Penerbit UTHM

IJIE

Journal homepage: http://penerbit.uthm.edu.my/ojs/index.php/ijie
ISSN : 2229-838X e-ISSN : 2600-7916

The International Journal of Integrated Engineering

Verifying the Correctness of UML Statechart Outpatient Clinic Based on Common Modeling Language and SMV

Pathiah Abdul Samat[1], Muhammad Amsyar Azwarrudin[1], Norhayati Mohd Ali,[1] Novia Admodisastro[1]

[1]Dept. of Software Engineering and Information System, Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, 43400 Serdang, MALAYSIA

DOI: https://doi.org/10.30880/ijie.2021.13.05.015
Received 1 May 2021; Accepted 30 May 2021; Available online 31 July 2021

Abstract: Unified-modelling language (UML) is a standard general purpose modelling language, which is widely, used in system design of banking, biological, plantation and healthcare. Recently, there are many systems of healthcare are modeled using behavioral diagram such as UML statechart for design purposes. However, the behavior of healthcare statechart is rarely verified to ensure it is behaving as we needed. In software engineering, a software should be verified before it is transform to the further phases. In this paper, a statechart of outpatient clinic is verified to ensuring the correctness of its design. Therefore, to achieve our objective, we have applied Common Modeling Language (CML) and SMV model checker for verification formal system modeling and specification of property of statechart outpatient clinic. The result shows that the statechart of outpatient clinic is behave as required and the statechart is allowable to transform to the next phase.

Keywords: Formal verification, CML, model checking, temporal logic, UML statechart

1. Introduction

UML is a standardized general purpose modelling language, which uses graphical notations, such as statechart diagram [1]. Statechart diagram is use to model the dynamic behavior of a system [2], [3] and [4]. In recent years, there a numbers of health care system that are modelled using statechart to represent the dynamic behavior of its system. Example of the statechart of health care system discussed in [5] and [6]. However, the statechart model must verify to ensuring its correctness or in other word, the property of statechart model is satisfied. In software engineering, correctness is defined as the obedience to the specifications that stated how we depend on the software and how its reaction when it is employ with correctly. However, the correctness of software design tendency to failure, if specification of software is ambiguous. Based on [7], the behavior of statechart is analyze with respect to some correctness specification expressed by temporal logics formula. In this paper, we use CML [8] to modeled statechart to formal language of SMV [9] and express the properties in temporal logic as Computational Tree Logic (CTL) [10].

Software verification is important tasks in software model, if we not verify the software model, the bug might be present in the model and errors will occur in the source codes. As a result, the system tendency to failure. We often read of incidents where some malfunction caused from fault in hardware or software system. The most tragic example of such a fault is the destruction of the Ariane 5 rocket [11], due to floating-point overflow; one bug and one crash [12]. Based on Ariane 5 rocket tragedy, the need for trustworthy hardware and software system is critical. As increasing number of such systems are being used in our lives, it is important that their properties is properly verified. Practically, it is impossible to shut down a malfunctioning system in order to restore safety; where in reality we are very much dependent on such systems for both their continuous operation and proper functioning. The lesson learned from this tragedy is that the system or software must go through the process of verification and validation during its design for ensuring on their correctness. One of the automatic software verifications is model checking technique.

*Corresponding author: pathiah@upm.edu.my

The numerous researchers use model checking to tackle the correctness of statechart problem [13] and [14]. In health care system, there are a number of researcher apply model checking to verify the correctness of statechart [15], [16] and [17]. A framework has developed to verify the clinical guideline using UML statechart and SPIN model checker [15]. This framework verifies specific requirement in the guideline to check whether it is present semantic error and inconsistencies. Another researcher proposed to constructing the models in the Refinement Calculus of Object Systems (rCOS) and then verifying Trustable Medical System (TMS) in the real time the safety properties by tool UPPAAL [16]. They use rCOS to develop TMS because it supports both static structural and dynamic behavioral refinement of object-oriented system and can effectively reduce the complexity of system by separating concerns. The healthcare workflow also can be verify using Alloy specification [17]. This approach transforms the healthcare workflow metamodel to Alloy specification and verify using Alloy Analyzer.

This paper is organized as follows: The following section discuss the materials and methods of research. Then the results and discussion are present in section 3. Finally, the conclusion and direction for future research are discuss in section 4.

2. Material and Method

UML Statechart is valuable diagram, which is able to representing the behavioral of state machine. The most important, statechart can react to external environment of a system through events, which triggered by a transition. UML statecharts [18] are hierarchical automata associated with objects (class instances) to model their behavior. The valuable innovation of statechart can assist in modeling the life cycle of objects behavioral in graphically. In exact terms, statecharts, consisting of states and transitions, convey how objects behave through time because its surrounding can affect the reaction to an event. Statechart that was proposed by [19] is an extension of the finite state machine (FSM) which has features like concurrency, hierarchy, and communication.

Model checking is a popular automatic technique used to check the specification of finite state machine system [20],[21] and [22]. Model checking requires two input: formal modeling and formal specification. Both of the inputs must be written in language of model checker. The model checker will execute verification process, once both of input ready written in its language. Most of model checker tools produced either ‘yes’ to mean – model satisfied the specification or ‘no’ to mean – model not satisfied the specification.

2.1 Common Modeling Language

Common Modeling Language (CML) was propose by [23] as intermediate language to translate statechart model to input language of model checker.

Definition 1: Formally, Common Modeling Language is defined as CML=<S, S0, Sc, G, T, C, R, Root> where:

- S is a finite set of states, where each state, s is as one of the two state types: {AND, OR}
- S0 is a set of initial states (S0 S). S0 forms a valid initial transition relation.
- Sc is a set of states that forms a valid state configuration.
- G is a finite set of triggers
- T is a finite set of transition relation, T = S x G x S`.
- C: S, S is the state function. If s` $\subseteq$ C(s), then s` is an immediate descendant of s. The function C describes a component state of the model.
- R is a relation between components.

Translation from CML to I-SMV is define to describe the translation from CML to SMV's language.
Definition 2: Let I-SMV be the input language of SMV that consists of four tuples:

<M, V, N, Y>

In which:
M= A set of finite modules
V = A set of finite state variables
N = A set of next states
Y = Relation between one module to another

I-SMV represented as modular. The parent module is represented as main module. Another module represents as sub module. Module, M consists finite state variables, V. The next operator, N represent the evolving states from one to another state. A set parameter represents the relation, Y from one module to another module. A set of rules will use to represent the translation from CML to I-SMV.

There are many CML in a finite state machine. Each of CML represent as level, L. Therefore, level L, in CML is mapping to module M. The set of states, S and triggers, G corresponding to V as state variables, V. The transition, T

corresponding to N as next state. Lastly, the relation between levels, R corresponds to the relation between modules, Y. In this study, there are four rules that mapping from CML to I-SMV. The rules from CML to I-SMV are defined as follows:

- **Rule 1 (module)**: Let Lev is the set of levels in CML. Each Levi ∈ Lev modeled as module declaration in I-SMV as follows:

  $$\text{Module } Lev_i(arg_i, \ldots, arg_{i+1})$$

  If $Lev_i \in Lev$ does not exist, then the execution must be terminated. In I-SMV, arg_i is reference to the actual parameter of a module in the main module.

- **Rule 2 (Variable)**: Let S_t be the set of states and Gr is the set of triggers in CML. $St_i \in St$ is declared inside a module as follows:

  $St_i : s_1..s_{n+1}$; if St is integer type
  $St_i : \{s_1,..,s_n\}$; if St is enumerated type
  $St_i : \{\}$; if St is boolean type

  $Gr_i \in G_r$ is declared inside a module as follows:

  $Gr_i : g_1..g_{n+1}$; if Gr is integer type
  $Gr_i : \{g_1,..,g_n\}$; if Gr is enumerated type
  $Gr_i : \{\}$; if Gr is boolean type

  Rule 2 is used, if and only if the represented module exists and either $St \neq \{\}$ or $Gr \neq \{\}$.

- **Rule 3 (state change)**: Let T_r be the set of transitions. In CML, the state changes might occur with or without a trigger, Gr. This implies that the state changes is between the source state, Ss and target state, St with or without trigger. The state changes in I-SMV defines as follows:

  ```
  next (St):=
   case{
     Tri : St; if gr ∈ Gr, Gr ≠ {}
     Trj : Ss; if gr ∈ Gr, Gr = {}
     default: St;
   };
  ```

  The different between Tri and Trj are Tri caused by triggered transition whereas Trj caused by null-triggered transition.

- **Rule 4 (Relation between modules)**: Let R_a , R_b, be state variables for Lev_a and Lev_b. Let R_c and R_d be state variables for Lev_{cg}. The relation between those levels define as follows:

  Module main()
  $St\text{-}Lev_c : Lev_c(St\text{-}Lev_a .R_a , St\text{-}Lev_b .R_b)$;
  $St\text{-}Lev_a : Lev_a(St\text{-}Lev_c .R_c)$;
  $St\text{-}Lev_b : Lev_b(St\text{-}Lev_c .R_d)$;

  $St\text{-}Lev_c$, $St\text{-}Lev_a$, and $St\text{-}Lev_b$ are state variables in the main module. In I-SMV the arguments to a module define by state variable of destination message followed by state variable of source destination message.

2.2 Verifying UML Statechart of Outpatient Clinic

This subsection describe a case study by employing the method above. This case study employ the statechart of Outpatient clinic. Based on translation rules from CML to SMV, motivate us to provide automatic translation tool that utilized the statechart of outpatient clinic to input language of SMV. Fig 1, shows our automatic translation architecture.

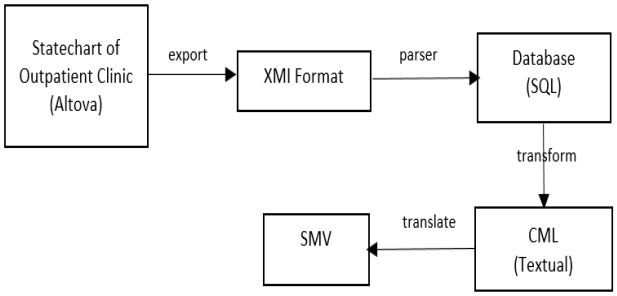

Fig. 1 - Automatic translation architecture

Based on Fig. 1, the verification process of state machines problem starts by expressing their behavior using UML statechart diagrams. To embark on modeling process, outpatient clinic has modeled a statechart under version Altova UModel. Fig. 2, illustrates the statechart of outpatient clinic. Based on Fig. 2, statechart of patient states in the example of surgical care service. There is one orthogonal state named as Active and two simple states which representing as Waiting and Consultation. Following the referral to clinic, the initial sub-states called "Pending" of parallel states "Appointment" and "Diagnostic tests" will activated.

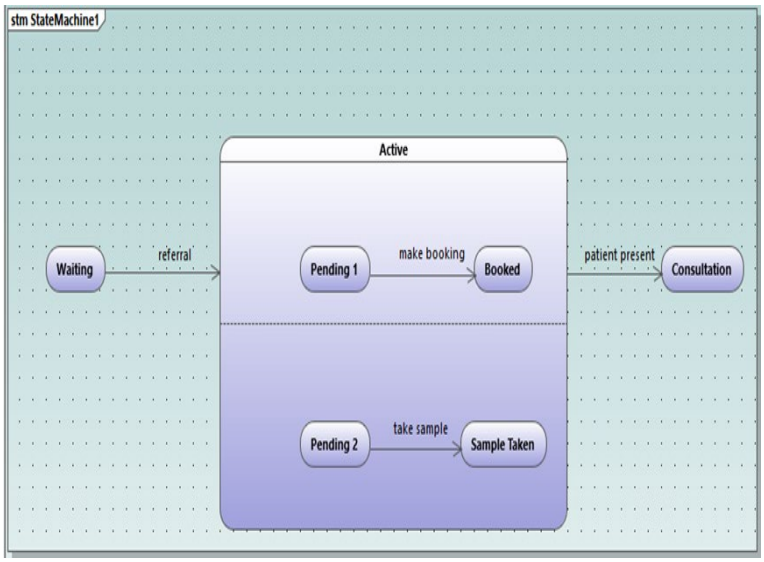

Fig. 2 - Statechart of outpatient clinic

When the event "make booking" fired and if there are available slots in the clinic, the patient considered to have the appointment booked. The statechart of outpatient clinic state has drawn using Altova. The following important properties need to verify for ensuring the correctness of statechart outpatient clinic:

i. All patients will serve for consultation if they are completed booked appointment and sample for diagnosis test has taken.
ii. All patients which are not in waiting list are strictly not allowed to make booking for appointment.
iii. All patients which always missed booked appointment and diagnosis test, they are still allowed for consultation.

We also define the property of statechart in reverse relation to check whether it reject or accept unusual behavior. The property is define as "All patients will always immediately entertain for consultation although he/she not making appointment and diagnosis test". Altova can produce XMI file and the file act as input to prototype system. With UML of version Altova, the diagram saved in XMI file. A special Java codes generated by using Netbeans IDE to read the XMI file. The methods such as DOM parser used to parse the required tag names into a set of tables by using a package named as ARCH. The tag names are <region>, <subvertex>, <transition> and <trigger> including the corresponding attributes. During the runtime, elements of XMI is transformed to textual CML by using data structure called as collection. There are five steps to transform the elements to textual CML:

i. Every element from the same component of both tables is group into COMPONENT in textual CML.
ii. For every Type which equal to uml:state in ExtrinsicObject, a list of states are created in textual CML under the STATE.
iii. For every Type equal to uml:trigger in the ExtrinsicObject, lists of trigger is created in textual CML under the TRIGGER_{n+1}.

iv. For every Type equal to uml:transition in Transition and if parent of uml:trigger equal to id of uml:transition, a set of transition are created in textual CML by getting source and trigger name together with target. The set of transition are group under TRANSITION.
v. For every component, if parent of uml:region equal to id of uml:state, a list of links are created in textual CML. The corresponding component is said successor-component and its parent component is said predecessor-component. The links can be formed into two types:
   - Successor-component: receive message from predecessor-component.
   - Predecessor-component: receive message from successor-component$_i$ and successor-component $_{i+1}$.

When all steps implemented, the output will used as guidance to model state machine and states transition of a system component. Fig. 3 shows the pseudocode of single textual CML.

```
SELECT  COMPONENT  FROM ExtrinsicObject, Transition
While (COMPONENT not finish) {
    <COMPONENT>:< printout COMPONENT >
    <STATE>     : While (STATE of AJ-1 not finish) {
                  printout STATE for this COMPONENT
                  }//end-while STATE
    <TRIGGER>   : While (TRIGGER of AJ-1 not finish) {
                  printout TRIGGER for this COMPONENT
                  }//end-while TRIGGER
    <TRANSITION>: while (transition of AJ-1 not finish){
                  if (PARENTID of uml:trigger == xmi:id
                     of uml:transition)
                  then (get SOURCE of uml:transition ;
                       get TRIGGER of uml:trigger
                       get TARGET of uml:transition)
                  }//end-while TRANSITION
 }//end-while COMPONENT
```

Fig. 3 - Pseudocode of single textual CML

The second pseudocode is used to implementing the relationship between single textual CML. Fig. 4, shows the pseudocode of relationship between components in textual CML.

```
Select * FROM ExtrinsicObject, Transition
<REFINEMENT>: get PARENTID of uml:Region and get xmi:id of
              uml:state;
              Case (COMPONENT){
               AJ-3: <This  module receive message from its
                     parent>
                     while (uml:Region not finish){
                      if (PARENTID of uml:Region == xmi:id of
                          uml:state)
                      then printout(COMPONENT of uml:state)
                     }//end-while
               AJ-2: <This  module receive message from its
                     parent>
                     while (uml:Region not finish){
                      if (PARENTID of uml:Region == xmi:id of
                          uml:state)
                      then printout(COMPONENT of uml:state)
                     }//end-while
               AJ-1:<This module receive message from its
                     children>
                     While (uml:state not finish){
                      If(child of uml:state == xmi:id of
                         uml:Region)
                      then printout(COMPONENTof uml:Region)
                     }//end-while
      }//end-case
```

Fig. 4 - Pseudocode of relationship between components in textual CML

The output of above pseudocode used to steer users in modeling the synchronization of system components. On the other hand, the output produced will used as message passing or sharing between the systems component. In overall, elements of XMI such STATE, TRIGGER, TRANSITION and REGION are useful to build the structure of CML. The detail usage of pseudocodes above will described in the next section.

3. Result and Discussion

This section focusses to verify the behavior of statechart of outpatient clinic with its properties. There are two important tasks; first, modeling statechart into SMV language, second expressing properties in temporal logic formula. For this purpose, we transform the textual CML into SMV. Fig. 5 and Fig. 6, show the textual of CML and input language of SMV, respectively. Based on Fig. 5, the textual of CML is obtain from exporting and parsing the XMI of statechart by using five steps of transformation, which described in previous page. In implementation, we apply the

pseudocodes describe in Fig. 3 and Fig. 4 to perform the transformation. As shown in Fig. 5, textual of CML will role as template of SMV language. Each of segment in textual of CML will mapping to SMV language.

```
AJ_2
STATE : [Pending1, Booked]
TRANSITION : Pending1 x makeBooking => Booked
TRIGGER : [makeBooking]

AJ_3
STATE : [Pending2, SampleTaken]
TRANSITION : Pending2 x takeSample => SampleTaken
TRIGGER : [takeSample]

AJ_1
STATE : [Consultation, Waiting, Active]
TRANSITION : Waiting x referral => Active
TRANSITION : Active x patientPresent => Consultation
TRIGGER : [referral]
TRIGGER : [patientPresent]

REFINEMENT
AJ_2 : receive message from [AJ_1]
AJ_3 : receive message from [AJ_1]
AJ_1 : receive message from [AJ_2, AJ_3]
```

Fig. 5 - Textual of CML

Fig. 6, shows the output of transformation from textual of CML to input language of SMV using translation rules, which previously describe in page 2 and 3. Subsequently, the textual of CML is transform to input language of SMV for verification purpose. Based on the Fig. 6, AJ-1, AJ-2, AJ-3 from textual of CML represent the *module* in SMV. There are three modules in SMV language; MODULE AJ-1, MODULE AJ-2 and MODULE AJ-3. STATE and TRIGGER from textual of CML represent *variable* in SMV. TRANSITION from textual of CML represent *next* in SMV. Lastly REFINEMENT from textual of CML represent *main* in SMV.

```
MODULE main
    VAR
        aj_2 : AJ_2 (aj_1);
        aj_3 : AJ_3 (aj_1);
        aj_1 : AJ_1 (aj_2, aj_3);

SPEC AG((aj_2.state=Booked) & (aj_3.state=SampleTaken) ->
    AF(aj_1.state=Consultation))
SPEC AG(!(aj_1.state=Waiting) -> EX !(aj_2.state=Booked))
SPEC AG(!(aj_2.state=Booked & aj_3.state=SampleTaken) ->
    AX(aj_1.state=Consultation))
SPEC AG ((aj_1.state=Active) -> EF(!(aj_1.state=Consultation)))

MODULE AJ_2 ( aj_1 )
    VAR
        state : {Pending1, Booked};
        trigger2 : {makeBooking};
    ASSIGN
        init ( state ) := Pending1;
        init ( trigger2 ) := makeBooking;
        next ( state ) := case
                ( ( state = Pending1 ) & ( trigger2 = makeBooking ) ) : Booked;
                1 : state;
                esac;
        next ( trigger2 ) := case
                ( trigger2 = makeBooking ) : makeBooking;
                1 : trigger2;
                esac;

MODULE AJ_3 ( aj_1 )
    VAR
        state : {Pending2, SampleTaken};
        trigger2 : {takeSample};
    ASSIGN
        init ( state ) := Pending2;
        init ( trigger2 ) := takeSample;
```

Fig. 6 - Input language of SMV

Eventually, SMV model checker runs the verification process. In this step, the statechart of outpatient clinic will go through the verification process to ensure whether its behaviors are behave correctly when we give correctness properties. For this purpose, we expressed the properties of outpatient clinic as shown in Table 1.

Table 1- Properties of outpatient clinic in CTL

No	Description of property	Property in CTL
1	All patients will entertained for consultation if they are completely booked an appointment and sample for diagnosis test has taken	AG(ai_2.state=Booked) & (aj_3.sate=SampleTaken)→ AF!(aj_1.state=Consultation))
2	All patients which are not in waiting list are strictly not allowed to make booking for appointment	AG(!(ai_1.state=Waiting) → EX!(aj_1.state=Booked))
3	All patients will always immediately entertained for consultation room although he/she not making appointment and diagnosis test (Reverse Relation)	AG(!(ai_2.state=Booked & aj_3.sate=SampleTaken)→ AX(aj_1.state=Consultation))
4	All patients which already booked appointment and completed diagnosis test not necessary entertained for consultation	AG(!(ai_1.state=Active) → EF(aj_1.state=Consultation))

Based on Table 1, we specify the properties in AG*p*, AX*p*, EX*p*, EF*p* and AF*p*. In SMV, all four temporal logics are express in CTL as below:

i. AG*p* express that along all paths property *p* holds globally,
ii. AX*p* express that always the property *p* holds in the second state of the path,
iii. AF*p* express that along all paths property *p* holds at some state in the future,
iv. EF*p* express that there exists a path where property *p* holds at some state in the future and
v. EX*p* express that there exists a path where property *p* holds at second state immediately.

Fig. 7 shows the output of verification. The verification result shows that property 1 and 4 are verify TRUE and property 2 and 3 are verify FALSE.

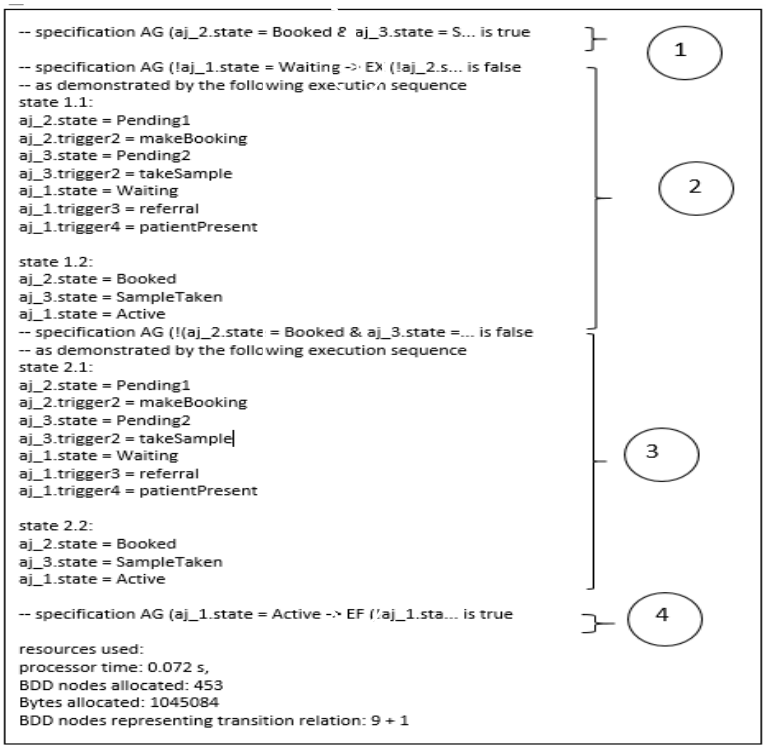

Fig. 7 - Result of SMV Verification

Based on Fig. 7, we formalize the property with correctly and vice versa to see whether the statechart gives the correct reaction according to the specifications. The property 1 is verify TRUE because it says that always when the patients have booked the appointment and diagnostic test, they will entertained for consultation. The property 2 is verify FALSE because *Booked* is not satisfied exactly in the second state. The result shows that the execution of sequence stop

at second state and not continue the rest of the states along the path. The property 3 also verify FALSE second state is not *Consultation* and stop execute the rest of states. The property 4 is verify TRUE because although patient always missed booked the appointment and diagnostic test, there exist in the future he/she will entertained for consultation. Temporal logic not stated exactly the time of satisfied, but it shows that once condition fulfill, the premise is satisfy either in the future or some or exactly second state.

4. Conclusion

This paper concentrated in providing an approach to verify the correctness of statechart model. As mentioned previously, correctness is defined as the obedience to the specifications and how software reacts when it is employing the correctness specification or property. The statechart used in this paper is outpatient clinic. It found that our approach successfully verifies the behavior of statechart of outpatient clinic based on the correctness property which precisely express in temporal logic. It means that, the behavior of statechart react, as we required when we employ the precise specification. It also proves that the statechart is safely design without any defect for next phase of software engineering. Therefore, we believe that our approach is useful in assisting people in using model checkers. Our approach is also beneficial in reducing the difficult tasks in using model checking such as formal modeling and formulizing the properties of a system.

In this paper, our approach only tackle verification of single orthogonal statechart. In future research, we focus to enhance our approach by providing verification of multiple orthogonal statecharts, which able to handle bigger scope behavioral of system.

Acknowledgement

This research sponsored by Ministry of Education Malaysia (MOE) and Universiti Putra Malaysia (UPM) via the Fundamental Research Grant Scheme - FRGS/1/2019/ICT01/UPM/02/01, Vot No: 5540288.

References

[1] Pooley, R., & King, P. (1999). The unified modelling language and performance engineering. IEE Proceedings-Software, 146(1), 2-10

[2] M. Sharbaf, B. Zamani and B. T. Ladani, "Towards automatic generation of formal specifications for UML consistency verification," 2015 2nd International Conference on Knowledge-Based Engineering and Innovation (KBEI), Tehran, 2015, pp. 860-865

[3] Samat, P. A., Zin, A. M., & Shukur, Z. (2011). Analysis of the model checkers' input languages for modeling traffic light systems. Journal of Computer Science, 7(2), 225-233

[4] Zou, Y. (2013). Verification of UML State Diagrams using a Model Checker (Doctoral dissertation, University of Wisconsin-La Crosse)

[5] Iwona, G. (2020). Formal Verification of Control Modules in Cyber-Physical Systems. Sencors Open Access Journal, 20(18), pp 51-54

[6] Butler, K. A, Mercer, e., Ali, B, Tao, C. (2015). Model Checking for Verification of Interactive Health IT System. AMIA Annual Symposium Proceeding Archive 2015, pp 349-358

[7] Alireza, S., Amir, M.R, Nima, J.N, and Reza, R. (2019). A Symbolic Model Checking approach in formal verification of distributed system, Human Centric Computing and Information Sciences, 9(4), 1-27

[8] Samat, P. A & Zin, A. M (2012). Common Modeling Language for Model Checkers, Journal of Computer Science,8(1):99-106

[9] Souri, A., Rahmani, A.M., Navimipour, N.J. et al. 2019. A symbolic model checking approach in formal verification of distributed systems. Hum. Cent. Comput. Inf. Sci. 9, 4 (2019)

[10] Kochaleema, K.H. and Santhoshkumar, G. 2019. Methodology for Integrating Computational Tree Logic Model Checking in Unified Modelling Language Artefacts: A Case Study of an Embedded Controller, Defense Science Journal, 69(1):58-64

[11] Jacques-Louis Lions et al. (1996). Ariane 5 Flight 501 Failure Report by the Inquiry Board. Technical report, European Space Agency, Paris, France.[9]

[12] Gerald, L. & Victor, C. 1999. Analyzing Mode Confusion via Model Checking, Lecture Note in Computer Science, 1680: 120-135

[13] Bahig, G., & El-Kadi, A. (2017). Formal Verification of Automotive Design in Compliance With ISO 26262 Design Verification Guidelines. IEEE Access, 5, 4505-4516

[14] Ba, T. N., & Arora, R. (2018, November). Towards Developing a Repository of Logical Errors Observed in Parallel Code for Teaching Code Correctness. In 2018 IEEE/ACM Workshop on Education for High-Performance Computing (EduHPC) (pp. 69-77). IEEE

[15] Gopalakrishnan, G., Hovland, P. D., Iancu, C., Krishnamoorthy, S., Laguna, I., Lethin, R. A. & Solar-Lezama, A. (2017). Report of the HPC Correctness Summit, Jan 25--26, 2017, Washington, DC. arXiv preprint arXiv:1705.07478

[16] Perez, B., Porres, I. (2010). Authoring and Verification of Clinical Guidelines: A model-driven approach, Journal of Biomedical Informatics,43(4), 520-536

[17] Xiong, X., Liu, J., Ding, Z. (2010). Design and Verification of a Trustable Medical System, Electronic Note Theoretical in Computer Science,77-92

[18] Wang, X., Rutle, A.(2014). Model Checking Healthcare Workflows using Alloy, Procedia Computer Science 37 (2014) 481 – 488

[19] Douglass, B. P. 2000. Real-Time UML developing efficient objects for embedded systems. Addison-Wesley

[20] Harel, D. & Kugler, H. (2004). The RHAPSODY Semantics of Statecharts (or, On the Executable Core of the UML). In Integration of Software Specification Techniques for Application in Engineering. Lecture Notes in Computer Science, 3147: 325-354

[21] Clarke Jr, E. M., Grumberg, O., Kroening, D., Peled, D., & Veith, H. (2018). Model checking. MIT press.

[22] Berard, B., Bidoit, M., Finkel, A. & Laroussinie, F. 2013. Systems and Software Verification: Model-checking Techniques and Tools, Springer-Verlag

[23] Razali, R. & Garratt, P. 2010. Usability Requirements of Formal Verification Tools: A Survey. J. Comput. Sci., 6: 1189-1198

[24] Samat, P. A., & Zin, A. M. (2012). CMGT: Support Tool for Using Model Checking. Australian Journal of Basic and Applied Sciences, 6(13): 63-73