A Validation Method for a UML Model Based on Abstract State Machines

Wuwei Shen[1]*, Kevin Compton[1], and James Huggins[3]

[1] EECS Department, University of Michigan
1301 Beal Avenue, Ann Arbor, MI 48109-2122 USA
wwshen, kjc@eecs.umich.edu
[2] Computer Science Program, Kettering University
1700 W. Third Avenue, Flint, MI 48504-4898 USA
jhuggins@kettering.edu

Extended Abstract

The Unified Modeling Language (UML) has been widely accepted as a standard for modeling software systems. There are many CASE tools which support UML. These tools provide features such as automatic code generation and reverse engineering of existing software systems. Because the UML provides a language for expressing requirements and modeling the activities of a system, we argue that the mistakes in design should be found at the early stage so can be easily corrected, saving time and money. But only a small number of tools are now used to support verification and validation of a UML model.

Generally speaking, a correct UML model should satisfy static and dynamic requirements. Verification of the dynamic part of a model is usually done by an existing verification tool, such as the model checker or prototype verification system (PVS). One such tool is due to Lillius and Paltor [8]. They give a semantic model for the state machine in UML by using structural operational semantics and they translate the dynamic model into the input language PROMELA for the model checker SPIN as a verification model. However, their tool separates the semantics and the verification model.

On the other hand, to reflect the static aspect of a model, UML offers several kinds of diagrams. These diagrams may be regarded as a language. It turns out that diagram-based UML is limited in its expressiveness, especially for describing certain important constraints. For this reason, the Object Constraint Language (OCL) was designed for the use in the context of diagrammatic specification languages like UML. OCL is a textual specification language, deeply connected to UML diagrams.

Unfortunately few UML CASE tools support OCL. One of the obvious reasons is that OCL and UML lack a precise semantics. One of the few CASE tools is USE [9], which can validate a static UML model and its constraints. USE translates OCL constraints into a tree structure and then executes the tree to find out whether the constraints are satisfied. However the semantics for OCL in USE is given by denotational semantics, which is cumbersome to use.

Abstract State Machines (ASMs) [1] were invented by Gurevich ten years ago. Since then, they have been successfully used in specifying and verifying many software systems, e.g.,

* Partially supported by NSF grant CCR 97-32735.

providing semantics for several different styles of programming languages. For example, we successfully gave both the static and dynamic semantics for C [7].

In this paper we describe an ongoing project at the University of Michigan to build a validation and verification tool for the UML. This is the first tool (to our knowledge) that can verify the both static and dynamic aspects of a model under the same framework. The architecture for this tool can be shown as follows:

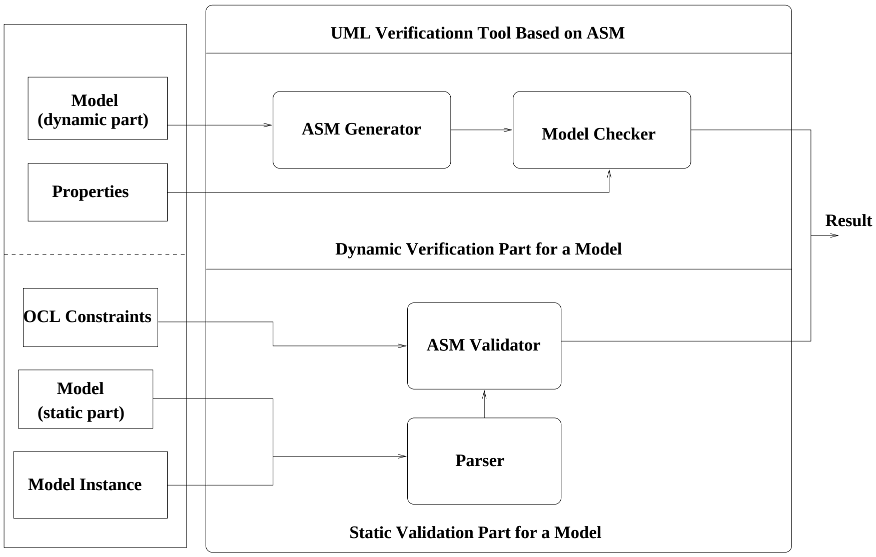


Fig. 1. A Structure for a UML Validation Tool

In the dynamic verification part, we translate the dynamic part of a UML model into an ASM specification, based on the schema given in [4] and [5]. Then we translate the ASM specification into the SMV input language. We deploy the Model Checker SMV to check whether a dynamic part of a model satisfies properties. More details about this part can be found in [6].

The architecture of the static validation part consists of two modules. One is the parser, used to take a model and its instance and generate a snapshot of a model. The parser is called by the ASM validator. A task of the ASM validator is divided into two parts. One is to give an ASM model for OCL. (I.e., it accepts the constraints written in OCL and gives the corresponding ASM rule.) The other part is to call the parser to get the model and its instance to evaluate the constraints.

To verify the static part of a model, we take the static part of a model (such as a class diagram), some OCL constraints associated with the model, and an instance of the model, like an object diagram. The main idea for this part is to compare the instances of a model with the specified model. For example, we compare an object diagram with its related class diagram. Combined with the constraints given in that class diagram, we check whether the constraints are satisfied in the object diagram. If the constraints are not satisfied, then the design should be changed, by either relaxing the constraints to include the instance of the model or changing the model to delete some undesirable instance of the model.

In the following, we give a simple example to see how the static part of the tool works. Figure 2 shows a UML class diagram of a simple example. Both a department and project have attributes `budget` showing available budgets and a department controls project on which any number of employees can work. Additionally, there is a constraint which can not be expressed in the diagram. The constraint is taken by the validator which evaluates its value to find out whether the constraint is satisfied in the instance of the model. The constraint written in OCL is shown below the class diagram in Figure 2.

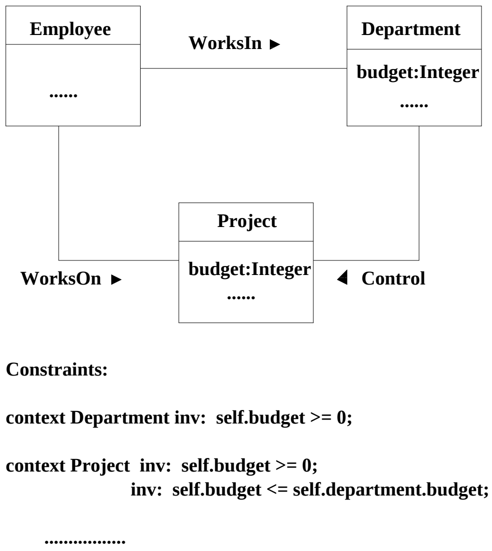
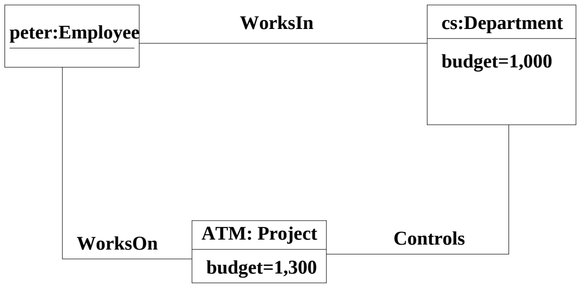


Fig. 2. A class diagram for a department example.

Figure 3 shows an instance of the above model. The parser takes the class and object diagrams as an input and the parser is called by the ASM validator to evaluate a constraint. Obviously this instance shown in the object diagram violates the model given in the class diagram and this error is reported by the validator. The above example gives a flavor of how the tool works for a static UML model.

The static and dynamic parts in developing a software system are two different processes. One can use the static part of the tool first and then the dynamic part, or vice versa; or even just use one of them. This completely depends on a user's requirement. The other benefit of the tool is that we can validate the UML metamodel. It can be applied to the Core package of the UML 1.3 metamodel and its well-formedness rules to check whether a UML model is valid or not. Combined with other UML CASE tools, this tool based on ASM will play an important role in developing some complicated software systems.



Object Diagram for the Department Example

Fig. 3. An object diagram for the department example

References

1. Abstract State Machine Homepage: http://www.eecs.umich.edu/gasm.
2. Matthias Anlauff, XASM- An Extensible, Component-Based Abstract State Machines Language, Proceeding of Abstract State Machine Workshop, 2000.
3. Egon Börger, Why Use Evolving Algebras for Hardware and Software Engineering?, in M. Bartosek, J. Staudek, J. Wiedermann, eds., SOFSEM '95: Theory and Practice of Informatics, Springer Lecture Notes in Computer Science 1012, 1995, 236–271.
4. Egon Börger, A. Cavarra, and E. Riccobene, An ASM Semantics for UML Activity Diagrams, in Teodor Rus, ed., Algebraic Methodology and Software Technology, 8th International Conference, AMAST 2000, Iowa City, Iowa, USA, May 20-27, 2000, Proceedings, Springer LNCS 1816, 2000, 293–308.
5. K. Compton, Y. Gurevich, J. Huggins and W. Shen, An Automatic Verification Tool for UML, Technical Report CSE-TR-423-00, EECS dept., University of Michigan, May 2000.
6. K. Compton, Y. Gurevich, J. Huggins and W. Shen, A Semantic Model for the State Machine in the Unified Modeling Language, in G. Reggio, A. Knapp, B. Rumpe, B. Selic, R. Wieringa eds, Dynamic Behavior in UML Models: Semantic Questions, UML 2000 workshop proceeding, Ludwig-Maximlilians- Unversität München, Institut für Informatik, Bericht 0006, Oktober 2000.
7. James K. Huggins and Wuwei Shen, "The Static and Dynamic Semantics of C: Preliminary Version", Technical Report CPSC-1999-1, Computer Science Program, Kettering University, February 1999.
8. Johan Lilius and Iván Porres Paltor, vUML: a Tool for Verifying UML Models, TUCS Technical Report No. 272, May 1999.
9. Mark Richters and Martin Gogolla. Validating UML Models and OCL Constraints, UML 2000, Springer Lecture Notes in Computer Science4 1939, 2000, 265- 277.