Software Specification and Testing Using UML and OCL

Jonathan Milley
Faculty of Engineering and Applied Science
MUN
St. John's, Newfoundland
Email: jmilley@engr.mun.ca

Dr. Dennis K. Peters
Faculty of Engineering and Applied Science
MUN
St. John's, NL
Email: dpeters@engr.mun.ca

Abstract

Software testing is an important and time consuming task for developers. Automating the testing tasks will allow developers to spend their time more productively. The Unified Modeling Language is widely used for designing object oriented software. With the advent of the Object Constraint Language UML allows not only description of the layout of software, but specification of behavior as well. Using documents created at design time, tools exist that will generate automated tests and oracles. While this implies a larger amount of work at the front end of the development cycle, there is a significant gain in quality and time by automating the testing phase.

I. Introduction

The Unified Modeling Language (UML) [2] is a standardized way to represent and design object oriented software systems. UML defines a set of diagrams that are used to define the structure and behavior of the software. The most basic diagram is the Class Diagram, in which you define the structure (classes, methods, member variables) of each class and their interaction with other classes. These interactions include many types from the "have" relation meaning an instance of an object has a member that is an instance of the other, to a "uses" relationship that means one object knows an instance of the other class and uses it, to a "derives from" meaning one class derives all the public methods and members from the parent.

Other UML diagrams include Use Case Diagrams which allow specification of behavior of methods in objects. A use case scenario is designed to show how the flow of method calls and instantiations occurs.

UML by itself is a powerful method of designing and documenting a software system. Many powerful tools exist to assist in creating and maintaining UML documents. The more advanced of which include features which allow translation of UML models into software code, and vice versa. A properly maintained UML model of a software system can greatly improve the ease of writing and maintaining the code.

The Object Constraint Language (OCL) is another modeling specification designed and maintained by the Object Management Group (OMG) which created and maintains the UML standard. OCL is designed as an add-on to UML. The OCL is meant to convey information that cannot be represented in diagrams. The OCL is chiefly used to write specifications on behaviour, expressions such as invariants, preconditions and postconditions.

OCL expressions are associated with classes in the model, and are written using objects and items from the diagrams. The language itself is independent of the target programming language. The constraints placed upon the classes can be translated into code along with the UML model to add code for ensuring the software conforms to the constraints specified upon it.

The Model Driven Architecture (MDA) is an initiative by the OMG to define a process for software development with UML and OCL as core techniques [8]. It takes the process from design (UML) to specification (OCL) to code (using translation). It also defines a platform independent way of design, using translation from a Platform Independent Model (PIM) to a Platform Specific Model (PSM) and then to code. This PIM to PSM translation allows greater ease of cross platform development by maximizing the amount of work that can be shared.

The UML and OCL models of a software system not only assist in developing the code and maintaining the system, they can also be leveraged in testing the software. The cost (in time and money) of a properly created and maintained UML/OCL model can be greatly outweighed by its multifaceted uses later in the development process.

II. Why use OCL?

The addition of OCL statements to UML documents addresses the deficiencies of UML and adds capabilities that bring the design and development stages closer together. Such deficiencies in UML as the inability to place exact constraints upon multiplicity of relationships. For example the ”has” relationship can be defined as one of 1 to 1; 1 to many; 1 to 0 or 1; and 1 to any. In UML there is no way to limit this to say one object has 1-4 of another. For example see figure 1, a UML model showing a multiplicity relating a car to its wheels. We would like to limit the size of myWheels to just four, as a car can only have four tires, see listing 1 for an example OCL expression to accomplish this.

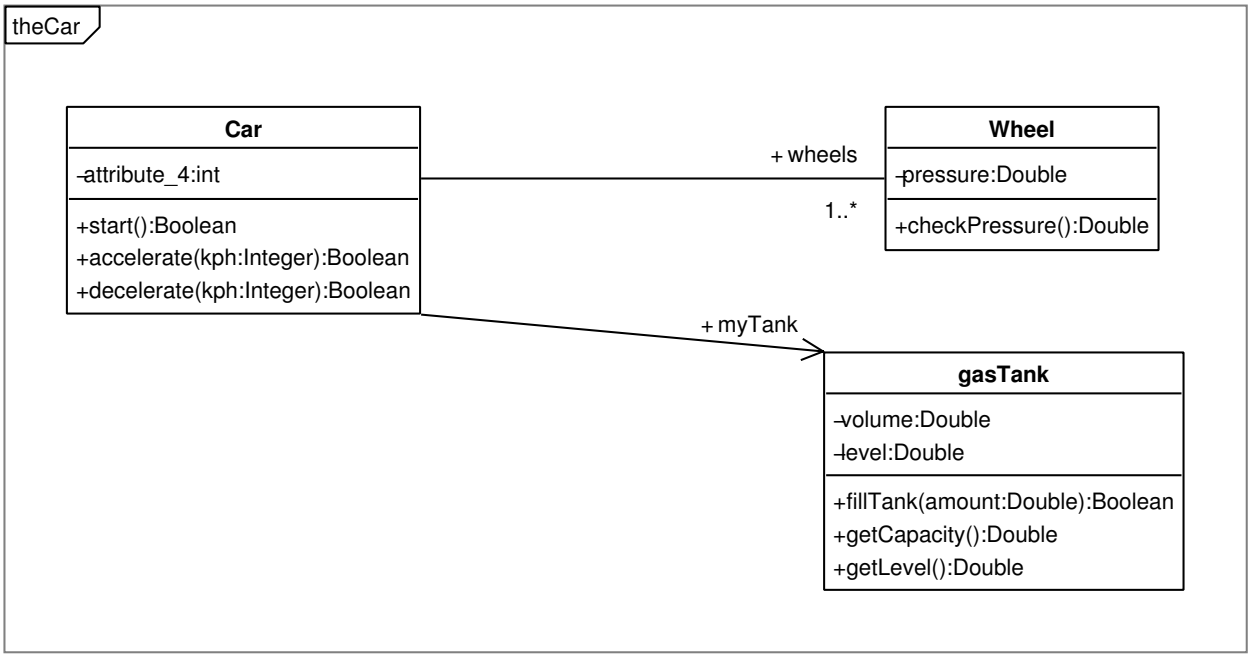

Fig. 1. Sample UML Model

Listing 1. Example OCL Expression

```
context Car
        inv: myWheels->size() = 4
```

As you can see from the second example in figure 2, taken from [8], OCL can add a significant amount of detail to a UML model. The mortgage system UML relates houses, people and mortgages in some fashion, but it doesn't put that relationship in context. Adding OCL expressions as in listing 2 adds context such as every person must have a unique socSerNr, or that a person can't have more mortgages then they can afford (the sum of all monthly payments can't exceed 30% of their salary). Previous to OCL these kinds of requirements would be included in textual descriptions without formal specification.

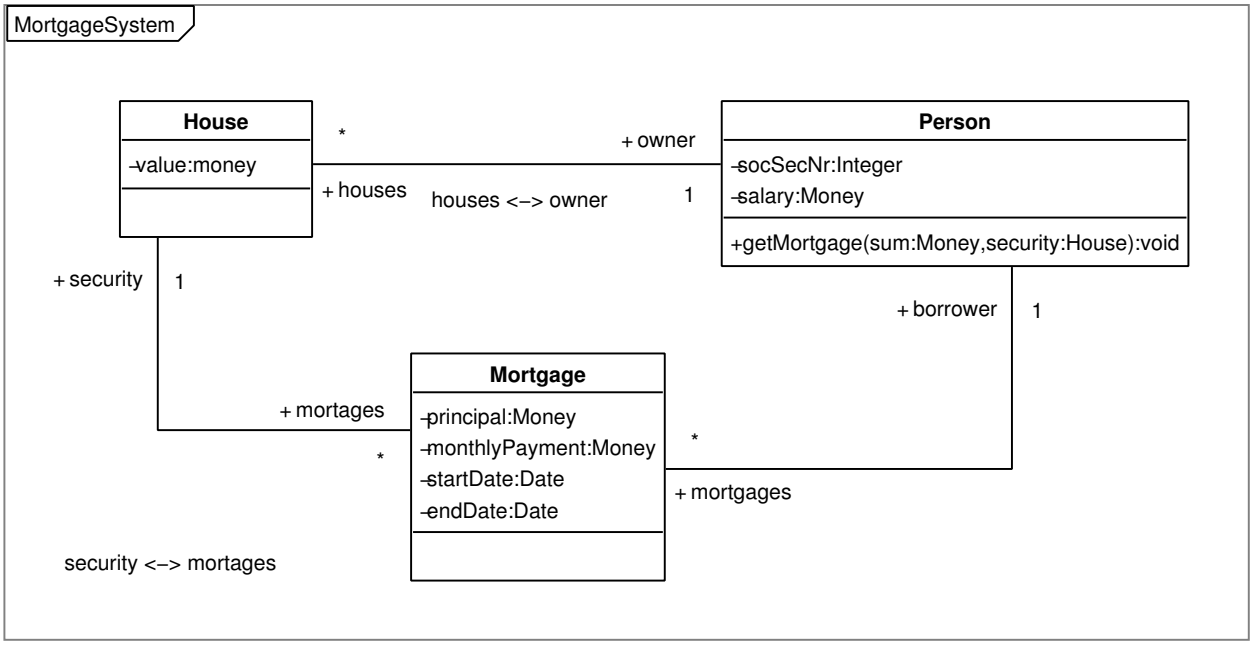

Fig. 2. Mortgage System UML Model

Listing 2. Mortgage System OCL example (taken from [8])

```
context Mortgage
inv: security.owner = borrower
context Mortgage
inv: startDate < endDate
context Person
inv: Person::allInstances()->isUnique(socSecNr)
context Person::getMortgage(sum : Money, security : House)
pre: self.mortgages.monthlyPayment->sum() <= self.salary * 0.30
context Person::getMortgage(sum : Money, security : House)
pre: security.value >= security.mortgages.principal->sum()
```

OCL expressions can be complex blocks of expressions involving control structures, loops, temporary variables, etc. It is a very full and powerful language that extends far beyond the simple examples included here.

Formalizing these constraints on the system helps the developer by clarifying the requirements and spelling them out in a fashion that can be automatically checked.

The benefits of OCL and UML go beyond their assistance as design tools, and their readability for developers. The major benefits of OCL and UML are achieved when using automated tools to leverage the upfront work in the design stage. Clearly fully developing and specifying a system using UML and OCL is a major undertaking and unless this extra cost in time and money can be balanced out in improvements and savings further on in the process it gains you nothing.

The more mature and fully developed your models are the easier to use and more effective these tools will be. The smaller the gap between model and code the less work will be required in migrating.

One chief benefit of using UML models is translating those models to code. Many good tools exist to handle this; perhaps the most widely known product is Rational Rose. Rational rose encapsulates all the UML modeling aspects from class diagrams to use case specification. When translating UML from it's abstract language independent form the tools use their built in knowledge of the target language to implement skeletons and stumps of all the elements in a structured project ready to be filled in and developed.

Adding OCL to a UML model improves this translation by using the OCL expressions to generate code for checking all the constraints and ensuring the code, once developed, will meet the specifications.

III. Pitfalls of OCL

Some shortcomings of OCL exist, and it is not a perfect technology for conceptual modeling. It can be argued that OCL is too focused upon implementation to make it an effective tool for conceptual modeling of a system [7]. Mandana Vaziri and Danial Jackson from the MIT Laboratory for Computer Science argue in [7] that the nature of OCL focuses the designer on implementation details when they should be concentrating on conceptual design problems.

Another issue raised is that since OCL allows complicated expressions with loops, etc. the use of OCL at run time can cause unforeseen problems that may be hard to diagnose if there is a mistake in the OCL expressions. This adds an extra set of problems for testers, debuggers, and developers.

One more issue with employing OCL in the development process is maintaining the specification in line with the model and code. Invariably in the development process design decision will be revisited and evaluated in the face of new information. If the conceptual model of the software changes it is important that all aspects of the system are in sync. This adds an extra level of complexity and responsibility into the process and can impact productivity and deadlines. However if the model is out of sync with the code its specifications will not be available for use in automated testing, etc.

IV. Tools for UML and OCL

The most well known and probably most employed is the Rational Unified Process [1], including Rational Rose the UML modeling tool. Rose is a very powerful and mature product, and is very capable but also complex. The Rational line of products are expensive commercial products produced by Rational Software, now owned by IBM.

ArgoUML[2] is an open source project that is a very capable UML design tool that integrates UML 1.3 (version 2 of the UML specification is now being formalized [3]) with tools for importing code into a model, as well as translating a model into code. It has modules for translation into several object oriented languages including Java, C++, PHP, and C#. It can also ”inject” OCL checking into the code, a process which creates runtime checkers of assertions and invariants by wrapping the class functions in its own set of functions which include the checkers. This OCL injection makes the code tricky to edit, so this injection should be done after the code is written (the ArgoUML editor allows code editing within the class diagrams.

Poseidon[3] is a commercial derivative of ArgoUML that adds additional functionality and improves usability. It has changed its architecture significantly since it started with the ArgoUML codebase, and no longer share the same UML model format. Being a commercial project it has the benefit of customer support when you buy a license. It has most of the functionality you would need from a UML development tool.

MagicDraw UML[4] is increasingly capable for UML development. This tool is approaching full UML 2.0 compliance. The commercial versions of the software include all the advanced features you expect from a Model Driven Architecture development environment. MagicDraw has full support for reverse engineering models from code in many languages, as well as framework generation from a model.

The open source development environment Eclipse[5], is a framework for developing many languages with plugins. Eclipse also incorporates UML modeling. One of the plug-ins for Eclipse is Octopus[6], a tool for OCL specification of UML models. Octopus is capable of generating a java framework from the UML and OCL.

V. Related projects

Other technologies exist that are related to OCL, one increasingly mature project is the Java Modeling Language (JML) [6]. JML differs from OCL in that it is embedded in the code itself, and uses Java syntax for its expressions. JML specifications are embedded in the header comments on classes and functions in Java. This allows a regular Java compiler to build the code without the JML specifications included to allow for faster running code. However when you use the JML version of the Java compiler it uses the JML specifications to include assertion and invariant checkers in the runtime code, and can assist in debugging and testing. At the 2004 ACM symposium Applied Computing a paper was presented that proposed a method to translate OCL specifications into JML [4]for Java

[1] http://www.ibm.com/software/rational
[2] http://argouml.tigris.org
[3] http:// www.gentleware.com
[4] http://www.magicdraw.com
[5] http://www.eclipse.org
[6] http://www.klasse.nl/english/research/octopus-intro.html

projects. Performing such a translation would make available the JML tools for validation and testing of Java software systems.

JML is a behavioral interface specification language, and therefore is used to specify contracts upon the behavior of elements of a projects interface. The main goal of JML is to create a method of specification that is applicable to production software environments, and therefore it leverages knowledge that developers may already have. It uses Java expressions in its assertions, and uses a style of specification from other sources such as Eiffel and VDM.

This is inherently a language specific way of handling specification and that narrows its applicability. This language specific nature is of course a boon if your code environment is mainly Java. Because it is tied to Java the gap between model and code is nearly non existent, and you do not have to worry about translation errors.

Alloy is a conceptual modeling tool, initially based on the Z notation [5]. Alloy conceptually captures the model of the classes, as well as the specifications upon it.

Alloy has its own set of tools for analyzing and leveraging the specifications to employ the specifications in testing and developing.

VI. Conclusion

UML and OCL are a powerful combination of tools for designing and developing software. Combining diagrams in the UML model including class diagrams to define the structure of the software to use case diagrams to exemplify typical behavior, with precise specification in OCL can make the transition from design phase to development phase nearly painless. The variety of commercial and open source tools available make this style of development available and applicable to nearly every object oriented software project. From a single developer personal project to a large scale development environment with many people collaborating there is a tool to fit the task.

Much work is going on in academia to improve upon these processes and utilize the specifications in OCL for many additional uses. Currently at the United Nations University, International Institute for Software Technology (UNU/IIST) researchers are working on an automated tool to generate test cases from OCL specifications [1]. Similar and related work is going on at many institutions; this continued work can only make implementing a UML and OCL design phase more valuable in the long term.

While transitioning to a full UML and OCL design process is a complicated and likely costly endeavor, its value can be seen in every phase of the development cycle. Of course it must be evaluated differently for every development team and project. There are many additional resources to be found to assist in this evaluation, and most of the software tools offer limited time evaluation licenses with full capabilities.

While UML with OCL is just one way to design and specify object oriented software systems, it is arguably the most mature and widely supported process. For projects that are highly language specific, and platform dependant, specialized tools and environments such as JML for Java may be better fit into the development cycle. However leveraging the implementation independent nature of UML can make transitioning to a different programming environment easier and less costly.

Choosing a process for software design is nearly as important as choosing the development language and tools. An easy to use, but powerful, system for designing and specifying software can provide great value to a project. The reusability of the specification for various things such as testing and code generation can offset much of the front end work involved in implementing a full specification. Whether you use all the features of a design tool, or just a small fraction of the capabilities it can assist in making software development easier.

References

[1] P. Antonio, P. Salas, and B. Aichernig. Automatic test case generation for ocl: a mutation approach. Technical report, United Nations University, International Institute for Software Technology, May 2005.

[2] O. M. Group. Introduction to omg's unified modeling language (uml). Technical report, Object Management Group, 2005.

[3] O. M. Group. Unified modeling language: Superstructure. http://www.omg.org/docs/formal/05-07-04.pdf, August 2005.

[4] A. Hamie. Translating the object constraint language into the java modelling language. In *SAC '04: Proceedings of the 2004 ACM symposium on Applied computing*, pages 1531–1535, New York, NY, USA, 2004. ACM Press.

[5] D. Jackson. Alloy: A lightweight object modelling notation. Technical report, Laboratory for Computer Science, Massachusetts Institute of Technology, 2001.

[6] G. Leavens, A. Baker, and C. Ruby. Preliminary design of jml: A behavioral interface specification language for java. http://www.cs.iastate.edu/ leavens/JML/prelimdesign/prelimdesign_toc.html, 2005.

[7] M. Vaziri and D. Jackson. Some shortcomings of ocl, the object constraint language of uml. Technical report, MIT Labratory for Computer Science, December 1999.

[8] J. Warmer and A. Kleppe. *The Object Constraint Language: Second Edition*. Object Technology Series. Addison Wesley, 2 edition, August 2003.