Defence Science Journal, Vol. 69, No. 1, January 2019, pp. 58-64, DOI : 10.14429/dsj.69.12294

Methodology for Integrating Computational Tree Logic Model Checking in Unified Modelling Language Artefacts: A Case Study of an Embedded Controller

K.H. Kochaleema[*] and G. Santhoshkumar[#]

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

ABSTRACT

A unified modelling language (UML) based formal verification methodology that can be easily integrated into an embedded system software development life cycle is suggested. The approach augments UML diagrams with formal models through an interfacing domain and adds semantics to these diagrams. The suggested methodology; commences from functional specification and use case modelling, selects the most critical behaviour where formal verification can add value to the development cycle, analyses the selected behaviour using UML state transition diagram, derives a state chart matrix from the same, and a high level language software translates the state chart matrix to a labelled transition system. Safety properties are derived from system specifications and are expressed as computation tree logic (CTL) formulae. CTL model-checking algorithm from the literature is used for model-checking. The applicability of the suggested approach is established using a safety critical embedded controller used for deployment and recovery of sensor structures from an airborne platform.

Keywords: Unified modelling language; State chart diagram; State chart matrix; Safety property specification; Computational tree logic; Formal verification

1. INTRODUCTION

Verifying safety-critical embedded systems with unyielding timeliness requirements is challenging. Apart from mundane factors - size, design-complexity, unforeseen environmental interactions, hardware-restrictions etc., there are constraints for functional verification, particularly in sea-faring defence systems. Simulation-based verification is insufficient and sometimes impractical in these cases[1]. Even though formal methods are best suited for better understanding and verification of complex software requirements, the usage of formal methods in model-based software development is minimal. Majority of visual modelling community are reluctant to adopt these methods, however complex and critical the software being developed. It is broadly agreed that practice of formal methods in software engineering is essential, while there are several systems ascertaining the applicability of formal methods in industrial applications presenting very good results. But, embedded designers are still reluctant to adopt formal methods, either due to the complex nature of formal notations or the lack of awareness of the quality attributes they provide in delivering robust systems.

The paper proposes a model-checking based comprehensive system-level design methodology, facilitating the use of the formal models in the development process, adding confidence for realising correct and reliable systems. It integrates formal methods into unified modelling language (UML) such that the embedded system specification expressed in natural language becomes mathematically verifiable, early in the development cycle.

An airborne sensor array management system (SAMS) is considered as a characteristic system for methodology demonstration. SAMS is a safety-critical multidisciplinary system carried on-board aircrafts and deployed at interest marine locations. It enables smooth and safe sensor structure deployment and retrieval in definite time-limits. The diverse interface requirements and complex behavioural requirements reveal the limitation of simulation-based testing and calls for formal model generation and verification semantics thereafter.

2. RELATED WORK

Unified modelling language is the standard visual modelling tool used for modelling complex software systems[23]. Verification and validation of such systems' UML diagrams advance error detection to early development phases, resulting in fail-safe and reliable operation. Integration of formal methods with UML diagrams adds semantics to UML diagrams, enabling formal verification and validation during software development life cycle.

Existing literature focusses on integrating formalism in UML diagrams. They mainly focus on UML class diagrams, sequence diagrams, activity diagrams[2] etc. and translate these to formal specification language compatible with an existing

Received : 06 November 2017, Revised : 12 December 2018
Accepted : 18 December 2018, Online published : 10 January 2019

model-checking tool. Object constraint language (OCL) is established as the formal language for specification of properties of object structures in UML models[1]. IBM and OMG provide tool support, for verification of class diagrams and state-chart diagrams. Research has been published in integrating communicating sequential processes (CSP) into UML state-charts, class and sequence diagrams, used in safety-critical system-design. Prototype verification system (PVS) is the specification and verification language for UML class diagrams and OCL constraints[3], including use-case diagrams[24]. PROMELA, SPIN, UPPAAL, rCOS etc. are used in model-checking of various UML diagrams[25].

Consistency issues across modelling phases or abstraction levels are studied only by a few, therefore inconsistency problems across modelling phases are considered promising[8]. The research here, focusses on evolving a practitioner-friendly, but powerfull modelling methodology that can be easily integrated with existing methods.

3. PROPOSED MODELLING METHODOLOGY

3.1 Lightweight Formalism Integrated Model Checking Approach

UML is a semiformal modelling language with graphical notations for expressing different views of the system from different viewpoints. The modelling power of UML's modelling artefacts is augmented with formal semantics[5]. The approach integrates UML-based visual abstraction models with formal method based finite automata. It smoothly couples UML visual models and formal methods, synergising the benefits of visual modelling and formal techniques.

The method is named light weight as it suggests application in areas where formal specification accomplishes specific quality objectives and skips areas where they aren't appropriate. The sequence of actions involved in the proposed integrated model-checking approach is as shown in Fig. 1. There are three distinct domains, UML domain, Interface and Formal domain. The inventive part of the proposal lies in the Interface, domain with which the UML and formal domains are coupled.

The phases involved in the proposed methodology are shown as major blocks in the diagram. It commences with system specification in natural language and proceeds through Use case modelling and state chart transitions. The interfacing software links visual domain to formal model. The model checking algorithm runs the model through various runs and decides the entailment of property specification in the model. The steps in the proposed methodology are as follows.

3.1.1 Requirement Analysis of the Real-time System

The approach begins with a use case model depicting the functional capabilities of the embedded system[26,28]. In subsequent modelling abstractions, these capabilities manifest themselves as the behaviour of the system or interactions among the system and the environment[7].

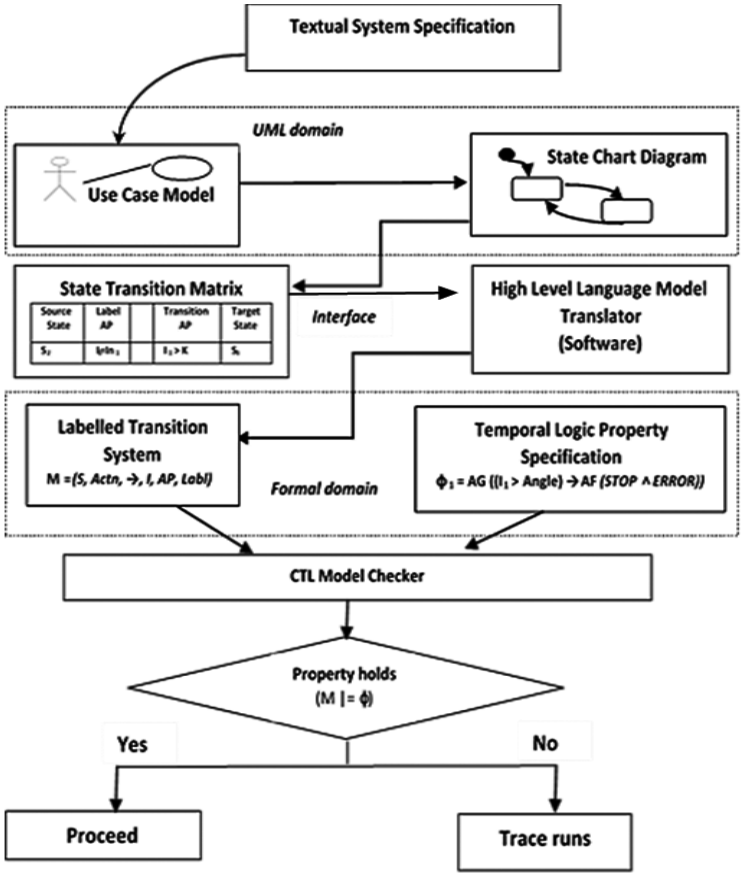

Figure 1. Integrated model-checking workflow.

3.1.2 Generation of Detailed Specification of the use Case using State Chart Diagram

This phase ensures the behavioural abstraction of the use case scenarios using UML state chart diagrams[26,28].

3.1.3 Generation of State Transition Matrix and Conversion into Labelled Transition System

This phase encompasses two major steps.

(i) *Generation of State Transition Matrix* - The state transition matrix (STM) tabulates the dynamics of the operation (events) and the target state. The sample STM is as shown in Table 1. The label arithmetic proposition (AP), holds true in a particular state. The transition AP leads to state change from Source to Target State.

Table 1. Sample state transition matrix

Source state	Label AP	Transition AP	Target state
S_2	$I_1=In_1$	$I_1> K$	S_3

The tabulation of these transitions is done manually by the modeller, and is repeated until all transition paths are covered.

(ii) *Auto conversion of the State Transition Matrix to Labelled Transition System*- A high level language program is developed to generate the labelled transition system (LTS) from STM. The program accepts STM as input and generates the formal model, LTS. LTS can be defined by the tuple,

LTS = (*S, Actn, →, I, AP, Labl*)

where *S* is the set of states, *Actn* is a set of actions, → ⊆ S × Actn× S is a transition relation, *I*⊆ S is a set of initial states, *AP* is a set of atomic propositions, *Labl* is a labelling function.

Here there is direct relationship between the UML model and the generated formal model. The events in the state-charts are mapped to transitions in LTS and the actions in the states in state-chart diagram appear as labels in resulting LTS states. This mapping ensures consistency between the two modelling domains. The commonality of expressions in these diagrams, gives a feel of UML modelling, even when the user is in the formal modelling realm.

3.1.4 Formal Property Specification

Real time systems are designed to function continuously and the perception of time-ordering of events is essential in modelling them. Pnueli put forward the idea of temporal logics in 1970s, describing the temporal ordering of events for modelling concurrent and real time systems. Temporal logics[27] describe the temporal ordering of events without explicitly mentioning time.

There are two popular temporal logics, liner temporal logic (LTL) and computational tree logic (CTL). LTL is based on sequential time lines and every instant has a unique possible successor. CTL is reasoned over a branching timeline and each time instant can get into many possible successors[27-30].

LTL models system behaviour as an infinite sequence of states and each state has a unique successor, based on a linear-time perspective. LTL abstracts time as a discrete entity and is characterised by distinct points[12]. But, there could be system behaviours, which splits into sub behaviours based on events and conditions. For modelling such behaviours, properties that state the existence of a path have to be specified. LTL provides only global quantifiers, and is inadequate while dealing with properties that mix existential and universal path quantifiers.

CTL addresses the above problems by introducing path quantifiers. The path quantifiers indicate whether a given formula applies to all possible paths from a given state or only some possible paths, using:

A – for every path

E – there exists a path

Most safety-critical systems are real time in nature and interact asynchronously with the surrounding medium and sensors. The behaviour of these systems will be decisive in nature. Under such circumstances, branching time logic is appropriate for the behavioural specification. The method suggested here makes use of CTL formulae for property specification.

3.1.5 Formal Verification

The CTL model-checking algorithm is used for formal verification. The LTS model M generated by the high level language translator software is checked for the entailment of CTL property specification.

Model-checking is a widely accepted tool for automatic verification of both hardware and software systems. It's a procedure that checks whether a given structure M is a model of the logic formula ϕ[24,25]. M is an automata-like structure representing the system and ϕ, a temporal logic formula expressing the system's desirable properties. The model checker verifies whether M satisfies the property during execution[13]. Here the advantage is that the checking is exhaustive compared to system-testing using multiple scenarios. Moreover model-checking is carried out, well before system implementation and thus streamlines system debugging and regression testing.

The inputs to a model checker are the system description expressed as a finite state system and a few performance specifications, i.e. property specifications, expressed as temporal logic formulae[18-19]. The model checker runs an algorithm and verifies that either the properties hold during model execution, or confirms with a counter example that the property is violated during the model-run. The counter example generated provides insight into design errors overlooked at this stage.

4. CASE STUDY

The proposed methodology is substantiated by applying the same in modelling, an embedded controller used in a safety critical military system.

4.1 System Description

A sensor array management system (SAMS) for military applications is chosen as the typical case for modelling and verification purpose. SAMS is used in the context of dunking sensor systems for acoustic data acquisition and processing onboard military platforms. It operates as an airborne sensor array deployment/retrieval mechanism, using which sensor structures can be deployed from naval platforms. The major subsystems of SAMS are as shown in Fig. 2.

There is a user interface and an embedded sensor deployment controller (SDC) in the system which initiates commands for positioning the wet end sensor array to desired sea depth. SDC shall read the lower command from the operator, initiated through the user interface. It shall also read various winch sensors periodically, and using these values it generates safety interlocks for ensuring the safety of the Winch

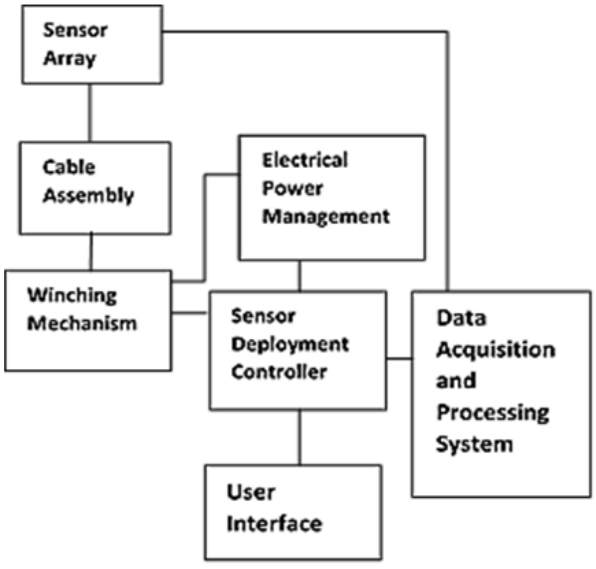

Figure 2. Sensor array management system.

and sensor array during operation. The major performance requirements of SAMS are as follows

- The deployment and retrieval shall take minimum time
- It shall have the facility to monitor the various mechanical sensors (speed, cable tension, cable drift)
- Jerks shall be avoided during deploying/hoisting
- There should be redundancy/exception handling for retrieving the sensor, in case of any failure.

Undesired behaviour of SDC is to be avoided by design itself, mainly because operational failures result in loss/damage of SA, along with instigation of unsafe hovering positions of aircraft, endangering lives of on-board crew.

4.2 Light Weight Formalism Integrated Model Checking Approach

4.2.1 Analysis of the Real-time System Modelling Operational Requirements

The major operational capabilities drawn from the above system requirements are secure lowering and hoisting of the sensor array. Lowering operation encompasses four use cases, as shown in Fig. 3.

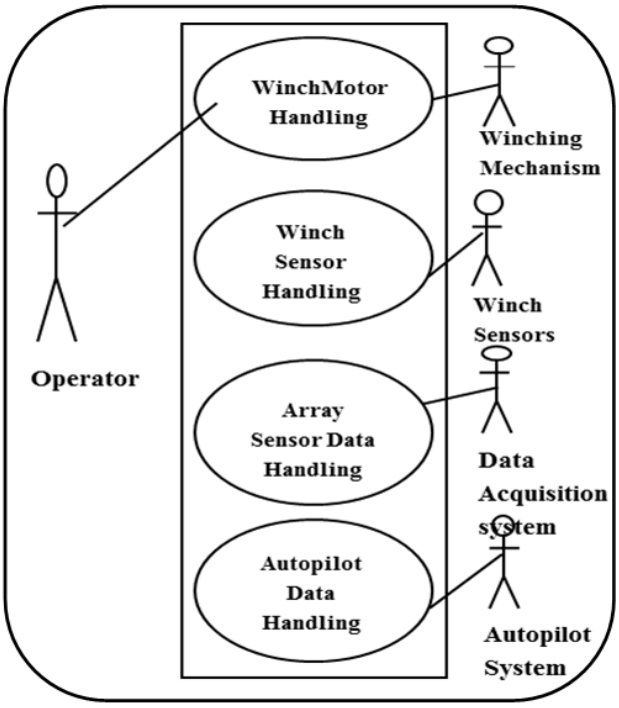

Figure 3. Sensor deployment use cases.

4.2.2 Generate a Detailed Specification of the use Case using State Chart Diagram

There are four different parallel sequence of actions take place in the lowering mode of operation (Fig. 4). They are:

-WinchMotorHandling
- WichSensorHandling
- ArraySensorDataHandling
- AutoPilotDataHandling

Amongst these, WinchSensorHandling sub behaviour alone is considered for demonstration purpose. The independent sequences of processing that take place upon reception of various events are as shown in Fig. 5.

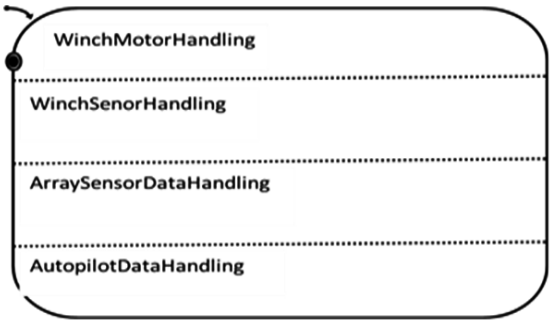

Figure 4. Sensor deployment state-chart representation.

4.2.3 Formal Model Generation -Translation of UML State Chart to Labelled Transition System

The major difficulty in generating the formal model of a UML based visual model is the translation of the UML diagram to a formal notation comprehensible by the model-checking algorithm. The research work provides a hands-on method for this translation.

Step1:

UML State-chart to STM translation

Step2:

STM to LTS translation

The state-chart in Fig. 5 is mapped into STM as shown in Table 2.

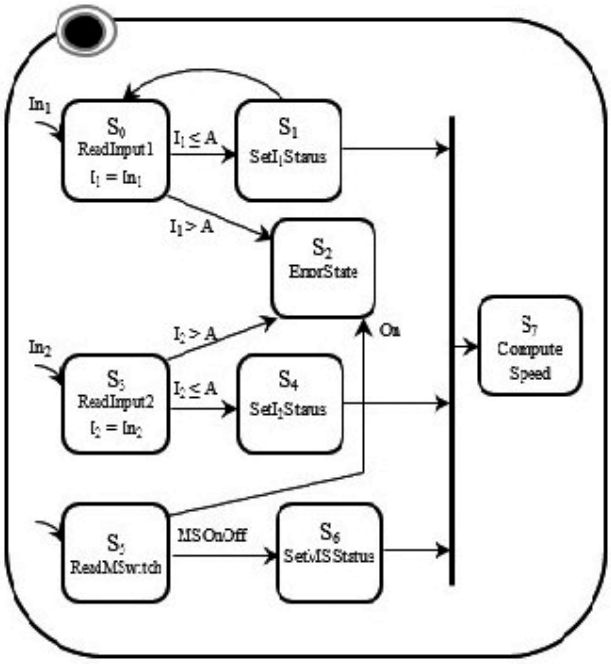

Figure 5. WinchSensorHandling state chart –exploded view.

Secondly, the STM is converted into LTS, using high level language translator (HLLT) software and is the most inventive step in the approach. HLLT scans the input STM and generates the corresponding LTS. HLLT facilitates the automatic generation of the formal model and serves as the interface for linking UML domain and formal domain, smoothly. The output of the program appears as shown in Fig. 6.

Table 2. State transition matrix

Source state	Label AP	Transition AP	Target state
S_0	I_1=In_1	I_1<=Angle	S_1
S_0	I_1=In_1	I_1>Angle	S_2
S_1	I_1Valid=TRUE	I_1Valid=TRUE I_2Valid=TRUE I_3Status=OFF	S_7
S_1	I_1Valid=TRUE	Timer	S_0
S_2	I_1Valid=FALSE	Timer	S_0
S_3	I_2=In_2	I_2>Limit	S_2
S_3	I_2=In_2	I_2<=Limit	S_4
S_4	I_2Valid=TRUE	I_1Valid=TRUE I_2Valid=TRUE I_3Status=OFF	S_7
S_4	I_2Valid=TRUE	Timer	S_3
S_2	I_2Valid=FALSE	Timer	S_3
S_5	MSStatus=ON	MSStatus=ON	S_2
S_5	MSStatus=OFF	MSStatus=OFF	S_6
S_6	I_3Status=OFF	I_1Valid=TRUE I_2Valid=TRUE I_3Status=OFF	S_7
S_6	I_3Status=OFF	Timer	S_5
S_2	I_3Status=ON	Timer	S_5

4.2.4 Formal Property Specification

CTL is used for formalising the property to be verified in the above model. The arithmetic propositions involved are:

AP = {I_1> Angle, I_2> Limit, I_3 Status = OFF, STOP, ERROR)

The safety specification states that whenever I_1 sensor values are beyond limits, lowering shall be suspended and error be indicated to the operator. This is treated as the safety property of the system, as it is a bad behaviour which shall never be exhibited during operation. Thus, the system's informal performance requirement is translated into formal CTL formula, using the above set of arithmetic propositions.

SafetyProperty1 (ϕ_1)

If the I_1sensor value> Angle defined, eventually Lowering is suspended and Error is indicated to the Operator.

ϕ_1 = AG((I_1>Angle) →AF *(STOP ∧ ERROR))*

Similarly for the second safety requirement based on I_2 sensor.

SafetyProperty2 (ϕ_2)

If the I_2 sensor value > Limit Specified, eventually Lowering is suspended and Error is indicated to the Operator.

ϕ_1 = AG((I_2> Limit) →AF *(STOP ∧ ERROR))*

4.2.5 Formal Verification

The formal model is generated and the property specification is available in formal language. The next objective is to verify that the model generated satisfies the safety properties during various runs of the system.

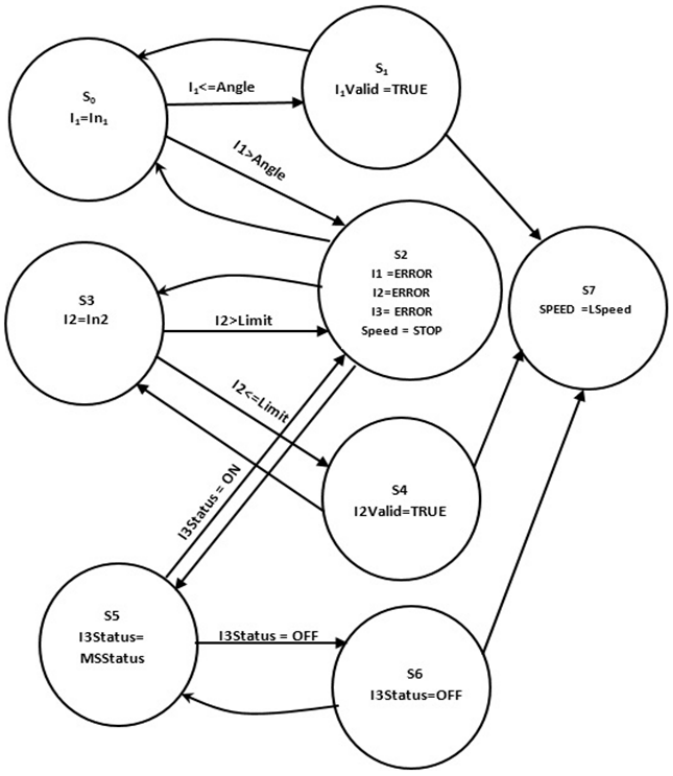

Figure 6. WinchSensorHandling formal model – labelled transition system.

Model-checking

In this particular case-study, the LTS generated from the UML state diagram is the formal model M and ϕ_1, ϕ_2 form the formal property specification. The prevalent CTL model-checking algorithm is used as the model checker. It verifies if model executions of the initial states s of M satisfy the CTL formula ($M \models \phi$).

Model-checking Algorithm

1. *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)*
2. *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 Running

The sub formula [(STOP ∧ ERROR) holds in state S_2
1. (*STOP ∧ ERROR)*]; [S_2]
2. [*AF* (*STOP ∧ ERROR*)] [S_0, S_2]
3. [(I_1> *Angle*)] [S_0, S_2]
4. [(*Drift*>*Angle*)→*AF*(*STOP*∧*ERROR*)][S_0, S_2]
5. [*AG((Drift*>*Angle*)→*AF(STOP ∧ ERROR))*] [S_0, S_2]

Initial state S_0 is a subset of denotation of ϕ, $[\phi]$. This proves that the model satisfies the safety property ϕ_1. Once proven that the model satisfies the desired properties, a direct translation of the model into implementation ensures reliable system operation in field.

5. CONSISTENCY ACROSS MODELLING PHASES

A continuous symbols trace exists throughout the modelling phases, in this approach. It starts from the use case names in the use case model. Secondly, these names label the state-chart diagram. The same are used in generating STM. The HLLT software uses this matrix as input and generates a formal model in terms of the same symbols and expressions appearing in STM. The performance requirements of the system expressed in natural language is the basis for generating the safety properties. The arithmetic expressions generated from these are mapped into CTL property specifications.

6. BENEFITS

The approach's major gains are as follows

i. *Consistent abstraction across modelling phases:* Direct mapping exists between the UML model and the generated formal model, by way of events(instate-charts) to transitions (in LTS) and actions (in state-charts) to labels (in LTS).

ii. *Smooth progression from informal to formal domain*: This transition phase is made automatic for the modeller, he is freed from acquiring knowledge and skill in formal modelling.

iii. *Easy blending with model driven development process:* Easily integrated into a UML based model driven development process.

iv. *Modular and flexible abstraction of complex embedded system behaviour:* Allows selection of critical behaviour for abstraction and modelling. Modeller can adopt it for modelling complex behaviours wherein he can exhaustively analyse the various system states under different input conditions. This built-in modularity addresses the tediousness in handling compound systems with numerous states.

7. LIMITATIONS

The entire approach is dependent on STM generated from UML state-charts. If the modeller makes mistakes, or omits states during this phase, the subsequent steps will be incorrect.

8. CONCLUSIONS AND FUTURE WORK

UML is an accepted visual modelling language with precise notations and expressive power to handle complex software systems. The methodology illustrated, integrates formalism into UML diagrams, by way of STM. This is simple and comprehensive for adoption in embedded software development life cycle. Generation of exhaustive STM is significant in this approach and is a modeller driven activity. This could be an overhead, but is simple and straight forward as compared to existing formal methods.

The state transition diagrams, in the behavioural abstraction phase, translated into a tree like structure can be traversed for reachability. If a path exists from the root to destined nodes, its treated as a valid trace/run and leads to behavioural property verification during model-checking. This is a proposal with research potential.

REFERENCES

1. Cabota, J.; Clarisób, R. &Rierab, D. On the verification of UML/OCL class diagrams using constraint programming. *J. Sys. Software,* 2014, **93**, pp. 1–23. doi:10.1016/j.jss.2014.03.023
2. Berardi, Daniela.; Calvanese, Diego. & De Giacomo, Giuseppe. Reasoning on UML class diagrams. *Artificial Intelligence*, 2005, **168**(1-2), 70-118. doi: 10.1016/j.artint.2005.05.003
3. Holzmann, G. The model checker SPIN. *IEEE Transactions on Software Engineering*, 1997, **23**(5). doi: 10.1109/32.588521
4. Mota, E.; Clarke, E.M.; Groce, A.; Oliveira, W.; Falcão, M. & Kanda, J. VeriAgent: an approach to integrating UML and formal verification tools. *Electron. Notes Theor. Comput. Sci.*, 2004, **95**, 111–129.
5. Hooman, Jozef; Kugler, Hillel; Ober, Iulian; Votintseva, Anjelika & Yushtein, Yuri. Supporting UML-based development of embedded systems by formal techniques. *Software Sys. Model*, 2008, **7**, 131–155 doi: 10.1007/s10270-006-0043-7
6. Lucas, Francisco J.; Molina, Fernando &Toval,Ambrosio. A systematic review of UML model consistency management. *Info. Software Technol.*, 2009, **51**, 1631–1645. doi:10.1016/j.infsof.2009.04.009
7. Clarke, Edmund M.; Emerson, Allen E. & Sifakis, Joseph. Model Checking: Algorithmic Verification and Debugging. *Commun. ACM*. 2009, **52**. doi:10.1145/1592761.1592781
8. Harel, David. Statecharts: A visual formalism for complex systems. *Sci. Comput. Programming,* 1987, **8**, 231-274.
9. Harel, D. & Gery, E. Executable object modeling with statecharts. *IEEE Computer*, 1997, **30**, 31–42.
10. Lima, V.; Talhi, C.; Mouheb, D.; M. Debbabi, M. & Wang, L. Formal verification and validation of UML 2.0 sequence diagrams using source and destination of messages. electronic notes in theoretical computer science, 2009, 254, pp. 143–160. doi: 10.1016/j.entcs.2009.09.064
11. Alawneh, L.; Debbabi, M.; Hassaıne, F.; Jarraya, Y. & Soeanu, A. A unified approach for verification and validation of systems and software engineering models. *In* Proceedings of the 13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems (ECBS'06). doi: 10.1109/ECBS.2006.17
12. Linzhang, W.; Jiesong, Y.; Xiaofeng, Yu.; Jun, Hu.; Xuandong, Li & Guoliang, Z. Generating test cases from UML activity diagram based on gray-box method. *In* Proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC'04). 1530-1362/04
13. Clarke, Duncan & Lee, Insup. Testing real-time constraints in a process algebraic setting. *In* Proceedings of the 17th International Conference on Software Engineering(ICSE'95). pp. 51-60. doi: 10.1145/225014.225019
14. Van, Marcel F.; Christian, Amstel F.J.; Michel, Lange &

Chaudron, R.V. Four automated approaches to analyze the quality of UML sequence diagrams. *In* 31st Annual International Computer Software and Applications Conference (COMPSAC 2007).
doi: 10.1109/COMPSAC.2007.6

15. Addouche, N.; Antoine, C. & Montmain, J. Methodology for UML modeling and formal verification of real-time systems. *In* International Conference on Computational Intelligence for Modelling Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06).
doi: 10.1109/CIMCA.2006.144
16. 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, 1999 (HASE'99), pp. 46-55. ISBN:0-7695-0418-3
17. Alur, R. & Henzinger, Thomas A. Logics and models of real time: A survey. AT&T Bell Laboratories, Murray Hill, NJ / Computer Science Department, Cornell University, Ithaca, NY 14853. https://hdl.handle.net/1813/7102
18. Frits, Juho. Model checking embedded control software. TKK Reports in Information and Computer Science, Espoo 2010. http://lib.tkk.fi/Reports/2010/isbn9789526031033.pdf.
19. Lee, David. & Yannakakis, Mihalis. Principles and methods of testing finite state machines - A Survey. AT &T Bell Laboratories Murray Hill, New Jersey.
20. Formal methods specification and analysis guidebook for the verification of software and computer systems. Vol. ii, a practitioner's companion. NASA-GB-001-97.
21. Tawhid, Md Bin Waez.; Dingel, Juergen & Rudie, Karen. Timed automata for the development of real-time systems, Technical Report 2011-579
22. Sgroi, M.; Lavagno, L. & Alberto, Sangiovanni-Vincentelli. Formal models for embedded system design. University of California, Berkeley/Università di Udine, Italy
23. Gagnon, Patrice.; Mokhati, Farid & Badri, Mourad. Applying model checking to concurrent UML models. University of Québec at Trois-Rivières, Canada/University of Oum-El-Bouaghi, Algeria/University of Québec at Trois-Rivières, Canada.
24. Madl, Gabor.; Abdelwahed, Sherif. & Schmidt, Douglas C. 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.
doi: 10.1007/s11241-006-6883-y
25. Alhumaidan, F. State based static and dynamic formal analysis of UML state diagrams. College of Computer Sciences and Information Technology, King Faisal University, Hofuf, KSA- May 2012.
doi: 10.4236/jsea.2012.57056
26. Manna, Z. & Pnueli, A. The temporal logic of reactive and concurrent systems, specification. Springer-Verlag, 1992.
doi: 10.1007/978-1-4612-0931-7
27. Baier, Christel. Principles of model checking. ISBN: 978-0-262-02649-9
28. Clarke, Edmund M. Model checking. ISBN: 9780262038836
29. Douglass, Bruce Powel. Real-time UML: Developing efficient objects for embedded systems. 3rd Edn, 2003.
30. Hubris, H. & Frappier, M. Software specification methods, iSTE, 2006. ISBN: 9781905209347
31. Miles, R. & Hamilton, K. Learning UML 2.0, O'Reilly Media, Sebastopol, 2006, ISBN-13: 978-0-59-600982-3
32. Booch, G.; Rumbaugh, J. & Jacobson, I. The unified modeling language user guide. Addison Wesley, 1999.. ISBN: 0-201-57168-4
33. Drusinsky, Doron. Modeling and verification using UML statecharts: A working guide to reactive system design, runtime monitoring and execution-based model checking, 2006.. ISBN: 0750679492
34. Object Management Group, UML Specification 1.5, http://www.omg.org/uml, 2003.
doi:10.1016/j.entcs.2004.04.008
35. Beato, M.E.; Barrio-Solórzano, M.; Cuesta, C.E. & Fuente, de la P. UML automatic verification tool with formal methods. *Electron. Notes Theoretical Comput. Sci.*, 2005, **127**, 3–16.
doi: 10.1016/j.entcs.2004.10.024

CONTRIBUTORS

Ms K.H. Kochaleema received her MSc and MTech in Software Engineering from Cochin University of Science and Technology, Kochi, Kerala. Currently working as Scientist G in DRDO-Naval Physical and Oceanographic Laboratory, Kochi. She is heading the Independent Verification and Validation Division of Systems Engineering Group. She has research experience in the field of design and development of embedded controllers of sonar systems.
In the present work, she is responsible for the methodology proposal, identification of a suitable application as case study.

Prof. G Santhoh Kumar received his MTech in Computer and Information Science and PhD in 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. His areas of interest include : Formal modelling and verification, computer vision and artificial intelligence.
In the current work, he has helped in problem formulation and provided overall necessary guidance and support to carry out this study successfully.