A Framework for Specifying and Analyzing Temporal Properties of UML Class Models

Mustafa Al-Lail
Colorado State University
Computer Science Department
mustafa@cs.colostate.edu

Abstract. Software designers widely use UML Class Models to specify the static structure of object-oriented systems. Temporal properties of class models can be expressed using the TOCL, an extension of OCL with elements of a linear temporal logic. Specification and verification of temporal properties expressed in TOCL is non-trivial and no automated tools exist that aid such verification. Existing approaches rely on transforming the UML models to other languages that have automated analysis support. Such transformation is complex and can introduce errors. Towards this end, this paper proposes a framework for specifying and directly analyzing temporal properties expressed in TOCL. The framework was validated using two demonstration case studies and in both cases, the approach uncovered design faults.

Keywords: Analysis, Verification, Class Model, Temporal Properties

1 Problem and motivation

UML Class Models are probably the most common specification diagrams used in the software industry. Automated analysis of class models often uncovers design problems in a timely manner and therefore saves time and effort. Specifying and analyzing temporal properties of class models is non-trivial due to the following challenges. First, specifying temporal properties in formal temporal logic notations, such as LTL and CTL, can be challenging [1], specifically for most Model-Driven Engineering (MDE) practitioners. Second, existing approaches (e.g., see [2–10]) rely on transforming the UML behavioral models to another language that supports automated analysis. Such transformation is complex and can introduce errors due to the gaps in semantics between UML and the target languages. Third, given the complex state spaces and the dynamic nature of the allocation and deallocation of object-oriented systems, developing model-checking support for such systems is challenging [11].

This research proposal presents a framework that addresses the preceding challenges. In particular, the framework aims to provide the following results: (1) a class model analysis approach that is UML-oriented, (2) an object-oriented technique for specifying temporal properties, and (3) a development process with an automated tool. A successful development of such framework will yield results that can be utilized by MDE practitioners to develop reliable complex systems.

2 Background and related work

A number of model-checking-based techniques exist for specifying and analyzing temporal properties in UML behavioral models, such as statemachines and activity diagrams [2–10]. These techniques involve developing an exogenous transformation process. Typically, the UML behavioral models are transformed to languages that are supported by model checking tools. For example, the vUML [2] tool automatically transforms UML statemachines to PROMELA specifications. A LTL temporal property is specified in PROMELA language as well. The SPIN model checker is then invoked to verify the desired property.
Three main shortcomings are associated with these approaches. First, effective use of these model-checking techniques requires developers to have specialized skills that are not UML-related. Second, the correctness of the analysis results depends on the correctness of the transformation and whether it preserves the semantics of the source UML models. Third, the results of the analysis performed by the back-end analysis tool must be presented to developers in UML terms in order to be examined, thus requiring another transformation process.
The existing structural anlaysis tools of UML/OCL such as USE [12] and OCLE [13] provide little support for temporal analysis. Towards this end, researchers have demonstrated how scenarios can be statically modeled as a sequence of state transition, which in turn, can be verified using USE and OCLE [14]. However, adapting such an approach for verifying temporal properties is still an ongoing challenge. This research proposal aims to fill this gap.

3 Approach and uniqueness

3.1 The UML-based analysis approach

The research question that led to this approach is the following: *Given a UML design class model, and a temporal property, is there a scenario, which is supported by the class model, that violates the property?* A design class model specifies the set of all possible states of a system and includes operations contracts to specify the system behavior. A *scenario* is a sequence of state transition supported by the class model. A temporal property is specified in TOCL, which is a temporal logic extension of OCL [15]. Fig. 1 presents an overview of the approach. At the front-end of the approach, a system designer is responsible for 1) creating a design class model, and 2) specifying a temporal property in TOCL. Then, the system designer utilizes the USE Model Validator at the back-end to generate a number of behavioral scenarios against which the temporal property is checked. If any of the *scenarios* violates the TOCL property, the tool returns it as a counterexample. The back-end processing is transparent to the system designer. The approach consists of the following four major steps:

Step1: Unfolding of Application Design Class Model. This step takes a design class model as input and produces a transition-based class model of

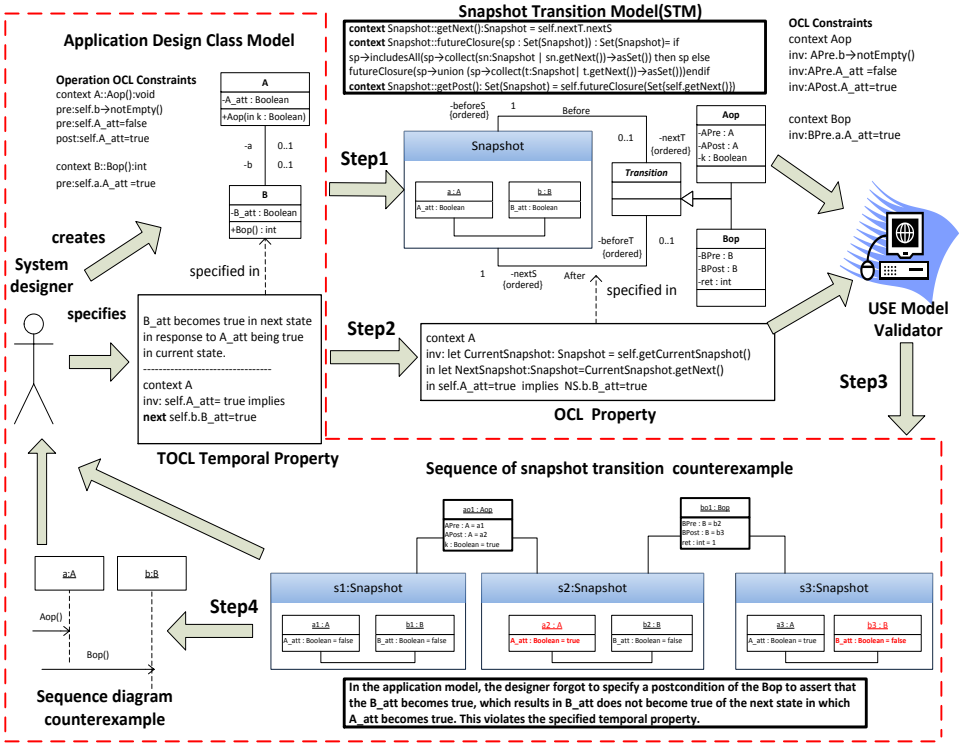

Fig. 1: An Overview of the analysis approach

behavior, which is called a *Snapshot Transition Model* (*STM*). The *STM* is a class model that characterizes the unfolding of the behavior of the design class model as valid sequences of state transitions caused by executions of operations. A state is called a *snapshot* and it is a structured class that represents a configuration of objects. The *STM* is formed by (1) creating a *Snapshot* class whose instances represent states in a transition system, (2) creating a hierarchy of transition classes that represents operation invocations, (3) converting the operations' contract conditions to invariants of the transition subclasses, and (4) defining traversal query operations between snapshots. The *STM* is mechanically generated from the design class model [14].

The approach utilizes the snapshots traversal operations to specify and analyze temporal properties in OCL. Specifically, the operation `getNext()` returns the next `snapshot` and the operation `getPost()` returns the set of all `snapshots` that come after a `snapshot`. The operations `getPrevious()` and `getPre()` are defined similarly.

Step2: Interpreting TOCL as OCL property. The unfolding of the transition system of the design class model results in linear traces (sequence of snap-

shots represented by the STM) that can be constrained by OCL. The approach thus interprets the TOCL property, specified in the class model, as OCL first-order constraint that is defined in the context of the SMT. Fig. 1 shows an example. The TOCL and OCL properties are instances of formal property specification patterns, discussed in Section 3.2.

Step 3: Analysis. The approach uses the USE Model Validator [16] to produce scenarios (e.g., instances of the STM) and check if any of them violates the OCL property generated in Step 2. The Model Validator uses boolean satisfiability (SAT) solver to perform the analysis. The tool generates a constrained number of scenarios, not all possible scenarios. The designer uses scopes to restrict the number of instances that each class can have and limit the number of transitions in a scenario. As such, the Model Validator enumerates all possible scenarios within the defined scopes and checks them against a given property. When no counterexample is found, the scopes can be increased to provide the system designer with higher confidence that the property holds on the model; but that does not guarantee that there is no counterexample in bigger scopes. Provided the right tool, the analysis could also be performed by more powerful solvers such as Satisfiability Modulo Theories (SMT) solvers.

Step 4: Extracting sequence diagrams. A big number of objects and transitions produces a counterexample that is complicated and difficult to examine. To make the analysis result more readable, an algorithm was developed to extract a sequence diagram from a scenario [17].

3.2 The temporal property specification technique

Many software designers find specifying temporal properties, in a formal notation, challenging [18]. The research question that led to this technique is: *How can we accommodate UML modelers who are unfamiliar with formal temporal language notations?* Dwyer et. al [18] designed a number of property specification patterns to aid in specifying temporal properties in different formal notations such as LTL and CTL. To address the above question, the patterns of Dwyer et al. [18] are defined in the two object-oriented notations, TOCL and OCL. A user of the technique determines the pattern that best fits the requirement and then uses the corresponding TOCL pattern to write the intended property. The OCL property is then systematically generated from the TOCL.
A total of eight patterns are proposed by Dwyer et al. [18] among which the response pattern is the most widely used in practice [1]. The response pattern captures the requirement that a state condition eventually holds in response to another condition. Pattern scopes specify the portion of the system execution in which a property must hold. Table 1 gives the TOCL and OCL patterns of the response pattern in two scopes. The expression $S \models P$ indicates that the property P holds in the snapshot S. In instances of the patterns, the approach generates the OCL condition that asserts that P is satisfied in S. Refer to Table 1 for an example.

Table 1: Response Pattern

Scope	TOCL Pattern on class model	OCL Pattern on the *STM*
Globally	**context** *[Class]* **inv:** *[P] implies sometime [S]*	**context** *[Class]* **inv:** *let CS: Snapshot = self.Snapshot* *in let FS: Set(Snapshot) = CS.getPost()* **in** *[P] implies FS → exists(s:Snapshot | [s ⊨ S])*
Example property	**TOCL instance**	**OCL instance**
B_att eventually becomes true in response to A_att being true in current state	**context** *A* **inv:** *self.A_att=true implies sometime self.b.B_att=true*	**context** *A* **inv:** *let CS: Snapshot = self.Snapshot* *in let FS: Set(Snapshot) = CS.getPost()* **in** *self.A_att=true implies FS → exists(s:Snapshot | s.b.B_att=true)*
After Q	**context** *[Class]* **inv:** *[Q] implies always ([P] implies sometime [S])*	**context** *[Class]* **inv:** *let CS: Snapshot = self.Snapshot* *in let FS: Set(Snapshot) = CS.getPost()* *in let PS: Snapshot= FS → select(s:Snapshot | [s ⊨ P])* *in SS: Snapshot= FS → select(s':Snapshot | [s' ⊨ S])* **in** *[Q] implies PS.getPost() → includes(SS)*

4 Conclusions

The main contribution of this research is a framework for specifying and analyzing temporal properties of the UML class models. The framework provides three results. First, the analysis approach does not require exogenous transformation to other languages, nor does it require that system designers to be familiar with notations other than UML and TOCL. Therefore, the analysis approach is totally UML-oriented. Second, software design patterns are good solutions to some software engineering problems. In the framework, the problem of specifying temporal properties is addressed by defining TOCL specification patterns. UML modelers can employ these TOCL patterns that represent a set of commonly occurring properties to correctly and concisely specify temporal properties in object-oriented notation. Third, the development of a tool that fully automates the procedures and the algorithms is still ongoing, although a large portion of the framework has been implemented.

This research is validated by applying it to the specification and verification of two demonstration case studies [19, 17]. The first case study is based on the Generalized Spatio-Temporal Role-Based Access Control Model(GSTRBAC) [20]. The second case study is based on the Steam Boiler Control System specification problem [21]. The results of the studies show that all the temporal properties of the two systems can be expressed by the property specification technique, and that the analysis approach is capable of uncovering errors.

Future work will concentrate on improving the framework and addressing its limitations. First, the approach is lightweight and only checks finite scenarios; therefore, unbounded liveness properties that require infinite scenarios can not be checked. Investigation will be performed on how the approach can be extended to support such properties. Second, the scalability and the efficiency of the property specification and analysis approaches will be investigated. Specifically, a slicing technique will be developed to verify properties on large class models.

References

1. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in property specifications for finite-state verification. In: ICSE. (1999) 411–420
2. Lilius, J., Porres, I., Paltor, I.P., Centre, T., Science, C.: vUML: a Tool for Verifying UML Models. (1999) 255–258
3. Eshuis, R.: Symbolic model checking of uml activity diagrams. ACM Trans. Softw. Eng. Methodol. **15** (January 2006) 1–38
4. Zhang, S.J., Liu, Y.: An Automatic Approach to Model Checking UML State Machines. In: SSIRI (Companion). (2010) 1–6
5. Schäfer, T., Knapp, A., Merz, S.: Model Checking UML State Machines and Collaborations. Electr. Notes Theor. Comput. Sci. **55**(3) (2001) 357–369
6. Shen, W., Low, W.L.: Using Abstract State Machines to Support UML Model Instantiation Checking. In: IASTED Conf. on Software Engineering. (2005) 100–105
7. Dubrovin, J., Junttila, T.A.: Symbolic Model Checking of Hierarchical UML State Machines. In: ACSD. (2008) 108–117
8. Raschke, A.: Translation of UML 2 Activity Diagrams into Finite State Machines for Model Checking. In: EUROMICRO-SEAA. (2009) 149–154
9. Niewiadomski, A., Penczek, W., Szreter, M.: A New Approach to Model Checking of UML State Machines. Fundam. Inform. **93**(1-3) (2009) 289–303
10. Xie, F., Levin, V., Browne, J.C.: Model checking for an executable subset of uml. In: ASE. (2001) 333–336
11. Distefano, D.: On Model Checking the Dynamics of Object-Based Software - a Foundational Approach. PhD thesis, University of Twente (2003)
12. Gogolla, M., Büttner, F., Richters, M.: USE: A UML-based Specification Environment for Validating UML and OCL. Sci. Comput. Program. **69**(1-3) (2007) 27–34
13. Chiorean, D., Paşca, M., Cârcu, A., Botiza, C., Moldovan, S.: Ensuring UML Models Consistency Using the OCL Environment. Electron. Notes Theor. Comput. Sci. **102** (November 2004) 99–110
14. Yu, L., France, R.B., Ray, I., Ghosh, S.: A Rigorous Approach to Uncovering Security Policy Violations in UML Designs. In: ICECCS. (2009) 126–135
15. Ziemann, P., Gogolla, M.: OCL Extended with Temporal Logic. In: Ershov Memorial Conference. (2003) 351–357
16. Soeken, M., Wille, R., Kuhlmann, M., Gogolla, M., Drechsler, R.: Verifying uml/ocl models using boolean satisfiability. In: MBMV. (2010) 57–66
17. Al-Lail, M., Abdunabi, R., France, R., Ray, I.: An Approach to Analyzing Temporal Properties in UML Class Models. In: Submitted to MODEVVA workshop. (2013)
18. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Property specification patterns for finite-state verification. In: FMSP. (1998) 7–15
19. Al-Lail, M., Abdunabi, R., France, R., Ray, I.: Rigorous Analysis of Temporal Access Control Properties in Mobile Systems. In: ICECCS. (July 2013)
20. Ramadan Abdunabi, Mustafa Al-Lail, Indrakshi Ray, Robert France: Specification, Validation, and Enforcement of a Generalized Spatio-Temporal Role-Based Access Control Model. IEEE Systems Journal (2013)
21. Abrial, J.R., Börger, E., Langmaack, H.: The Stream Boiler Case Study: Competition of Formal Program Specification and Development Methods. In: Formal Methods for Industrial Applications. (1995) 1–12