Defence Science Journal, Vol. 72, No. 1, January 2022, pp. 40-48, DOI : 10.14429/dsj.72.17228

Generic Methodology for Formal Verification of UML Models

K.H. Kochaleema[#,*] and G. Santhosh Kumar[#]

[#]*Department of Computer Science, Cochin University of Science and Technology, Kochi - 682 022, India*
[*]*DRDO-Naval Physical and Oceanographic Laboratory, Kochi - 682 021, India*
[*]*E-mail: kochaleema@npol.drdo.in*

ABSTRACT

This paper discusses a Unified Modelling Language (UML) based formal verification methodology for early error detection in the model-based software development cycle. Our approach proposes a UML-based formal verification process utilising functional and behavioural modelling artifacts of UML. It reinforces these artifacts with formal model transition and property verification. The main contribution is a UML to Labelled Transition System (LTS) Translator application that automatically converts UML Statecharts to formal models. Property specifications are derived from system requirements and corresponding Computational Tree Logic (CTL)/Linear Temporal Logic (LTL) model checking procedure verifies property entailment in LTS. With its ability to verify CTL and LTL specifications, the methodology becomes generic for verifying all types of embedded system behaviours. The steep learning curve associated with formal methods is avoided through the automatic formal model generation and thus reduces the reluctance of using formal methods in software development projects. A case study of an embedded controller used in military applications validates the methodology. It establishes how the methodology finds its use in verifying the correctness and consistency of UML models before implementation.

Keywords: Formal verification; Computational tree logic; Linear temporal logic; Property specification, State chart diagram; UML modelling

1. INTRODUCTION

Unified modelling language (UML) is a widely accepted modelling language, with versatile capabilities for visualising, specifying, constructing, and documenting software systems[1]. It provides artifacts that can represent structural, temporal, and functional views of the system under modelling. Currently, UML models are used for modelling all kinds of systems, including real time and embedded systems. In safety critical embedded software development, model verification and its validation are vital, particularly for warranting safety and reliability requirements during operation. UML models need to be verified formally before implementation, because they may contain unforeseen behaviour, leading to unexpected system states and error conditions. This has made verification and validation integrated development process like V-Model prevalent in industry. Formal methods provide effective verification techniques and integration of these techniques in early development stages increases quality and reliability, for lesser budgets[2]. This paper proposes a methodology for UML model verification using temporal logic (TL) property specifications. The technique is unique for its spontaneous transition from UML to formal domain and consistency across UML artifacts such as, Use Case diagrams and Statechart Diagrams. The method can fit in any Software Development Life Cycle (SDLC), making it practicable in real-world software development projects.

Received : 16 June 2021, Revised : 04 August 2021
Accepted : 18 November 2021, Online published : 05 January 2022

UML based model driven software development is prevalent in the embedded system industry. These models need to be verified formally, before code generation to mitigate the risk of software errors. Many studies have been reported in the past, discoursing the application of model checking and formal techniques in UML artifacts. Majority of the tools and techniques discussed, revolve around the idea of translating various UML artifacts to an input formal specification language, comprehensible by a model checking tool. Johan and Paltor[3] converts UML state machines to formal semantics in PROMELA and then uses SPIN for model checking. The vUML tool as reported by Johan and Paltor[4] is also based on this technique. Translation of UML Statecharts to input specification language of SPIN model checker is detailed by Latella[5,6], *et al.*. In another study by Gihwon[7], he describes a technique, which translates UML Statecharts to input language of SMV model checker. In a later study by David[8], Statecharts are converted to Timed Automata for model checking by UPPAAL. Zhang and Liu[9] discusses a technique which translates statecharts to the input language of another model checker, PAT.

Zamira[10], *et al.* have studied the UML-VT environment for translating UML models to input specification language of SPIN, UPPAAL and NuSMV. To assess the efficacy of activity diagrams in behavioural modelling of requirements, Eshuis[11] translated these diagrams for use by NuSMV model checker. Santos[12] *et al.*, reported the transformation of UML behavioural diagrams to support model checking using NuSMV. All these approaches rely on model transformation to

an input specification language of model checking tools like, SPIN, UPPAAL, NuSMV, rCOS, etc. Importantly, considering the past studies, this paper proposes a relatively robust process, integrated with formal verification, for detecting defects early in UML model driven SDLC.

In another recent study by Kochaleema and Santhoshkumar[13], a UML model-based formal verification methodology is described, translating UML diagrams to formal models and thus incorporating semantics to them. Specifically, this approach commences with Use case modelling and analyses the chosen critical behaviour using UML Statecharts. A State Transition Matrix is derived from this, and a Translator application translates this matrix to Labelled Transition System (LTS). Formal properties are derived from performance specifications and are expressed as Computational Tree Logic (CTL) formulae. The existing CTL model checking algorithm verifies property entailment in the LTS generated. The methodology is validated using the case study of an embedded controller widely used in military applications.

Existing approaches in past literature are tightly coupled with the input specification language of model checking tools and need UML model transformation amenable by the model checker. This demands knowledge in formal languages and can only be performed by an expert. This technology gap is addressed by Kochaleema and Santhoshkumar[13] in their recent study on Methodology for integrating Computational Tree Logic in UML artifacts. It synergises UML models with formal methods through an automatic translator application and combines the advantages of Model-Driven Engineering with formal methods; and supports analysis, design, Verification and Validation (V&V) activities of the embedded software development cycle.

Embedded systems used in military applications are mainly developed using UML based CASE tools, and validation of these models is mandatory for flawless implementation and consequent reliable system performance[13]. Though integration of formal methods with UML is a promising technique, its practice in the embedded software development domain is minimal. Embedded system developers avoid using model checking tools mainly for their steep learning curve and lack of tool support. Automation of formal verification methods can increase the use of formal methods in software development, especially for safety-critical software applications used in military systems[14,15]. The methodology proposed in this study makes a greater step in solving this concern. It provides user-friendly formal method integrated UML artifacts that can be used for model checking both Linear Temporal Logic (LTL) and CTL specifications. It conceals formal aspects in an engine, making it a practitioner-friendly modelling methodology.

The LTL based approach discussed in this study extends the methodology for Integrating CTL Model Checking in UML Artifacts, with sequential behaviour model checking feature, enhancing the verification capability to both sequential and branching-time behaviour. The method proposed by Kochaleema and Santhoshkumar[13] has three layers, viz., UML Layer, Interface Layer, and a Formal Verification layer. The Formal Verification layer caters for CTL property specification, followed by CTL model checking. However, sequential timeline system dynamics will also be there for embedded software systems, influencing the modelling requirements. Consequentially, the reasoning over linear time scale becomes binding, suggesting itself with the extension of this methodology with LTL specifications. The modelling methodology applied for both CTL and LTL specifications led to the conception of a generic model verification approach, comprehensive enough to capture all kinds of behaviour.

The main contribution of our method is a UML-LTS (Labelled Transition System) Translator application, making formal method integration possible with UML artifacts. In addition, the proposed methodology is process-oriented. It commences with system requirements, progresses through functional and behavioural modelling, and maintains consistency across phases in scenario names, state labels, events, and associated expressions. It offers modularity while modelling systems with complex behaviour through the innate selection of critical behaviour for formal verification. Automation achieved in the formal model generation reduces the reluctance of using formal methods for verification and validation, escalating the application of formal methods in software development projects

2. FORMAL VERIFICATION INTEGRATED UML MODEL CHECKING METHODOLOGY

The workflow of the proposed methodology is shown in Fig. 1. It puts forward an approach to integrate UML-based visual abstraction models with LTS. The steps involved are listed below.

(i) Use case modelling of scenarios, selected through

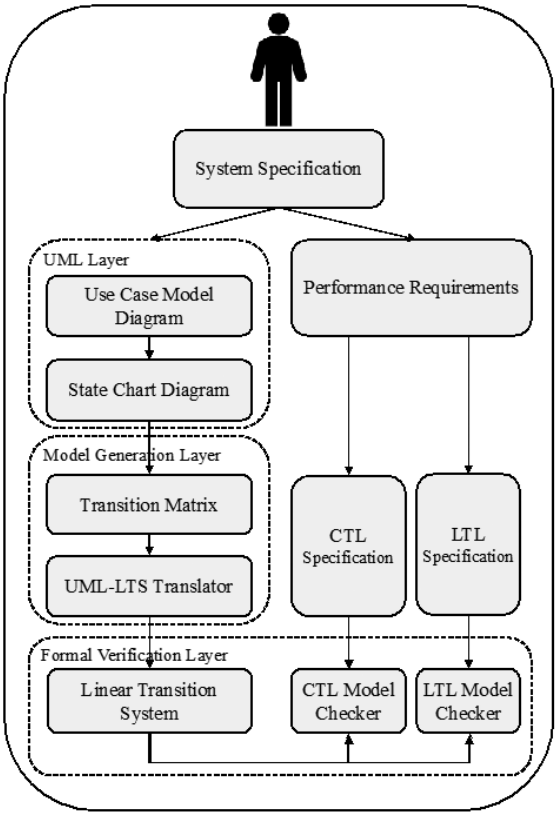

Figure 1. Methodology workflow.

criticality assessment.

(ii) Behavioural modelling using Statechart diagrams.

(iii) UML model to formal model transformation using UML-LTS Translator.

(iv) Property specification in CTL syntax for branching behaviour or LTL syntax for sequential behaviour.

(v) Formal verification using CTL or LTL model checking.

Primarily, the chosen specification is analysed using UML Use cases and Statecharts. From Statecharts, the modeller generates a Transition Matrix, enumerating source state, state labels, transitions and target states. A UML-LTS Translator application converts this matrix to LTS automatically. Now, modeller can make use of system performance requirements and derive corresponding property specifications in Temporal Logic (TL) syntax[16,17]. CTL/LTL model checker[18] verifies for CTL/LTL property entailment in the LTS generated by UML-LTS translator.

The approach finds its application in requirement modelling and analysis of safety-critical systems. The most critical behaviour shall be chosen from system specifications. As guided by the Failure Mode Effect Analysis (FMEA) of the system, the modeller can make this selection.

The methodology chooses the most challenging system behaviour whose failure can cause the most catastrophic consequence during operation and conducts a lightweight formal abstraction i. It simplifies the process of formal verification by modelling selected safety-critical behaviour. Model verification can bring substantial benefits in proving the reliable operation of the system much before implementation. The method ensures correctness and robustness attributes during the requirement analysis phase itself, remarkably reducing the probability of failures and debugging efforts.

In the next section, a case study for demonstrating the feasibility of the proposed approach is illustrated. A real-life system used for naval applications is considered for grander impact during the illustration.

3. METHODOLOGY VALIDATION USING LTL CASE STUDY

The methodology mentioned above was reported[13] recently with branching time abstraction and CTL model checking, stating its suitability for model checking decisive behaviour. The present study adds to this workflow, with sequential behaviour modelling and LTL model checking functions, making it suitable for modelling all types of embedded system behaviours. The addition of this feature expands its scope and applicability in model- driven engineering. The enhancements are made in property specification and formal verification phases. The property specification is comprehended using LTL, for linear time system behaviour. Therefore, formal verification is accomplished by LTL model checking. The UML-LTS translator can generate the formal model, irrespective of branching or linear time behaviour.

Application of this approach in modelling and verifying an embedded controller behaviour is delivered in the following subsections. The context of operation of the controller is given in section 3.1, followed by other subsections illustrating successive steps in the methodology.

3.1 Embedded Controller - System Description

Sensor deployment controller (SDC) is an embedded controller used in airborne sensor processing systems. It controls the deployment of acoustic sensor structures in deep-sea from an airborne platform. The context of operation of SDC is shown in Fig. 2. GUI provides necessary operator interface for SDC. Sensor is deployed underwater using Winch, mainly for data collection and further processing by Data Acquisition System onboard. Assuring safe and secure operation is very critical for Sensor assembly as well as aircraft. SDC monitors various winch sensor parameters as a safety measure. The speed with which the Sensor is deployed and hoisted is also a critical parameter, controlled by SDC. The speed values are generated based on progressive depth values of the immersed sensor structure. This particular requirement is reasoned over sequential timeline, setting up an LTL specification scenario.

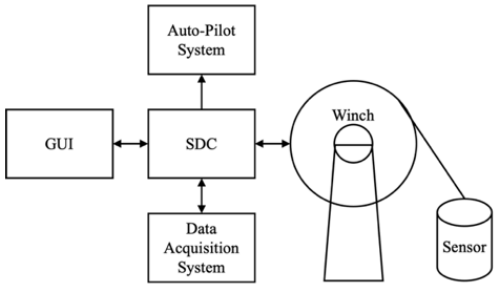

Figure 2. Sensor deployment controller - context of operation.

3.2 Methodology Workflow

The five-step workflow stated in Section 2 is firmly complied and discoursed in subsequent sections.

3.2.1 Use Case Modelling of Selected Scenario

The four use cases identified are shown in Fig. 3.

WinchMotorHandling handles all the functionalities involved in controlling the electric motor and its speed during operation.

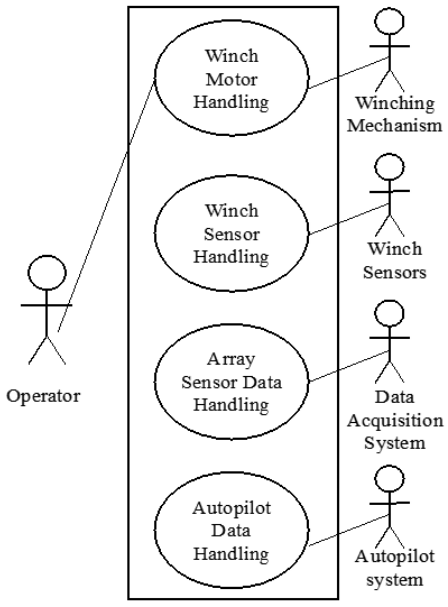

Figure 3. Lowering sensor structure - use cases.

WinchSensorHandling takes care of mechanical sensor interfacing and safety interlocking requirements.

ArraySensorDataHandling encompasses the data communication with on-board data acquisition system.

AutopilotDataHandling abstracts the interface requirements with Flight Control System/Auto-Pilot System onboard aircraft.

3.2.2 Behavioural Modelling using State Charts

Lowering Sensor Structure has composite behaviour with four independent sub-behaviour branches, each with its own dynamic behaviour. *WinchMotorHandling* sub-behaviour is linear, whereas *WinchSensorHandling* sub-behaviour is branching in nature.

In this case study, *WinchMotorHandling* is chosen as sample behaviour, for its linear time dynamics. The sequences of states and the progressive transitions of these states, following the submerged depth values are shown in Fig. 4. SDC maintains the motor speed at given values at specific depth of immersion. As the depths of immersion increases, the speed also changes, as indicated in Fig. 4.

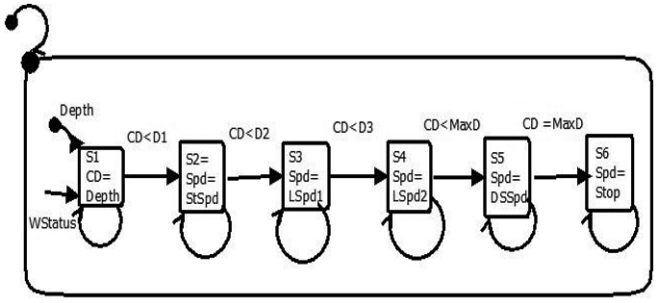

Figure 4. WinchMotorHandling Statechart.

3.2.3 UML Model to Formal Model Translation

The novelty of the methodology lies in this stage. The visual model in UML notations is transformed to formal notations, comprehensible by the model checking algorithm. It is a two-step process, with following micro-steps.

(i) UML Statechart to State Transition Matrix (STM) Translation.

(ii) State Transition Matrix to LTS Translation.

(i) UML Statechart to STM Translation

The STM tabulates the progressive behaviour of the controller over time. The statechart in Fig. 4 is mapped to STM in Table 1, following the procedure given:

Table 1. State transition matrix

Source state	Label AP	Transition AP	Target state
S_1	CD = Depth	CD < D1	S_2
S_2	Speed = StartSpeed	CD < D2	S_3
S_3	Speed = LowSpeed1	CD < D3	S_4
S_4	Speed = LowSpeed2	CD < MaxD	S_5
S_5	Speed = DSSpeed	CD = MaxD	S_6

(a) Sequentially visit states in Statechart Diagram. Enter state visited in source state column, say S_1.

(b) Fill up label Arithmetic Proposition (AP) column with invariant conditions of this state.

(c) Enter next state S_2 in target state column. Also fill up transition condition in transition AP column.

(d) Continue generating subsequent rows of the matrix, repeating steps (a), (b) and (c) till all states are visited.

(ii) STM to LTS Translation

In this step, the UML-LTS translator application program developed by authors, converts the STM to LTS. The automatic conversion from STM to LTS is the most inventive step in this approach. The UML-LTS translator (Fig. 1) scans input STM and generates LTS automatically; serving as an interface between UML and formal domains.

The output of the UML-LTS translator appears as shown in Fig. 5, clearly showcasing state transitions. The LTS generated is perceptibly a formal representation of the Statechart in Fig. 4. They both have same number of states, transitions, and labels, underscoring the smooth transition from UML to the formal domain.

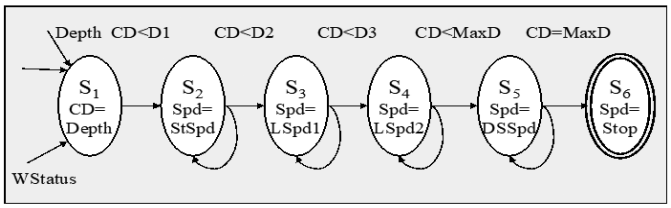

Figure 5. LTS for WinchMotorHandling.

3.2.4 Formal Property Specification

The modeller can deduce the formal property, right from requirement specification. In this case, the specification states that the embedded controller shall control the speed of electric motor to safe limiting values during lowering, following the position of the immersed sensor structure. Based on the depth values received, the motor speed can only be at limiting values or within allowable values. The same, when expressed as a logical expression, will turn out to be:

$$\text{MotorSpeed} \leq \text{AllowedSpeed}$$

This becomes the safety property that must always hold during lowering operation. Using LTL syntax, formalise this expression as given in equation (1).

$$G\varphi{:}\text{-}\neg\varphi(\text{MotorSpeed} \leq \text{AllowedSpeed}) \quad (1)$$

MotorSpeed ≤ AllowedSpeed is Globally TRUE.

3.2.5 Formal Verification

The objective of this phase is to verify that the formal model generated satisfies the above safety property during various runs of the system. The LTS generated from UML Statechart (Fig. 5) is the formal model M and equation (1) becomes the formal property specification φ. LTL model checking algorithm as defined in literature[18] verifies that all traces of M satisfy φ. If any trace is not satisfying φ, it can be used as a counterexample, and the model can undergo revisions until there are no counterexamples.

LTL Model Checking Algorithm

The execution sequence obtained while verifying the entailment of LTL property φ in model M (M |= φ) adhering to the LTL model Checking algorithm is given below. It shows that all infinite executions of M satisfy φ, and therefore, the model M is correct. The UML Statechart model from which this LTS is generated is thereby claimed to be correct. Code generated faithfully from this model assures reliable operation while exhibiting this behaviour.

Model Execution

Step 1 - Convert M to automaton AM

WinchMotorHandling LTS in Fig. 5 is converted to automaton AM, with initial state S_0 and final state S_6 (Fig. 6).

Step 2 - Convert negated property ¬φ to automaton $A\neg\varphi$

The property φ given in equation (1) is negated and corresponding automaton $A\neg\varphi$ is shown in Fig. 7. When the depth values received are less than MaxD, it remains in q_1 and on the reception of MaxD depth value, it goes to q_2. In negated property automaton, the non-final state of the original automaton becomes the final state, and therefore, q_1 is marked as the final state.

Step 3 - Compute product of AM and A ¬φ

Pair states *t* of AM and A of $A\neg\varphi$ together iff the set of propositions *P* (φ in this case), true in *t* is exactly ($AM \cap P$).

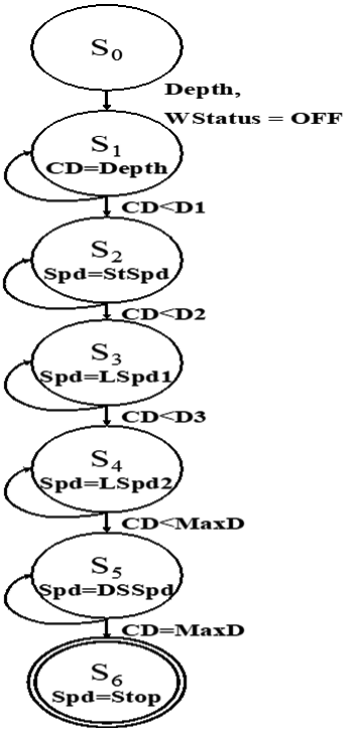

Figure 6. Automaton AM.

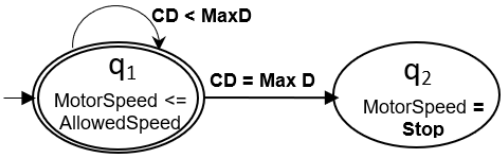

Figure 7. Negated property automaton, $A\neg\varphi$.

The product automaton obtained by taking cross product of AM in Fig. 6 and $A\neg\varphi$ in Fig. 7 is drawn in Fig. 8.

$$AM = \{S_0, S_1, S_2, S_3, S_4, S_5, S_6\} \quad (2)$$

$$A\neg\varphi = \{q_1, q_2\} \quad (3)$$

$$AM \times A\neg\varphi = \{S_0q_1, S_1q_1, S_2q_1, S_3q_1, S_4q_1, S_5q_1\} \quad (4)$$

These are the states in AM X $A\neg\varphi$ where φ is true and is exactly AM ∩ P.

Step 4 - Look for an accepting path in the product. If such a path exists, there is a counterexample to the claim that M satisfies the property φ.

Equation (4) shows that there are no accepting paths in the set obtained by taking AM X $A\neg\varphi$ and hence it is stated that model M in Fig. 5 satisfies the property given in equation (1). The above illustration leads to the following conclusion. Based on the depth parameter, the controller always maintains the motor speed at limiting values during lowering operation.

Step 5 - If no such path exists, then M satisfies φ

Obeying step 4, there are no counterexamples in this case, and therefore the formal model M in Fig. 5 satisfies the property φ in Eqn (1).

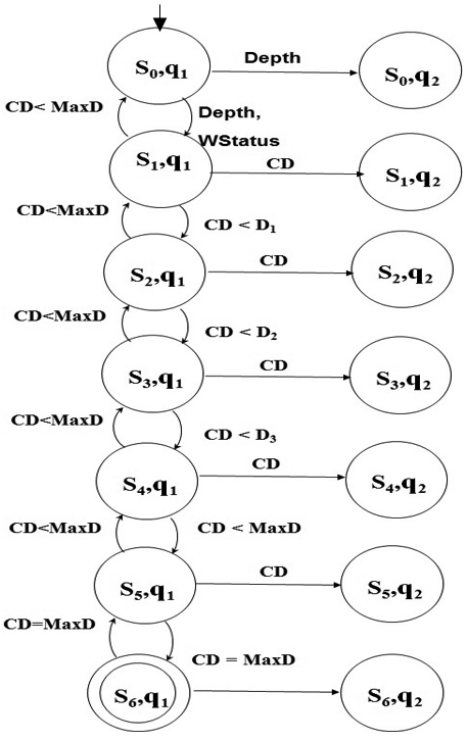

Figure 8. AM x A ¬φ.

Model Checking Result

According to Step 4 of the algorithm, no counterexample exists for the claim, since there are no accepting paths in the product automata (Equation (4). *M satisfies the property* φ and therefore Step 5 is fully complied. Hence, the formal model of **WinchMotorHandling** behaviour satisfies the safety property, *MotorSpeed <= AllowedSpeed* in all its infinite executions. This proves that the source behavioural specification, abstracted in UML Statechart, satisfies the above property.

Inference: *Formal Model M entails* φ, *and hance the source UML model satisfies* φ.

This step completes the case study and validates the methodology for modelling and verifying LTL behaviour. To demonstrate and confirm the generic nature of the methodology, the same procedure is applied in CTL model checking, and appropriate steps alone from the methodology is illustrated in Section 4 and sub-sections.

4. TEMPORAL LOGIC MODEL CHECKING

The linear time behaviour modelling and case study discussed above are part of the study towards generalising the proposed methodology, primarily for promoting its use in software projects. The sequential modelling add-ons, along with branching time behaviour modelling, offers a unified UML-based model checking solution suitable for all types of embedded system behaviours. It has a generic UML-formal model Translator, CTL, LTL property specification and model checking, together in one package. It can exhaustively explore all possible system behaviours, distinctly different from test scenario simulation and testing, wherein only some of the possible behaviours can be analysed. Embedded systems do exhibit branching time and linear time behaviours during operation, and formal verification of LTL and CTL property specifications in one common methodology is very advantageous, especially from a usability perspective. To establish the generic nature of the methodology, a concise description of CTL behaviour specification and model checking using the proposed methodology is given below.

4.1 Behavioural Modelling using UML Statechart Diagram

The Lowering Sensor operation embodies a composite behaviour with sequential and branching-time behaviour, as discussed in section 3.2.2. The branching time sub-behaviour, **WinchSensorHandling,** is chosen as demonstrator Statechart, with branching- time behaviour. The exact sequence of states and transitions taking place upon reception of various events are shown in Fig. 9. There are three inputs, In_1, In_2 and MSwitch. A status parameter is set if In_1 values are within the expected range; otherwise, it goes to an error state. Similar is the case for the In2 sensor also. The ON status of MSwitch also leads to the error state. . If all three sequences are behaving normal, it goes to compute speed state, S_7.

4.2 Formal Model Generation

The UML to formal model translation is achieved through the same steps, as described in section 3.2.3. The LTS generated using UML-LTS translator program is shown in Fig. 10.

4.3 Formal Property Specification

The decisive nature of the model led to CTL property specification and for demonstration, only one safety property is considered here.

SafetyProperty1 (ϕ_1)

According to system safety specifications, if I_1 sensor value goes beyond permissible limits (I_1 >Angle), lowering shall be suspended and Error is indicated to Operator. This

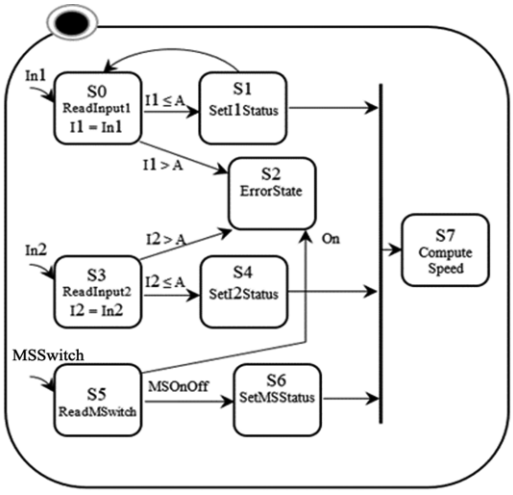

Figure 9. WinchSensorHandling – Statechart.

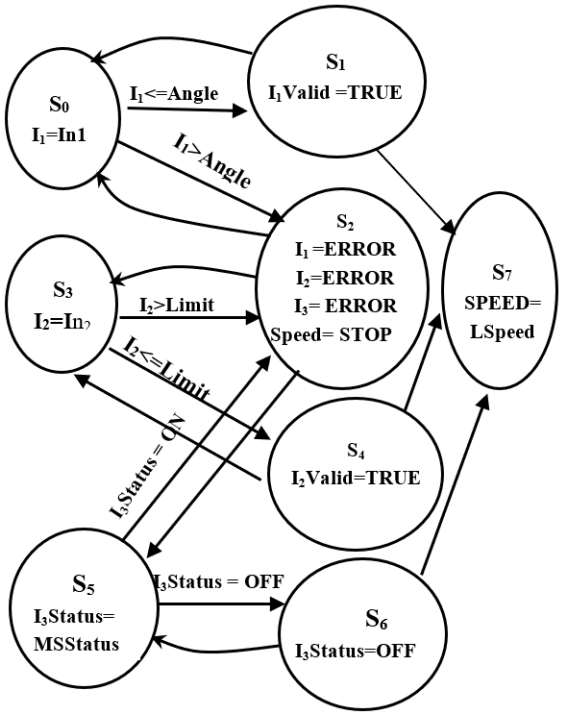

Figure 10. WinchSensorHandling – LTS.

requirement is translated into CTL formula, given in (5).

$$\phi_1 = AG\,((I_1 > \text{Angle}) \rightarrow AF\,(STOP \wedge ERROR)) \qquad (5)$$

4.4 Formal Verification

Formal verification is carried out using CTL model checking algorithm[18]. This algorithm checks that the initial states of M during model executions, satisfy the CTL formula ϕ, ($M \models \phi$). The LTS generated from UML Statechart (Fig. 9) is the formal model *M* and equation (5) becomes the formal property specification ϕ.

Model-checking Algorithm

(i) *Construct the denotation of* ϕ *where the formula holds:*

$[\phi] := \{s \in S : M,s \models \phi\}$

(Denotation $[\phi]$ *is the set of states where* ϕ *holds)*

(ii) *Then compare with the set of initial states:*

$I \subseteq [\phi]$?

To compute$[\phi]$*:*

Proceed "bottom-up" on the formula structure, computing $[\phi_i]$ *for each sub formula* ϕ_i.

Model Execution

Starting from posterior of ϕ_1 in Eqn. (5), progressively derive subformulae and generate $[\phi_i]$, the set of states in M, where each subformula holds. Denotation $[\phi_i]$ generated in this case is enumerated in model execution sequence shown. The execution stops when the subformula grows to ϕ_1.

The first subformula (STOP ∧ ERROR) holds in state S_2.

(i) $(STOP \wedge ERROR)]$; $[S_2]$

(ii) $[AF\ (STOP \wedge ERROR)]$ $[S_0, S_2]$

(iii) $[(I_1 > Angle)]$ $[S_0, S_2]$

(iv) $[(I_1 > Angle) \rightarrow AF(STOP \wedge ERROR)]$ $[S_0, S_2]$

(v) $[AG\ ((I_1 > Angle) \rightarrow AF(STOP \wedge ERROR))]$ $[S_0, S_2]$

Model Checking Result

Denotation $[\phi_5] = [S_0, S_2]$

Initial state S_0 is a subset of $[\phi_5]$ and satisfies step 2 of the CTL model checking algorithm. Therefore, model M in Fig. 10 satisfies safety property ϕ_1 given in equation (5).

Inference: *Formal Model M entails* φ_1 *and hence the source UML model satisfies* φ_1.

5. RESULTS AND DISCUSSION

The present study led to the invention of a formal method integrated visual modelling and verification methodology for embedded software development. It can exhaustively explore all possible behaviour of the system under all input conditions and lead to verified and validated system specifications well before implementation. This ensures reliable and fail-safe systems during operation. The methodology complies well with the V-model development process.

The characteristic features of the methodology are:

- A generic approach capable of modelling and verifying sequential and branching behaviours.
- Refine requirements formally and integrate this formal specification with UML.
- Verifies and validates performance requirements in the Requirement Analysis phase itself.
- Explores all system behaviours exhaustively, ensuing robust design.
- Generates formal model automatically, promoting the use of formal methods in SDLC.
- Gels smoothly with the model-driven development process.
- Maintains consistency across the modelling phases.
- Enables modular approach while dealing with complex embedded system behaviour.

In general, model checking poses a state explosion problem if the system under modelling has many input variables or includes many components with numerous behaviours in parallel. But the methodology discussed here begins with the behavioural abstraction of the most critical system behaviour, and the selection is made based on Failure Mode Effect Analysis (FMEA) results of the system. The methodology chooses the most challenging system behaviour whose failure can cause catastrophic consequences during operation and performs a lightweight formal abstraction. It simplifies the process of formal verification by modelling selected safety-critical behaviour, for which model verification can fetch substantial benefits in proving reliable system operation well ahead of implementation. Overall, the study offers an absolutely feasible formal verification integrated visual modelling methodology for embedded system development.

6. CONCLUSION

In this study, the two case studies discoursed underscore the suitability of the approach in modelling and verification of embedded software, irrespective of the temporal nature of modelling requirements, sequential or branching time. Formal modelling, along with CTL, LTL property verification, are inherent in the methodology through the procedure. This characteristic feature makes it a unified Temporal Logic-based methodology, which is practicable in the model-driven development process. The conclusions drawn from the present study are given:

- The study presents a new methodology, with hands-on application of formal methods in the UML model-driven software development process. It is achieved through automation. We developed a UML-LTS Translator application that automatically generates Transition Systems from UML Statecharts. The modeller can verify his/her models before coding, residing in the UML domain itself, because the application accepts UML Statecharts and State Transition Matrix as inputs.
- Formal verification of performance requirements is an integral part of the methodology, making it suitable for property validation in the safety-critical system development cycle. Besides, the generic nature of methodology, addressing sequential and branching property specifications, increases the scope of its application in a broader range of systems. Thus, our methodology has great potential for use in quality concerned software engineering and model-based software development process. This study, along with the analysis results, finds widespread applications and can readily be applied for verification and validation of performance requirements of safety-critical software in the requirement analysis phase itself.

7. FUTURE WORK

The lightweight formalism integrated formal verification methodology can be proved to be thorough once it is implemented to the full appreciation of the targeted audience. It is intended to be used by the embedded software development community. To make things easier and promote the use of formal methods in software development projects, we are working on the complete automation of this methodology. We have already developed a UML-LTS Translator, which can interpret UML Statecharts and generate Transition Systems automatically. A GUI for this Translator and automation of model checking procedure and TL property specification will

lead to a generic tool that can formally verify UML models. It will execute with UML as a rear engine and be accessible to the modeller through GUI. Customised menus and toolbar options will be designed and included in UML GUI for this purpose.

REFERENCES

1. Bruce, Powel, Douglass. Real-Time UML: Developing Efficient Objects for Embedded Systems, 2003.
2. Luciana, Brasil, Rebelo, dos, Santos., Eduardo, Rohde, Eras.; Valdivino, Alexandre, de, Santiago, J´unior & Nandamudi, Lankalapalli, Vijaykumar, A. Formal Verification Tool for UML Behavioral Diagrams, ICCSA 2014. Part I LNCS. (8579), 2014, 696–711 .
3. Lilius, J & Paltor, I. P. Formalising UML State Machines for Model Checking. Proceedings of UML'1999, Lecture Notes in Computer Science. Springer-Verlag, 1999. doi: 10.1007/3-540-46852-8_31
4. Lilius, J & Paltor, I. P. vUML: A tool for verifying UMLmodels. Proceedings of 14th IEEE International Conference on Automated Software Engineering. IEEE, 1999.
5. Holzmann, G. The model checker SPIN. *IEEE Transactions on Software Engineering*, 1997. doi: 10.1109/32.588521
6. Latella, D.; Majzik, I & Massink., M. Automatic verification of a behavioral subset of UML statechart diagrams using the SPIN model-checker. *Formal Aspects of Computing*, 1999, **11**.
7. Kwon, Gihwon. Rewrite rules and operational semantics for model checking UML statecharts. Proceedings of UML'2000, Lecture Notes in Computer Science, 1939, 2000.
8. Alexander, David.; Oliver, Moller, M & Wang, Yi. Formal verification of UML statecharts with Real-Time extensions. *Fundamental Approaches to Software Engineering*, LNCS, 2002. doi: 10.1007/3-540-45923-5_15
9. Zhang, S. J & Liu, Y. An Automatic Approach to Model Checking UML State Machines. 2010 Fourth International Conference on Secure Software Integration and Reliability Improvement Companion, Singapore, 2010. doi: 10.1109/SSIRI-C.2010.11.
10. Zamira, Daw.; John, Mangino & Rance, Cleveland. UML-VT: A Formal Verification Environment for UML Activity Diagrams. *In* International Conference on Model Driven Engineering, 2015.
11. Eshuis, R. Symbolic model checking of UML activity diagrams. *ACM Transactions on Software Engineering and Methodology*, 2006. doi: 10.1145/1125808.1125809
12. Santos, L. B. R.; Jr., Santiago, V & A. Vijaykumar, Nandamudi, Lankalapalli. Transformation of UML Behavioral Diagrams to Support Software Model Checking, EPTCS 2014. doi: 10.4204/EPTCS.147.10
13. Kochaleema, K. H & Santhoshkumar, G. Methodology for Integrating Computational Tree Logic Model Checking in Unified Modelling Language Artifacts: A Case Study of an Embedded Controller. *Def. Sci. J.*. 2019, **69**(1), 58-64. doi: 10.14429/dsj.69.12294
14. Vieri, Del, Bianco.; Luigi, Lavazza & Marco, Mauri. Model checking UML specifications of real time software. *In* Proceedings of IEEE International Conference Complex Computer Systems, 2002. doi: 10.1109/ICECCS.2002.1181513
15. Jozef, Hooman.; Hillel, Kugler.; Julian, Ober, I.; Anjelika, Votintseva & Yuri, Yushtein. Supporting UML-based development of embedded systems by formal techniques. *Softw. Syst. Model.,* 2008. doi: 10.1007/s10270-006-0043-7
16. Clarke, E. M.; Emerson, E. A & Sistla, A.P. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Programming Languages Syst.,* 1986, **8**(2), 244-263.
17. Zohar, Manna & Amir, Pnueli. The Temporal Logic of Reactive and Concurrent Systems, Specification. *Springer-Verlag.* 1992. doi: 10.1007/978-1-4612-0931-7
18. Edmund, M. Clarke, Jr.; Orna, Grumberg & Doron, A. Peled. Model Checking. 1999. ISBN: 9780262032704
19. Object Management Group, UML Specification 1.5, available at http://www.omg.org/uml 2003. doi: 10.1016/j.entcs.2004.04.008
20. Hamilton, R. K. Miles. Learning UML 2.0, 1st Edition, O'Reilly Media, Sebastopol. 2006. Print ISBN-13: 978-0-59-600982-3
21. Berardi, Daniela.; Calvanese, Diego & De, Giacomo, Giuseppe. Reasoning on UML class diagrams. *Artificial Intelligence*, 2005. doi: 10.1016/j.artint.2005.05.0
22. Beato, Encarnacion, Ma.; Solorzano, Manuel, Barrio.; Cuesta, Carlos, E & Fuente, Pablo, de, la. UML Automatic Verification Tool with Formal Methods. *Electronic Notes Theoretical Comput. Sci.,* 2005, **127**, 3–16.
23. Knapp, A & Merz, S. Model checking and code generation for UML state machines and collaborations. In Proceedings of 5th Workshop on Tools for System Design and Verification, Technical Report. 2002.
24. Hooman, Jozef.; Kugler, Hillel.; Ober, Iulian & Votintseva, Anjelika., Yushtein, Yuri. Supporting UML-based development of embedded systems by formal techniques. *Softw. Syst. Model.,* 2008. doi: 10.1007/s10270-006-043-7
25. Lucas, Francisco J.; Molina, Fernando & Toval, Ambrosio. A systematic review of UML model consistency management. *Info. Software Technol*., 2009. doi: 10.1016/j.infsof.2009.04.009
26. Edmund, M. Clarke.; Allen, E. Emerson & Joseph, Sifakis. Model Checking: Algorithmic Verification and Debugging. *Communications of ACM.*, 2009. doi: 10.1145/1592761.1592781
27. Lima, V.; Talhi, C.; Mouheb, D.; Debbabi, M & Wang, L. Formal Verification and Validation of UML 2.0 Sequence Diagrams using Source and Destination of Messages. *Electronic Notes in Theoretical Comput. Sci.,* 2009. doi: 10.1016/j.entcs.2009.09.064

28. Luay, Alawneh.; Mourad, Debbabi.; Fawzi, Hassaıne.; Yosr, Jarraya & Andrei Soeanu. A Unified Approach for Verification and Validation of Systems and Software Engineering Models. Proceedings of the 13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems, ECBS'06. 2006. doi: 10.1109/ECBS.2006.17
29. Nawal, Addouche.; Christian, Antoine & Jacky, Montmain. Methodology for UML Modeling and Formal Verification of Real-Time Systems, International Conference on Computational Intelligence for Modelling Control and Automation and International Conference on Intelligent Agents,Web Technologies and Internet Commerce (CIMCA-IAWTIC'06). 2006. doi: 10.1109/CIMCA.2006.144
30. Gnesi, S.; Latella, D & Massink, M. Model checking UML statechart diagrams using JACK. *In* Proceeding of 4th IEEE International Symposium on High-Assurance Systems Engineering, (HASE'99). 1999. doi: 10.1109/HASE.1999.809474.
31. Gabor, Madl, Sherif, Abdelwahed & Douglas, C. Schmidt. Verifying Distributed Real-time Properties of Embedded Systems via Graph Transformations and Model Checking. Institute for Software Integrated Systems, Vanderbilt University, Nashville/ Center for Embedded Computer Systems, University of California, CA. 2006 doi: 10.1007/s11241-006-6883-y
32. Doron, Drusinsky. Modeling and Verification Using UML Statecharts: A Working Guide to Reactive System Design, Runtime Monitoring and Execution-based Model Checking. 2006. ISBN: 0750679492
33. Beato, M. E.; Barrio-Solórzano, M.; Cuesta, C.E & Fuente, de, la, P. UML automatic verification tool with formal methods. *Electronic Notes in Theoretical Comput. Sci.,* 2005. doi: 10.1016/j.entcs.2004.10.024
34. Mohammed, Misbhauddin & Mohammed, Alshayeb. UML model refactoring: a systematic literature review. *Empirical Software Engineering,* 2015 doi: 10.1007/s10664-013-9283-7
35. Alexander, Knapp & Till, Mossakowski. Multi-view Consistency in UML: A Survey. Lecture Notes in Computer Science book series (LNCS). 10800, 2018 doi: 10.1007/978-3-319-75396-6_3

CONTRIBUTORS

Ms K.H. Kochaleema received her MTech in Software Engineering from Cochin University of Science and Technology, Kochi, Kerala. Currently working as Scientist G in Naval Physical and Oceanographic Laboratory, Defence Research and Development Organisation, Kochi. She is heading Quality and Reliability group of NPOL. She has thirty years of experience in the field of design, development, verification and validation of embedded systems for sonar systems for various platforms like, ship, submarine and naval helicopters.
In the present work, she is responsible for the methodology proposal, identification of a suitable application for case study and validation of the proposed methodology using the same.

Prof. G. Santhoh Kumar received his MTech in Computer and Information Science and PhD in Wireless Sensor Networks from Cochin University of Science and Technology (CUSAT), Kochi, Kerala. Currently, working as Professor and Head of the Department of Computer Science in the Faculty of Technology of CUSAT. He has seventeen years of teaching and research experience. His areas of interest include, formal modelling and verification, computer vision and artificial intelligence.
In the current work, he has provided suitable guidance in problem formulation and offered necessary direction and overall support to carry out this study successfully.