International Journal Of Computer Science And Applications Vol. 1, No. 1, June 2008
ISSN 0974-1003

Verification Framework for Detecting Safety Violations in UML State chart Models of Reactive Systems

C.M. Prashanth
Dept. of Computer Engineering
N.I.T.K, Surathkal
INDIA-575 025
+91 9448185670
prashanth_bcs@yahoo.co.in

Dr. K. Chandrashekar Shet
Dept. of Computer Engineering
N.I.T.K., Surathkal
INDIA-575 025
+91 9845237101
kcshet@nitk.ac.in

Janees Elamkulam
IBM India Ltd.
Airport Road, Bangalore
INDIA-560 017
+91 9448185670
Janees.ek@in.ibm.com

ABSTRACT
The model based development is a widely accepted phenomenon to build dependable software. This has lead to development of tools which can generate deployable code from the model. Hence, ensuring the correctness of such models becomes extremely important. Model checking technique can be applied to detect specification violations in such models at the early stage of development life cycle. In practice, such validations are done using off-the-shelf model checkers. This technique though popular has a drawback that, model should be described in the native language of the model checker. In this paper, we propose a framework for the verification of the dynamic behavior of reactive systems modeled using UML (Unified Modeling Language) statechart diagrams. The model is translated to an intermediate representation by parsing the information embedded behind the UML statecharts, this intermediate representation is used for checking the safety violations. Verification framework proposed is scalable to complex systems.

D.2.4 [**Software Engineering**]: Software/Program Verification – Model Checking:

General Terms: Design, Verification

Keywords
Statechart, Unified Modeling Language (UML), Framework.

1 INTRODUCTION
The development of dependable software has been the major goal for the advent of software engineering discipline. The traditional way of verifying software systems is through human inspection, simulation, and testing. Though these methods are cost effective, unfortunately these approaches provide no guarantee about the quality of the software. Human inspection is limited by the abilities of the reviewers, simulation and testing can only explore a minuscule fraction of the state space of any software system. Model driven software development has been a prominent means to enhance the understandability of the system's structure and behavior. It has prompted industries to develop tools which can generate the code from the model in high level languages like C, C++ or JAVA (IBM's Rational Rose RT [1] is one such tool for the development of embedded real time systems). As deployable binaries are generated from the model, ensuring model's correctness becomes highly essential. The commonly used formal model verification technique is model checking. Model checking [2] is a pragmatic technique that, given a finite-state model of a system and a logical property (expected system property), systematically checks whether model holds the property or not. If the model does not hold the expected property, an error trace (also called as counter example) is generated. The original model can be refined by leveraging information generated by the counter example; this approach is known as counter example guided model refinement [3]. Several model checking tools like SPIN (Simple Promela INterpreter) [4], SMV (Symbolic Model Verifier) [5], SLAM[6], BLAST (Berkeley Lazy Abstraction software verification Tool) [7] and RuleBase [8] are in existence. The major drawback of using model checking tools for verification is that, they expect system to be modeled using their proprietary input language. The input languages of most of these tools are text based and lacks advantages of visual representation. Numerous researchers have tried to address this issue. We have done thorough review of the earlier works [9] and found that, though they suggest modeling the dynamic behavior of the system using UML(Unified Modeling Language) statechart diagrams (provides visual representation to the models), subsequently these statechart diagrams are translated to the input language of the model checker before verification. The translation process removes the abstraction of the models and exponentially increases the state space of the complex systems. This could lead to state-explosion-problem [10]. We in this paper present a verification frame work which avoids the usage of off-the-shelf-model-checker and translation of UML statechart models to input language of the model checker, hence the state explosion is minimized.

In the next section we provide necessary introduction to UML statechart diagram and in section 3, framework for verification of complex reactive systems is presented. We draw conclusions in section 4.

ISSN 0974-1003

2 BACKGROUND

2.1 UML Statechart diagram

The Unified Modeling Language is a general purpose visual modeling language that is designed to specify, visualize, construct and document the artifacts of the software system The UML is simple, powerful and based on a small number of core concepts that most object oriented developers can easily learn and apply. The UML specification consists of two interrelated parts:

- UML semantics[1]: A meta model that specifies the abstract syntax and semantics of the UML object modeling components
- UML Notation: A graphic notation for the visual representation of the UML semantics

The UML has a collection of graphical notations each with a well defined semantics. It allows construction of several diagrams using these notations and relationship among them. These diagrams aid to visualize a system from different perspectives. The UML includes class diagram, sequence diagrams and statechart diagrams that can be used to specify both structural (class diagrams) and dynamic (sequence and statechart diagrams) views of software systems.

The UML statechart diagrams are used for behavioral modeling of the software and greatly increase the understanding of a system by revealing inconsistencies, ambiguities, and incompleteness that might otherwise go undetected. The statecharts were first introduced by David Harel in 1987 [12] as a visual formalism for complex reactive systems. The primary motivation behind this model was to overcome the limitations inherent in conventional state transition diagrams (or state diagrams for short) to describe the complex systems. State diagrams are directed graphs, with nodes denoting states, and arrows denoting the transitions. The UML statechart diagrams extend state diagrams to include notions of hierarchy (ability to cluster many states into a super state) and concurrency (orthogonality). Statechart shows sequence of states that an object goes through during its life cycle in response to stimuli, together with its responses and actions. A state in a statechart is represented by rounded rectangle and can be recursively decomposed into exclusive states (OR-state) or concurrent states (AND state). A simple transition may have a triggering event (whose occurrence cause the transition to take place), an enabling guard condition (which must be true for the transition to be taken), and output event and actions, all of which are optional. When a transition in a statechart is triggered (i.e. an event is received and guard condition becomes true), the object leaves its current state, initiates the action(s) for that transition and enters a new state. Any internal or external event is broadcasted to all states of all objects in the system. Transitions between concurrent states are not allowed, but synchronization and information exchange is possible through events. The transitions are represented by directed arrows with labels showing triggering event guard condition and actions (optional). An initial state is shown as a small solid filled circle and a final state is shown as a circle surrounding a small solid filled circle. The final state represents the completion of activity in the enclosing state

[1] It is not discussed in this paper and interested readers can refer to [11] for details.

The Fig.1a is an abstraction of hypothetical "Training" system, modeled using UML statechart diagram. The Figures 1b and 1c shows expansion of Attend Course and Examination states of Fig 1a into concurrent sub states (AND states) and sequential sub states (OR states) respectively. Visually AND state is depicted by splitting the state using dashed lines as shown in Fig. 1b. The Fig.1d shows completely expanded/flattened view of the abstract model, called global representation (without hierarchy).

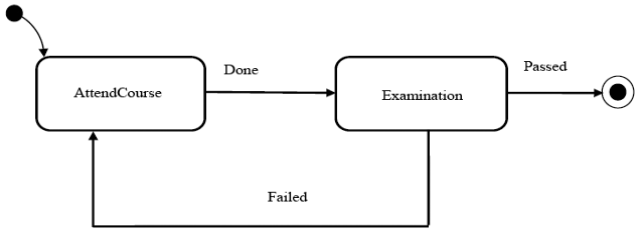

Figure 1 a. Abstract statechart model of "Training system"

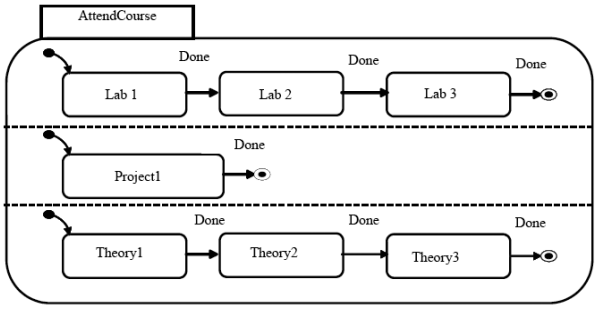

Figure 1 b. Concurrent refinement

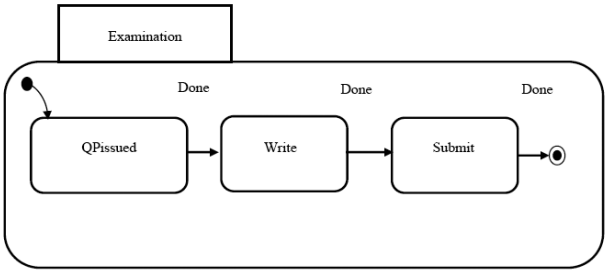

Figure 1 c. Sequential refinement

2.2 Earlier works

A widely known approach for verifying the complex systems is, by modeling them in the input language (mostly text-based) of the off-the-shelf model checker and passing them on to model checker. The property expected is specified in temporal logic. Subsequently, the need of visual formalism to the models was realized and UML statecharts were used for modeling dynamic

behavior of the system. The verification of such models is done by first representing the UML statecharts in Extended Hierarchical Automata (EHA) [13] [14] [15] and then mapping to input language of the model checker. This approach is well received and successful for less complex systems. As the complexity of the system grows, this technique of flattening (removal of abstraction) the original model during verification would lead to "state-explosion" and hence aborts the verification process.

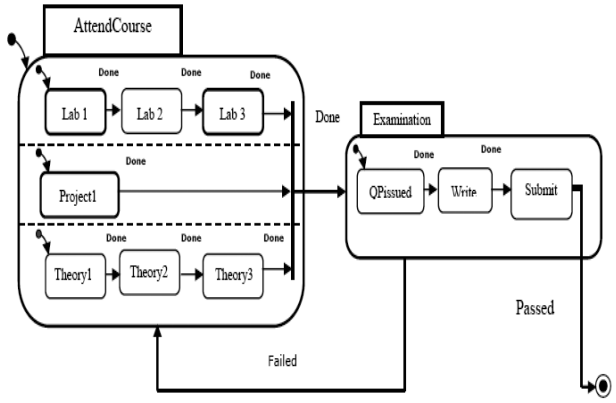

Figure 1 d. Global representation

3 VERIFICATION FRAMEWORK

In this section, we present a framework, which supports the construction of automated safety violation analysis tools for UML statechart diagrams. This can be integrated with industrial CASE tools (which generate code from the models) for verifying models

3.1 Methodology

Our approach depicted in Fig. 2 does not use off-the-shelf model checker and hence translation to any other modeling language is avoided. The logics of the UML statechart diagram are captured in a textual format and then the same is converted to state space graph (Kripke Structure). Unlike most of the model checkers, here the data structure preserves the abstraction and whole state space graph is not brought into memory at once. The verification is done iteratively so that the state explosion problem discussed earlier in the paper would be minimized.

3.2 Architecture

The architecture of UML model verifier shown in Fig. 3, automatically searches the complete set of states of the state space for an incorrect behavior and out puts error trace if any. There are four principal components, viz.,

- The UML editor
- Model compiler
- Property extractor
- Checker

The UML editor allows creating both abstract and concrete statechart models of the reactive systems. It supports all UML visual notations and semantics to capture all important design decisions. The user is prompted to enter relevant information as (s) he creates the model. This .information is later used by "model compiler" for constructing intermediate representation of the model.

The Model Compiler reads the UML statechart model and generates an intermediate representation of the same by parsing the UML statechart. This intermediate textual representation is eventually translated to state space graph.

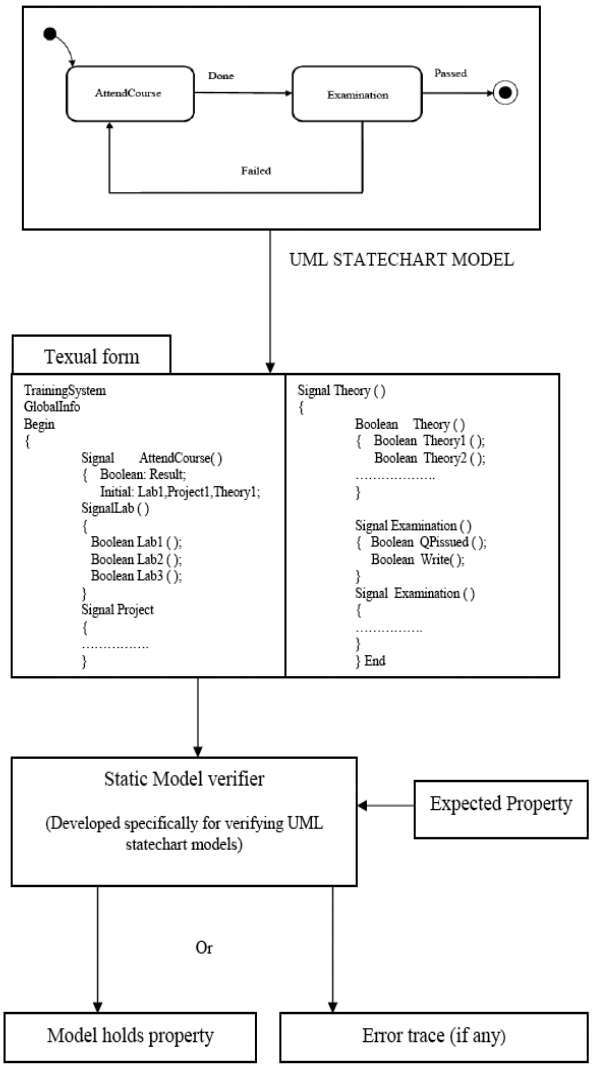

Figure 2. Verification model for UML statecharts

The Property Extractor extracts model properties such as set of valid states, transitions and events from the textual form and the state space graph. It also gets safety property to be verified from the external world**.**

The Checker searches the state space iteratively for the safety violations. It outputs "Yes", if the specified safety property holds, otherwise outputs "No" and the error trace (counter example). As

this is the core part of the verifier we discuss in detail, the technological aspects of this module in the next section.

3.3 Checker Algorithm

Detection of bad states in huge state space graph becomes hard at times. This is due to the limited availability of resources, memory in particular. Therefore devising a memory efficient algorithm is indispensable.

An iterative search approach is presented below:

Step 1: Start from the abstract level
Step 2: Let S represents set of reachable sates (given by property extractor)
Step 3: Let Ø be the expected property (let Ø be the safety violation or bad state)
Step 4: Let I be the set of initial states (given by property extractor)
Step 5: Find out set s which can be reached in one step from the current state
Step 6: Search the resulting s to find Ø
Step 7: Iterate the steps 5 and 6 until no new states are visited or Ø is found.
Step 8: Deepen/expand the part of the model and iterate the steps 2 through 7, until all the parts of the model is exhaustively searched
Step 9: If Ø is not found output "Yes" otherwise output "No" and path from the initial state.

This divide and conquer search algorithm is memory efficient as it does not keep the entire state-space of the model in the memory instead iteratively expands the model.

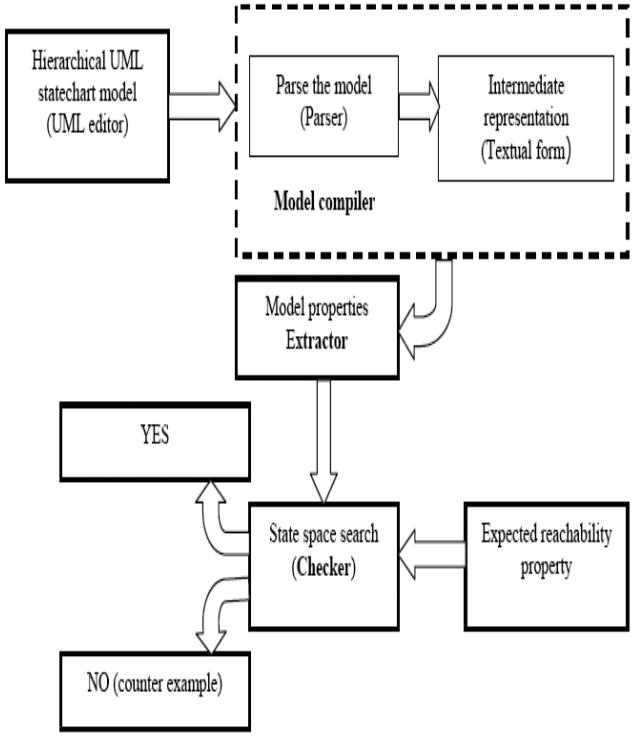

Figure 3. High-level design of UML model verifier

4 CONCLUSIONS

Most of the existing approaches translate UML statechart model into text based modeling language which can then be verified using off-the-shelf model checker. This translation process removes the salient features of the statecharts like hierarchy or abstraction. In other words, flattening of the statecharts leads to large state space requirement and makes verification approach not scalable to complex systems. The proposed iterative verification technique does not keep the entire state space of the UML statechart models in the memory. Hence, system with large state space can be handled efficiently. The usage of off-the-shelf model checker for verification of UML statechart models is avoided here; therefore no translation of UML models to the proprietary language of the model checker is necessary.

5 ACKNOWLEDGMENTS

We thank IBM India Limited – software labs, Bangalore for supporting us to take up this work and providing us necessary information.

6 REFERENCES

[1] IBM Rational Rose Real Time.2007 http://www.ibm.com/developerworks/rational/library/797.html , (visited on 01/12/2007)

[2] Edmund M. Clarke,Jr., Orna Grumberg and Doron A. ,1999 Model Checking (The MIT press, 1999)

[3] Edmund M Clarke, Ansgar Fehnker, et.al.2003: Abstraction and Counterexample refinement fin model checking of Hybrid Systems, Vol.14, No 4, International journal of foundations of computer science, (2003), 583-604

[4] Gerard J. Holzmann 1997, The Model Checker Spin, IEEE Trans. on Software Engineering, Vol. 23, No. 5, (1997), 279-295

[5] Kenneth L. Mc. Millan 1992, Symbolic Model Checking: An approach to the state explosion problem, (Ph.D thesis submitted to Carnegie Mellon University (CMU), 1992)

[6] The SLAM project 2007, http://research.microsoft.com/slam/ (visited on 13/10/2007)

[7] Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, Grégoire Sutre, Software Verification with BLAST. 235-239, Electronic Editions (Springer LINK)

[8] I. Beer, S. Ben-David, C. Eisner and Landvar 1996: RuleBase-an industry-oriented formal verification tool, Proceedings of 3rd Design Automation Conference (DAC), Association for Computing Machinery Inc.,(1996), 655-660.

[9] C.M. Prashanth, Dr. K.C. Shet, Janees Elamkulam 2007, "A Reality chek of model checking the UML statechart models and research directions", International conference on Computers, Communication, Control systems and Instrumentation, Bangalore, India , 21-22 (November 2007) pp 16-22.

[10] Valmari A. 1998: The State explosion Problem, Lectures on Petri Nets I: Basic Models, LNCS 1491, Springer-Verlag (1998) , 429-528

[11] Rational software 2007, IBM, Microsoft et al, UML semantics OMG document: (visited on 02/09/2007). ftp://ftp.omg.org/pub/docs/ad/97-08-04.pdf

[12] D. Harel 1987, Statecharts: A Visual Formalism for Complex Systems, Science Computer Programming 8, (1987), 231-274.

[13] Diego Latella, Istvan Majzik and Mieke Massink, 1999 Automatic verification of a behavioural subset of UML statechart diagrams using the SPIN model checker, Formal Aspects of Computing, volume 11(6), (1999), 637-664

[14] G.J. Holzmann et.al., 1998 Implementing statecharts in PROMELA/SPIN, proc. workshop on industrial strength formal specification techniques WIFT'98, USA, IEEE computer society, (1998).

[15] Adam Darvas et.al., 2002 Verification of UML statechart models of embedded systems 5th IEEE design & diagnostics of electronic circuits and systems workshop, (2002),70 -77