Comparison of the Modeling Languages Alloy and UML

Yujing He
Department of Computer Science
Portland State University
Portland, Oregon, USA

Abstract - Alloy is a new modeling language for software design, while Unified Modeling Language (UML) is a standard modeling language widely used in industry. This paper analyzes the similarities and the differences between Alloy and UML. It focuses on the complexity differences, accuracy differences, and the expression differences between these two languages. Both Alloy and UML can be used to specify the requirements to design complex software systems. The syntax of Alloy is largely compatible with UML. UML is more complicated, while Alloy is more concise. UML is more ambiguous, while Alloy is more accurate. UML is more expressive, while Alloy is more abstract. It is promising that part of UML can be formalized and transmitted to Alloy to allow automatic model validation analyzing in order to reduce errors in the requirement and design stages.

Keywords: Alloy UML OCL Model Comparison

1.0 Introduction

In the software development, the errors introduced in the early stages such as requirement and design stages normally will cost more to fix than errors found in the later stages. Therefore, the validation of the design stage is emphasized more and more by software developers. Nowadays, the standardized modeling notation for expressing object-oriented models and designs is UML (Unified Modeling Language). UML is widely accepted by industry. However, UML models don't have a formal semantics, and the constraints of UML models have to be expressed using OCL (Object Constraints Language). Alloy [1] is a new object-oriented modeling notation that appears in recent years, which is more concise and more precise, and furthermore, there is an analyzer tool to validate the models expressed in Alloy.

This paper compares several aspects of UML and Alloy. It is structured as follows: Section 2 describes the features of UML and Alloy; Section 3 compares the differences of Alloy and UML/OCL from complexity, accuracy and expression aspects. Finally Section 4 presents conclusions.

2.0 UML and Alloy

2.1 UML

The UML is used as a standard way to present a system's blueprints. UML uses diagrams to describe the models. The graphical notation of UML has no counterpart in textual style, and OCL is used to specify the constraints of the model in text. There are three categories of diagrams in

UML, including Structure Diagrams, Behavior diagrams, and Model Management Diagrams. Structure Diagrams represent static application structure; Behavior Diagrams represent dynamic behavior; and Model Management Diagrams represent ways to organize and manage the application modules. Advanced UML features include Object Constraint Language (OCL), and the Action Semantics Extensions. OCL can be used to specify restrictions such as invariants, preconditions, postconditions, etc. Action Semantics Extensions can be used to express actions in UML as action objects that have inputs and outputs. [3]

2.2 Alloy

Alloy is a structural modeling language that has been developed by the Software Design Group at MIT since 1997.Alloy is based on first-order logic to express complex structural constraints and behavior. It is designed to precisely describe the specification and the modeling of the system [1]. Alloy was created by making use of the core package of the UML meta-model and the well-formed constraints of UML. The subset selected by Alloy from UML is consistent [10]. Alloy is a little language with only a few modeling notations, which is easy to read and write. The grammar and the syntax of Alloy are also easy to grasp. Generally, the alloy model description includes several paragraphs: domain, state, def (definition), inv (invariants), cond (condition), assert, and op (operation). The simple structure of the Alloy model is as follows:

```
model M{
  domain{   }
  state{    }
  def D{   }
  inv I {    }
  op O( ) {   }
  assert A {   }
}
```

The '*domain*' declares sets representing the classification of atoms in the system. Each domain can be viewed as a primitive type. The '*state*' declares sets of states of the objects in the system. The '*definition*' declares the objects in terms of other objects. The '*invariants*' gives the invariant conditions or relationships within the system. The '*condition*' states some situations that do not always hold, which is different from invariants. The '*assertion*' will be checked to see whether they follow the invariants or not. And the '*operation*' states the operations of the objects in the system [2].

Alloy is a stand-alone language. The alloy model must be built textually. After building the alloy model by the analyzer, the model can be represented by the graphical part and the textual part. The graphical part represents the 'domain' and the 'state' of the textual part [2].

3.0 Comparison of Alloy and OCL/UML

Both Alloy and UML can be used to specify the requirements to design complex software systems. They can describe the states of a system and the transitions between states. The syntax of Alloy is largely compatible with OCL/UML, and both Alloy and OCL have formal syntax and semantics. OCL is used to specify constraints in UML model. OCL can specify invariants, preconditions, postconditons, and transitions of states in UML models. Alloy is similar to OCL. The syntax and semantics of Alloy is simpler. Alloy is fully declarative, while OCL is both declarative and operational. Here we will focus on the differences between UML and Alloy.

3.1 Complexity Differences

The description in OCL is more object-oriented, and its type system is also close to that of object-oriented language. The style of the Alloy language expressions is more declarative, so it can specify the computation in terms of the input data without a step-by-step sequence of commands.

Though the notations of OCL is similar to Alloy, it is more complicated to be used because it is applied in the context that include subclass, parametric polymorphism, operator overloading, multiple inheritance, etc. OCL is implemented oriented. Alloy is conceptual oriented. The syntax, semantics and grammar of OCL make the description in OCL more verbose than Alloy [11].

The structure of Alloy models will be built only from atoms and relations. Relations are used to relate atoms. They are first-order, and they are typed. A relation can be empty, unary, binary, ternary, etc. Alloy only considers finite relations. In Alloy, there are no sets or scalars. Sets are represented with unary relations. Scalars are represented with singleton unary relations. Every expression denotes a relation. This allows the Alloy model to be more succinct. A function is just a binary relation that maps the atoms on the left side to at most one item on the right side. Relations can be used to model different structures, such as Containment, Labeling, Grouping, and Linking. There are also three relational constants in Alloy: the empty relation, the universal relation, and the identity relation. Expressions in Alloy are constructed by nesting operators to variable. All expressions are relations, and every operator takes one or more operators and come out with also a relation. For the operators on sets (e.g. union, intersection, difference), the order of the elements within the tuple structure of a relation is not important. For operators indicating relations (e.g. join, transpose, reflexive, etc.), the order of the elements in a relation tuple matters [2].

The expressions of OCL are sometimes lengthy and hard to read. In OCL, the structure of the expressions with quantifiers, logical and collection operators are sometimes hard to figure out. The following lists some semantic and syntactic differences between OCL and Alloy [2,4,10,11]:

(1) In OCL, the set operators cannot be applied to the objects of the same class. In this case, quantifiers have to be used. The type system of OCL requires type compatible that is attained by using type coercion (oclIsKindOf). In Alloy, types are implicit and are associated with domains in the model [10].

(2) Alloy does not have such notions as field, method or integer arithmetic, etc. UML includes more notions and more types than Alloy. OCL supports a plenty of types, including basic types (e.g. Integer, Boolean, String, Real), user-defined model types (e.g. classes or interfaces), and object collection types (e.g. sets, sequences, bags). Alloy dispenses with the notions of tuples, sets, and so on. It has no built-in notion of composites at all. Composite types will be treated as atoms.

(3) OCL distinguishes the navigation of sets and scalars. For sets, it uses '->' to navigate, while for scalars, it uses '.' symbol. This adds complexity to explain and check its semantics. Alloy treats sets and scalars uniformly since scalars are regarded as singleton sets. The key operator of Alloy 'navigational dot' of Syntropy can be used in a more uniform and flexible way than in UML. A uniform syntax to 'navigation' can be used to both of sets and scalars, and Alloy uses the symbol '.' for the navigation with both sets and scalars in the expressions. The symbol '->' is used to represent product or join in Alloy.

(4) Alloy regards attributes and relations as the same, while UML uses different syntax or different semantics for attributes and relations [2]. Operators in Alloy can be applied to entire set and relations. Alloy models can handle relations with arbitrary arity, and it allows reuse of model fragments by structuring mechanism. Alloy's indexed relations are represented using qualifiers in UML.

(5) The classification in UML is static by default. A textual annotation will be needed to allow a classification to be dynamic. To express disjointness and exhaustiveness in Alloy, no textual annotations are required. The set in Alloy has at most one superset, while UML can have multiple inheritances. In UML, exhaustiveness is represented as a property of the superset [10].

3.2 Accuracy Differences

OCL has been part of UML since UML version 1.3. As part of the UML 2.0 standardization process, the revised version 1.6 of the OCL 2.0 submission was approved by the Object Management Group (OMG) in March 2003 and has been made available to the public. This version is more complete than the previous one in syntax and semantics [13]. This is an improvement of OCL, it provides extensive semantic descriptions based both on metamodel and formal mathematic model, but these semantics are currently still neither consistent nor complete enough [12].

OCL uses operations to describe the constraints. The operations arouse some problems in OCL. For example, an operation may go into infinite loop or it may be undefined, and this makes OCL less precise. Another problem is that the operations applied to several classes that have inheritance relationship, and then such operations may be refined by the objects of those classes. Therefore, the meaning of the expression containing the operations may be ambiguous to decide.

Most problems can be described by using only a small percentage of UML diagrams. And UML is not precise so that it is difficult to build tools to validate the systems described in UML. The semantics of OCL are still unsettled to be stable enough for tool to be developed upon it, while Alloy has a more rigorous semantic foundation.

Alloy is a simpler modeling language compared with OCL. It is more precise, and it is more tractable which allows automatic analyzing. Alloy is designed for automatic analysis. It can describe a system more precisely than UML. Alloy can be supported and analyzed by an associated tool 'Alcoa' [5]. The alloy analyzer can check the consistency in the model, and can generate snapshots for it [1]. Alloy supports high-level abstract modeling, but it is not meant to modeling code architecture as the class diagrams in UML.

3.3 Expression Differences

UML is generally more expressive than Alloy. It has more data types than Alloy. And UML has much more ways to describe system architecture, problem domain modeling, behavior capturing, etc. However, the expressions to specify relation are quite powerful in Alloy. For example, Alloy has transitive closure operator, but there is no such an operator in UML.

Here we will give a simple example of the STUDENT model to illustrate Alloy and UML separately. The meaning of the 'Students' model is straightforward. There are undergraduate students and graduate students, no student is both undergraduate and graduate student; a student should register, and only registered students are legal students; every student has a unique student ID, and he or she has a major and only can has one major for study; students with the same major are regarded as classmates, students can have several classmates.

3.3.1 Graph Differences

The above STUDENT model can be represented by both Alloy and UML using graphs as in Fig-1 and Fig-2.

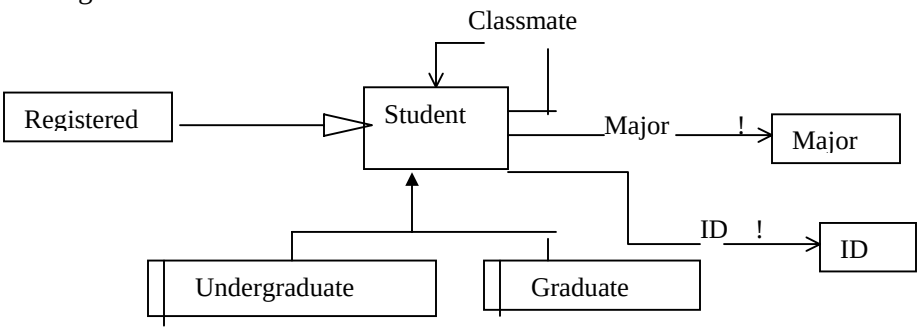

Fig 1. Alloy model of students

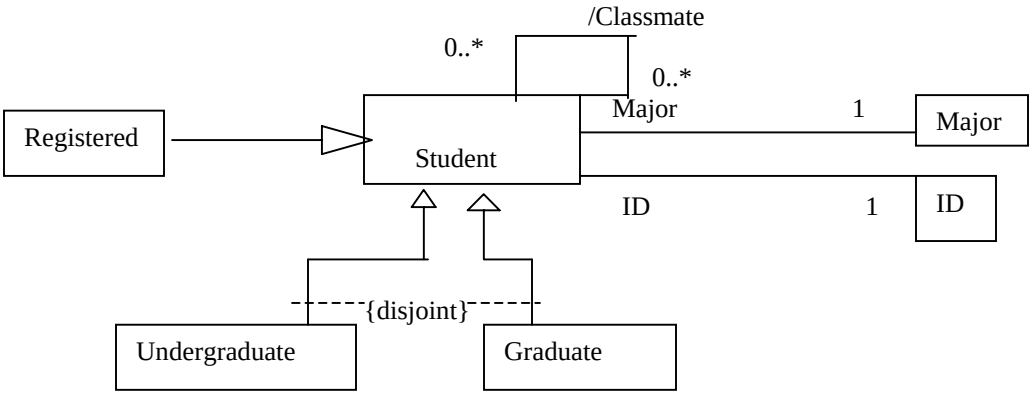

Fig 2. UML model of students

From the graphs in Fig-1 and Fig-2, we can see that Alloy graph is simpler and more abstract, while UML graph is verbose but easy to understand. When a model becomes more complex, the simplicity of Alloy graph will be more evident. Some differences between the graphic representations of Alloy and UML models are listed as follows:

(1) Alloy uses boxes to denote a set of objects, the same as UML. But Alloy separates some types of boxes such as static set and fixed set. Static Set: with a vertical line on the left of the box. Fixed Set: with a vertical line on the right of the box.

(2) Arrows are used to connect entities and their attributes in Alloy, while UML uses lines. In Alloy, an arrow with a closed head denotes subset, the same as UML. But in Alloy, subsets that share the same arrow are disjoint, if the subsets are exclusive, then the arrowhead should be filled. In UML, if subsets are disjoint, a symbol '{disjoint}' can be used to indicate this. In Alloy, the hatch mark behind the arrowhead means the relation is static. While in UML, a symbol '{frozen}' will be used to indicate the static relation [11].

(3) Alloy uses regular expression operators to represent multiplicity instead of using integer ranges as in UML. In Alloy, the markings at the end of the arrows indicate multiplicity constraints. The symbol '!' means exactly one; '?' means zero or one; '*' means zero or more; and '+' means one or more. Omission of the markings means the same as '*'. In UML, multiplicity is indicated by using integers, such as '1' means exactly one. '0..1' means zero or one. '0..*' means zero or more. '1..*' means one or more.

3.3.2 Textual Differences

(1) Alloy

The graph of Alloy reflects the 'domain' and the 'state' parts of the model. The text of Alloy model alone can represent the equivalent model. It includes the description of the graph and the description of the constraints. The textual part of Alloy model of Fig-1 is as follows. With the description of the model specified at the beginning of this section and the graph in Fig-1, the corresponding meaning of the expressions in this Alloy Model is quite easy to understand, since the textual notions map to the graph and the description of the model straightforwardly.

---

model Students{
  domain { Student, Major, ID}
  state{

```
        partition Undergraduate, Graduate: static Student
        Registered: Student
        Classmate: Student -> static Student
        Major: Student -> Major !
        ID: Student -> ID !
      }
      def Classmate{
        all a, b | a in b.Classmate <-> (a.major = b.major)
      }
     inv Basics{
        all s | s in Registered
        no s | s in Undergraduate && s in Graduate
    }
     op Register(s: Student, m: Major, d: ID){
          s not in Registered
          all s| s.Major' = s.Major
          all s| s.ID' = s.ID
          all s| s.Classmate' = s. Classmate
          Student' = Student
     }
     assert LegalStudent{
          all s: Registered
     }
}
```

---

(2) UML

UML diagrams and OCL combined represent models. Though UML graph doesn't have equivalent textual counterpart, it uses OCL to describe the constraints. The OCL for the graph Fig-2 is described as follows. The mapping of the textual OCL to the diagram is not straightforward, and the expressions are lengthy and more difficult to understand than those in Alloy.

Student

```
self.Classmate = Student.allInstances->select(Major=self.Major)
self.oclIsKindOf(Registered) implies self.ID. notEmpty
self.oclIsKindOf(Registered) implies self.Major.notEmpty
self.oclIsKindOf(Undergraduate) implies not self.oclIsKindOf(Graduate)
self.oclIsKindOf(Graduate) implies not self.oclIsKindOf(Undergraduate)
```

Student::Registered (m: Major)

```
pre: not self.oclIsKindOf(Registered)
post: self.Major=m
      Student.alInstances -> forall (s | s!=self implies s.Major@pre = s.Major )
      Student.alInstances -> forall (s | s!=self implies s.ID@pre = s.ID )
       Student alInstances -> forall (s | s.Classmate@pre = s.Classmate )
```

5.0 Conclusion

By comparing the features and the differences between UML and the new modeling language Alloy, we can see that some of the advantages of Alloy can be referenced as the improvement of the widely used language UML. UML is more complicated, while Alloy is more concise. UML is more ambiguous, while Alloy is more accurate. UML is more expressive, while Alloy is more abstract. As a result, Alloy provides a way to validate the design stage by using supported tools Alcoa [14]. It is promising that part of UML can be formalized and transmitted to Alloy to allow automatic model validation analyzing in order to reduce errors in the requirement and design stages.

6.0 Acknowledgement

I appreciate the unreserved encouragements from my father Wenfa He, my mother Mandai He, and my sister Yufei He. I also want to thank Professor Dick Hamlet for his instruction and comments on this paper, and thank Luc Vannier for his advices and suggestions. Thanks for the support from the CS department and the CSCE College of Portland State University.

7.0 References

[1] The Alloy Analyzer-3.0 Beta; 2005; *http://alloy.mit.edu/index.php*

[2] Daniel Jackson; Alloy: A Lightweight Object Modeling Notation; *ACM Transactions on Software Engineering and Methodology*, Vol.11, No.2, Pages 256-290; April 2002.

[3] Introduction To OMG's Unified Modeling Language ™ (UML); 2005; *http://www.omg.org/gettingstarted/what_is_uml.htm#12DiagramTypes*

[4] Geri Georg, James, Bieman, Robert France; Using Alloy and UML/OCL to Specify Run-Time Configuration Management: A Case Study; *Practical UML-Based Rigorous Development Methods-Countering or Integrating the eXtremists, Workshop of the pUML-Group held together with the UML2001*, October, 2001.

[5] Daniel Jackson, Lan Schechter and Ilya Shlyakhter; Alcoa: The Alloy Constraint Analyzer; *Proceedings of the 22nd International Conference on Software Engineering*, pages 730-733, Limerick, Ireland, 2000.

[6] Tiago Massoni, Rohit Gheyi, and Paulo Borba; A UML Class Diagram Analyzer; *The 3rd International Workshop on Critical Systems Development with UML*, Lisbon, Portugal, October 2004.

[7] UML2Alloy; 2005; *http://www.cs.bham.ac.uk/~bxb/UML2Alloy.html#Example*

[8] Wuwei Shen, Kevin Compton, James Huggins; A Toolset for Supporting UML Static and Dynamic Model Checking; *The 26th International Computer Software and Application Conference (COMPSAC 2002)*, Oxford, England, August, 2002.

[9] Introduction to OCL & USE; 2005; http://guinness.cs.stevens-tech.edu/~naumann/cs643/11-DN-6up.pdf

[10] Mandana Vaziri, Daniel Jackson; Some Shortcomings of OCL, the Object Constraint Language of UML; 1999; *http://www.research.ibm.com/people/m/mvaziri/papers/omg.pdf*

[11] Daniel Jackson; A Comparison of Object Modelling Notations: Alloy, UML and Z; *UML Workshop*: Mannheim, Germany, 1997.

[12] Stephen Flake; Towards the Completion of the Formal Semantics of OCL 2.0; *Conferences in Research and Practice in Information Technology*, Vol. 26. V. Estivill-Castro, Ed. 2004.

[13] OCL 2.0 Submission; 2005; *http://www.klasse.nl/ocl/ocl-subm.html*

[14] Daniel Jackson, Ian Schechter, and Ilya Shlyakhter; Alcoa: The Alloy Constraint Analyzer, *ICSE* Limerick Ireland, 2000.