ISSN 2047-3338

Using Spin Model Checker for Learning the Semantics of UML Models

Farhan M Al Obisat
Department of Mathematics and Information Technology, Tafila Technical University

***Abstract*– This study is concerned with the developing criteria for learning the semantics of UML models in an intelligent tutorial system. This is achieved by going through literature and studying the current approaches for checking the semantics of UML diagrams. This paper concerns with the ability of learning the semantics of UML models using formal methods, For this reason this research enhance an existing system for grading UML models called MUML. The result of this work is an enhanced environment for teaching and checking UML behavior using the means of formal methods. Evaluation results on diagram-based learning do its expectations compared to the traditional learning techniques of the students' assignments.**

***Index Terms*– UML Semantics, Learning, Formal Representation, Model Transformation, Spin and Hugo/RT**

I. INTRODUCTION

MODEL checking [4] is a formal method for analyzing and verifying hardware and software systems. It is used generally for formal verification as it takes an automaton based model of a system and temporal logic properties as input then explores the entire state space of the model to determine whether it violates the given properties or not. Model checkers returned a counter example to confirm the violation to the analyzer. Both desired and unwanted properties of the model can be logically formulated. Therefore, both the model and its properties are presented in an acceptable format to be used as an input to a specific model checker, which tests the model and gives a feedback and reports conditions such as unreachable states, deadlocks, and conditions where the properties are violated. Later on, the results obtained by the model checker can be used to improve and refine the model until it becomes free of errors.

Model checkers can be classified into two main types: Symbolic and explicit. Symbolic model checkers are mostly used to check the hardware where the states of the system are encoded. On the other hand explicit state model checkers are used typically to check the software systems. They also examine the states that are stored explicitly, each in turn.

Models which are designed using UML need to be transformed into a formal notation in order to be examined using model checkers. There are a lot of work and efforts that describe how to convert a UML model into a Promela specification. Promela is the input language of the SPIN model checker [4].

II. METHODOLOGY

Going through literature review could make a good insight about approaches that are used for verifying the formal transformation of UML models. Current verification systems and tools have been reviewed and some of them have been installed and tested, such as TABU approach [2]. In this study Hugo/RT is used to transform UML models for model checking and code generation, as UML models can contain active classes either with state machines or collaborations, then these active classes could be mapped into the system language of Spin model checker for performing formal verification [8].

To determine which model checker to be used for verifying the semantics of UML diagrams a review of literature was done in this stage. For verification purpose, the most used model checkers are SPIN model checker [5], SAL model checker which can support both bounded and symbolic model checking [9], and SMV model checker as well as its derived NuSMV [1].

Based on the study of the available formalization approaches and model checking systems, a list of possible systems that can be used for verifying UML models are selected. Some of these systems are installed and tested and based on that the reasonable systems are then selected. In this study Hugo/RT is selected for achieving the formalization process for the UML models, and Spin model checker is selected to check and verify the generated formal model.

A framework for checking and verifying the semantics of UML models is developed by following processes:

1. Identify steps required for verifying semantics of UML models and this is done by mapping UML model elements into the formal notation using the verification systems that are chosen, and then identify a technique for measuring the semantic correctness of UML models. This process is done by analyzing the output from the model checker output result, while in this study we use the TRAIL file of SPIN model checker that contains the result of semantics check.
2. Developing and building the framework; the framework for handling this method is to use Hugo/RT formal

transformation technique that transform student model into a temporal logic format to catch the properties and convert them again into Promela language file. Promela, which contains the elements and properties of the original UML model is used as an input for the Spin model checker, whereas a result of Spin execution is stored in a file which's called a Trail file.

3. Test and evaluate the proposed framework; the test of the framework is done by doing the conversion process manually by experts (it takes more time, especially if the state space of the model is huge). The evaluation is made by doing the conversion for some sample models, once by using the systems and the second time is manually by experts in order to compare the outcomes.

All these processes mentioned above are used for the evaluation process to guarantee the verification of UML model semantics. Figure 1 shows an example of an actual case for a question given to students via MUML [7]. The question contains obvious names for active actors or classes which would make it easy for student to pick them from the given text. Furthermore, a description for the system behavior is described in details. The example in Figure 1 is written in a way that makes it easy for students to identify the elements of the model. For example, it is clear that the answer should have at least one *ATM* class and one *Customer* class. It is also clear that the behavior of the system should include some operations based on some conditions, such as (*Enter code, Abort operation, Enter amount, Print receipt* and *Withdrawal*).

A. Semantics Checking

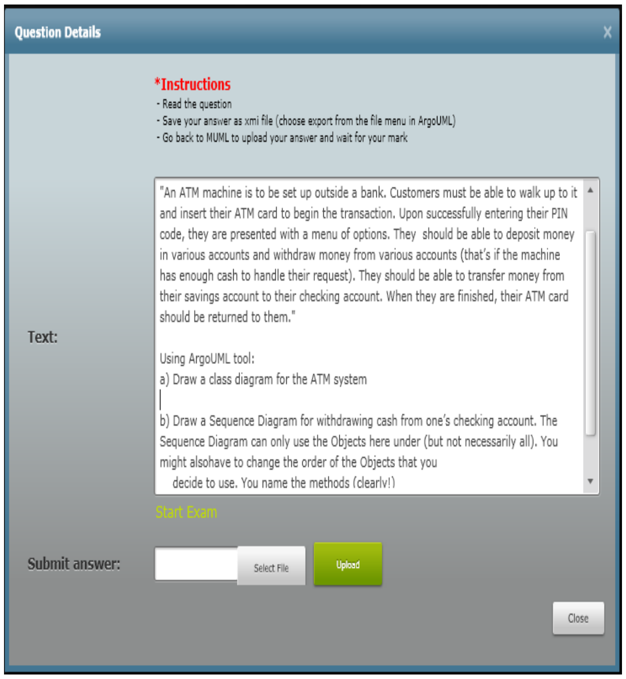
There are so many approaches for checking the semantics of UML model [6]. The approach in this research is slightly similar to the one proposed by [2] which is called TABU and also similar to STAIRS approach [10]. The similarity is that the semantics of the UML model is checked using model checking techniques. Approach in this work differs from TABU by using sequence diagrams instead of activity diagrams for representing the model behavior.

B. Formal Model Verification

Formal UML models are verified using model checking techniques. Many model checkers can be used for semantic verification. The Spin model checker is chosen for this study. The reason for this choice is that it specifically verifies software. For instance, the target of SMV model checker is the verification of hardware circuits, where the target of UPPAAL model checker is the verification of real-time systems.

Secondly, Spin is an open source and multi-platform, distributed for free as a research system [3]. Spin model checker has a user friendly graphical user interface that allows the user to write directly the system specifications using Promela the C like language. Promela is the input language for SPIN where it can be edited inside the system or imported from the file location. Figure 1 shows an example of Promela code written in jSpin which is a version of Spin model checker. The interface of jSpin is divided into three parts, one for showing and editing the code, and one part for the command line option and the third part for the output. The output is therefore saved in a TRAIL file which is empty if there are no execution errors. Figure 2 shows the interface of jSPIN model checker.

Since the input for Spin is Promela therefore, UML model needs to be translated to Promela. Hugo/RT is used in this work for mapping the UML model to Promela language. It is used to transform UML models for model checking, theorem proving, and code generation. Since UML models can contain active classes either with state machines, collaborations or with both, then these active classes can be mapped into the system language of Spin model checker and some other Theorem Proves [8].



*Instructions
- Read the question
- Save your answer as xmi file (choose export from the file menu in ArgoUML)
- Go back to MUML to upload your answer and wait for your mark

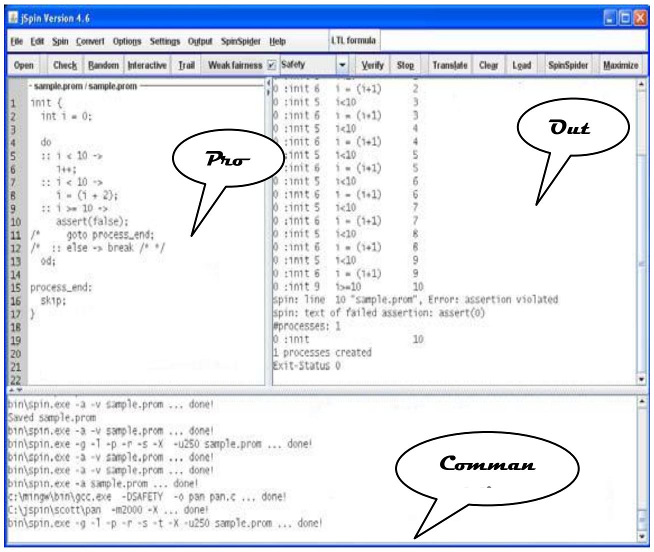
Text:

"An ATM machine is to be set up outside a bank. Customers must be able to walk up to it and insert their ATM card to begin the transaction. Upon successfully entering their PIN code, they are presented with a menu of options. They should be able to deposit money in various accounts and withdraw money from various accounts (that's if the machine has enough cash to handle their request). They should be able to transfer money from their savings account to their checking account. When they are finished, their ATM card should be returned to them."

Using ArgoUML tool:
a) Draw a class diagram for the ATM system

b) Draw a Sequence Diagram for withdrawing cash from one's checking account. The Sequence Diagram can only use the Objects here under (but not necessarily all). You might alsohave to change the order of the Objects that you
decide to use. You name the methods (clearly!)

Figure 1: An example of a Question Set by an Instructor


jSpin Version 4.6
File Edit Spin Convert Options Settings Output SpinSpider Help
LTL formula
Open Check Random Interactive Trail Weak fairness Safety Verify Stop Translate Clear Load SpinSpider Maximize
sample.prom / sample.prom
Pro
Out
Comman

```
init {
  int i = 0;

  do
  :: i < 10 ->
      i++;
  :: i < 10 ->
      i = (i + 2);
  :: i >= 10 ->
      assert(false);
/*      goto process_end;
/*   :: else -> break /* */
  od;

process_end:
  skip;
}
```

```
0 :init 6   i = (i+1)       2
0 :init 5   i<10            3
0 :init 6   i = (i+1)       3
0 :init 5   i<10            4
0 :init 6   i = (i+1)       4
0 :init 5   i<10            5
0 :init 6   i = (i+1)       5
0 :init 5   i<10            6
0 :init 6   i = (i+1)       6
0 :init 5   i<10            7
0 :init 6   i = (i+1)       7
0 :init 5   i<10            8
0 :init 6   i = (i+1)       8
0 :init 5   i<10            9
0 :init 6   i = (i+1)       9
0 :init 9   i>=10           10
spin: line  10 "sample.prom", Error: assertion violated
spin: text of failed assertion: assert(0)
#processes: 1
0 :init                     10
1 processes created
Exit-Status 0
```

```
bin\spin.exe -a -v sample.prom ... done!
Saved sample.prom
bin\spin.exe -a -v sample.prom ... done!
bin\spin.exe -g -l -p -r -s -X  -u250 sample.prom ... done!
bin\spin.exe -a -v sample.prom ... done!
bin\spin.exe -a -v sample.prom ... done!
bin\spin.exe -a sample.prom ... done!
c:\mingw\bin\gcc.exe  -DSAFETY  -o pan pan.c ... done!
C:\jspin\scott\pan  -m2000 -X ... done!
bin\spin.exe -g -l -p -r -s -t -X -u250 sample.prom ... done!
```

Figure 2: An Example of jSpin Model Checker GUI

C. Semantics Checking

This section describes the process for the semantics check. The input for this particular part of the learning systems is achieved by verifying some Linear Tree Logic properties (LTL). Instructors can insert these properties directly along with the given assignments, or by inserting them in the specific panel in jSpin editor. The learning system stores them in a file called *property.txt*.

For the example described in Figure 1, some properties could be chosen s to verify the semantics of the ATM machine student's model. Table 2 describes these properties and the correspondent specifications.

The semantics elements that are normally checked include:

- The relations between classes
- The states of each class object
- The activities of the model
- The sequences of model behavior or messaging
- The LTL properties given by instructors, if any

The task of checking the semantics of UML models is done by verifying the submitted answer and checking it against the properties given by instructors. The semantics check is to be performed based on the following algorithm shown in Figure 3 [7]:

Table 2: Formal properties and their representation in LTL

No	Property	Specification	Rechecking
1	The ATM cannot allow the user to request an operation if either the card or the PIN is invalid	_ (x → ¬ _ y)	*x = (start ∧ (¬ cardOK ∨ ¬ PINok))*
			y = (proc1 user∧ receive ∧ msg waitAccount)
2	ATM must first debit the amount in the bank, And then give the money to the user. In other words, the user does not receive pickCash until the bank receives debit	¬ x U y	*x = (proc1 user∧ receive∧ msg pickCash*
			y = (proc1 bank ∧ receive ∧ msg debit)
3	If the ATM receives insufficient funds, it should allow the user to choose other operation before finishing the session	_ (x → (¬ y U w))	*x = (proc1 user ∧ receive ∧ msg insufficientFunds*
			y = (end), and w = (proc1 atm ∧ send ∧ msg waitOperation)
4	Ensure the correct end of the session between the ATM and the user. It says that, after the user receives ejectCard, the ATM cannot send anything to the user	_(x → ¬_y)	*x = (proc1 user ∧ receive ∧ msg ejectCard)*
			y = (porc1 atm ∧ send ∧ proc2 user)

1. Convert the zargo file produced by the student using ArgoUML into Promela
2. Add TLT properties if necessary
3. Execute the Promela file and the properties using Spin model checker
4. Analyze the output TRAIL file from Spin execution
5. Count the number of errors and warnings
6. Multiply the sum of errors by one
7. Multiply the sum of warnings by half
8. Add the multiplication results of errors and warnings
9. If the result is less than or equals zero then the semantic checking is zero
10. Subtract the result from the checking assigned for semantic in the checking file and divide it by 100
11. Subtract the new result from the assigned checking
12. The result is the semantic checking for the given question

Figure 3: The Semantics Check Algorithm (adopted from [7])

The process of converting a UML file (*zargo* file) into Promela is to be done by using Hugo/RT. Hugo/RT is a message exchanging system for UML state machines. It generates an automaton representing a UML interaction for observing message traces.

The summary of the output from the Spin model checker for the four properties given by the instructor in Figure 1 is shown below in Table 3.

Table 3: Summary of Semantic checking and feedback

Properties	Output	Message produced	Feedback
1	ok	0 errors, 2 warnings	No errors regarding to the PIN and card validity
2	Produced TRAIL	5 errors	Please check the model for the sequence of getting cash and debit the balance.
3	ok	No errors	The ATM can allow the user to do anther operation if he has insufficient balance
4	Produced TRAIL	7 errors, 8 warnings	Seems to be some active secessions after the user gets back his card

From the result in Table 3, it is clear that a total of 12 errors and 10 warnings have occurred. The feedback will be saved in the *feedback file* and will be able to be presented to user to guide him/her to go back to the assignment and refine the model.

As a result, Spin model checker could execute Promela with an empty TRAIL file. That means the checking for semantics will be 100% free of errors and no feedback is generate. On the other hand SPIN will generate a TRAIL file that contains the model errors and violations.

III. CONCLUSION

The development of a framework for learning the semantics of UML models helps students as well as instructors to learn how to model with UML and how to focus on the behavior of systems in a learning environment. The paper also described how to benefit from the output of certain model checkers to determine to which degree students understand this critical part of the system design, since applying some properties to the model can't be checked manually.

The approach in this study make use of the above described software systems and combine all of them i.e., ArgoUML, Hugo/RT, Spin model checker under one interface for checking the relevancy, syntax and semantics of UML models submitted by UML course students and to obtain feedback for this particular part of the overall learning system. The paper also described the framework for checking the semantics of the given answer, by applying some formal rules. Based on these rules the checking for this part is obtained. The formal representation of the model answer allowed us to use the model checkers to be run on the transformed model. Finally, the paper described how to obtain the weakness of student's assignments for each part of the learning aspects and how to obtain the feedback for the given model.

REFERENCES

[1] Alessandro, M. Fernandez, M., Cittolin, Gand, R. 1999. *Manual on hatchery production of seabass and gilthead seabream. Vol 1. FAO*, Rome. 194p.

[2] Beato, M. Barrio, S. and Cuesta, C. 2004. UML automatic verification tool (TABU). *Proc. Specification & Verification of Component Based Systems (SAVCBS) 2004, SIGSOFT 2004/FSE-12 12th ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pp. 106-109.

[3] Daniel, A., and Guido, W. 2008. Prototyping visual interpreters and debuggers for domain-specific modelling languages. *In Ina Schieferdecker and Alan Hartman, editors, 4th European Conference on Model Driven Architecture - Foundations and Applications (ECMDA-FA'08), volume 5095 of Lecture Notes in Computer Science, pages 63–78. Springer-Verlag.*

[4] Gerard, J. 2002. Software analysis and model checking. *In Ed Brinksma and Kim Guldstrand Larsen, editors, CAV, volume 2404 of Lecture Notes in Computer Science, pages 1–16. Springer.*

[5] Gerard, J. 2003. *The Spin Model Checker, Primer and Reference Manual*. Addison-Wesley, Reading, Massachusetts.

[6] Harel, D. and Politi, M. 1997. Modeling Reactive Systems with Statecharts: The STATEM- ATE Approach. McGraw-Hill.

[7] Hazim, R. Sufian, I. Abdullah, Z. 2011. *The design and implementation of MUML,* Proceeding of the IETEC'11 conference, Kuala Lumpur, Malaysia, Copyright@H.Rawashdeh, et al, 2011.

[8] Knapp, A. and Merz, S. 2002. Model checking and code generation for UML state machines and collaborations. *in Proceedings of 5th Workshop on Tools for System Design and Verification (FM-TOOLS'02).*

[9] Leonardo, M., Sam, O., Harald, R., John, R., and Shankar N. 2004. The ICS decision procedures for embedded deduction. *In David Basin and Micha¨l e Rusinowitch, editors, 2nd International Joint Conference on Automated Reasoning (IJCAR), Volume 3097 of Springer-Verlag Lecture Notes in Computer Science*, pages 218–222, Cork, Ireland, July 2004.

[10] Seehusen, F., Solhaug, B., and Stølen, K. 2009. Adherence preserving refinement of trace-set properties in STAIRS: exemplified for information flow properties and policies. *Journal of Software and Systems Modeling, 8(1):45–65.*