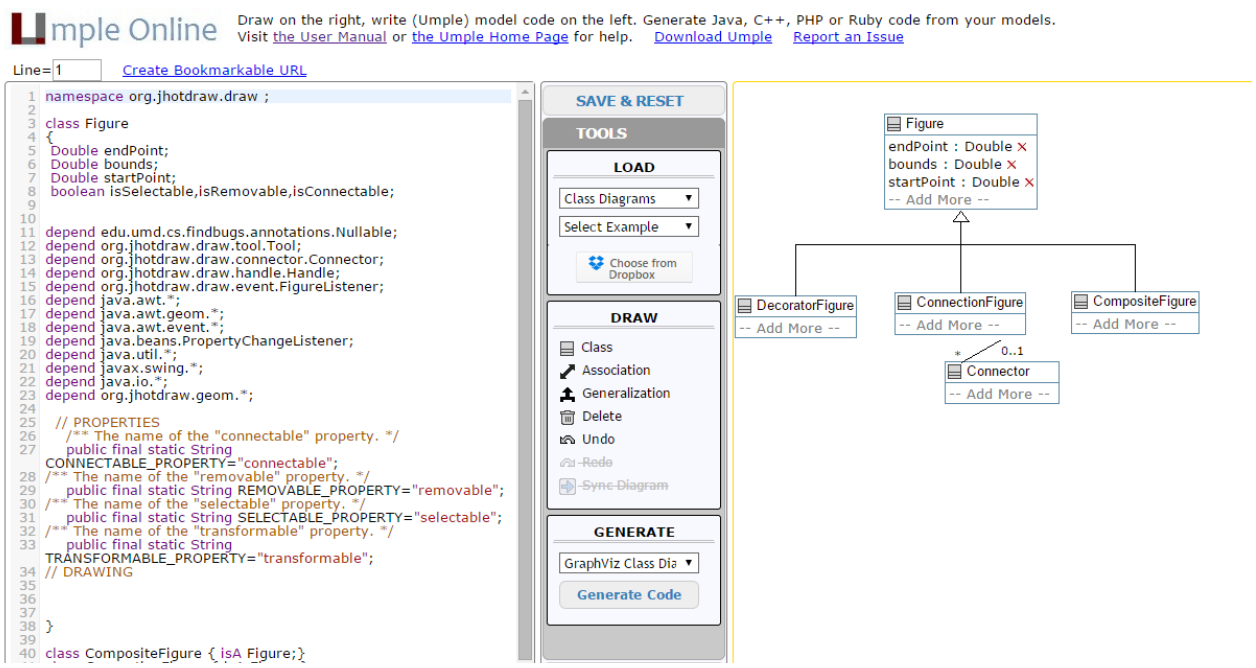


Fig. 2: Umplified package <org.jhotdraw.draw> from JHotDraw



Draw on the right, write (Umple) model code on the left. Generate Java, C++, PHP or Ruby code from your models.
Visit the User Manual or the Umple Home Page for help. Download Umple Report an Issue

Line=1 Create Bookmarkable URL

```
namespace org.jhotdraw.draw ;

class Figure
{
 Double endPoint;
 Double bounds;
 Double startPoint;
 boolean isSelectable,isRemovable,isConnectable;


depend edu.umd.cs.findbugs.annotations.Nullable;
depend org.jhotdraw.draw.tool.Tool;
depend org.jhotdraw.draw.connector.Connector;
depend org.jhotdraw.draw.handle.Handle;
depend org.jhotdraw.draw.event.FigureListener;
depend java.awt.*;
depend java.awt.geom.*;
depend java.awt.event.*;
depend java.beans.PropertyChangeListener;
depend java.util.*;
depend javax.swing.*;
depend java.io.*;
depend org.jhotdraw.geom.*;

  // PROPERTIES
    /** The name of the "connectable" property. */
    public final static String
CONNECTABLE_PROPERTY="connectable";
/** The name of the "removable" property. */
    public final static String REMOVABLE_PROPERTY="removable";
/** The name of the "selectable" property. */
    public final static String SELECTABLE_PROPERTY="selectable";
/** The name of the "transformable" property. */
    public final static String
TRANSFORMABLE_PROPERTY="transformable";
// DRAWING



}

class CompositeFigure { isA Figure;}
```

Fig. 3: Umplified package <org.jhotdraw.draw> with a higher level of abstraction drawn by UmpleOnline [4]