Formal Verification Unified Modeling Language Statechart Using Enhancement Common Modeling Language

Muhammad Amsyar Azwarrudin, Pathiah Abdul Samat, Norhayati Mohd Ali, Novia Indriaty Admodisastro
Faculty of Computer Science and Information Technology-Software Engineering Department,
Universiti Putra Malaysia, Serdang, Malaysia

***Abstract*—Modern systems are rapidly evolving and increasing in complexity to satisfy growing requirements. Such systems often incorporate multiple hierarchical statecharts within their behavior modeling diagram, which significantly complicates the verification process. To address this challenge, the Common Modeling Language (CML) was introduced as an intermediate modeling language for formal verification, serving as a bridge between Unified Modeling Language (UML) Statechart and the model checkers. However, CML supports modeling only a single hierarchical statechart, which limits its applicability to complex systems. This study introduces the Enhancement Common Modeling Language (E-CML), an extension of the CML, to support the verification of systems that incorporate multiple hierarchical statecharts. We introduce the group component in E-CML, comprising an initial state, a set of states, transitions, triggers, and a region, to formally differentiate the group components from superstates. We also propose new translation rules to map E-CML into Symbolic Model Verifier (SMV) syntax. E-CML operates through two main processes: transformation and translation. The transformation process transforms an XML Metadata Interchange (XMI) file into E-CML, while the translation process translates E-CML to an Input Symbolic Model Verifier (I-SMV) file. The system is verified using the SMV model checker, with formal properties specified in Computational Tree Logic (CTL) and represented in the I-SMV file. The results demonstrate that the behavior modeling diagram satisfies all formal properties, indicating that E-CML provides an effective framework for the verification of complex systems comprising multiple hierarchical statecharts.**

***Keywords*—CML; E-CML; formal verification; model checkers; UML Statechart**

I. Introduction

The Unified Modeling Language (UML) Statechart is a semi-formal modeling language widely used by software engineers to describe the behavior of software systems, whether simple or complex. In [1], the authors introduced the UML Statechart, which has advanced elements that can describe a complex system's behavior. The simple system's behavior consists of a single hierarchical statechart and a few modules to express the behavior. An example of a simple system is a traffic light system [2], [3]. Meanwhile, the complex system's behavior consists of multiple hierarchical statecharts and multiple modules to express it. The example of complex systems is surgical consultation in the outpatient clinic health system [4]. However, the UML Statechart lacks formal semantics that ambiguously express the behavior modeling. Recent research has addressed the problem that the UML Statechart cannot be used for verification processes [5], [6], even though software engineers widely use the UML Statechart for modeling simple or complex systems. In [7], the authors introduced the Extended Hierarchical Automata (EHA), an extension of finite state machines that supports the hierarchical structure and defines the operational semantics of EHA. The EHA is one of the formal semantics behavior modeling that represents the UML Statechart. Therefore, the UML Statechart can be used for the verification process to check the system's correctness.

Consequently, model checking can be employed as a verification method. Model checking is an automated process for verifying the system's behavior. However, it requires a formal model and a formal temporal logic specification to check the system's correctness. If the model checker does not produce a counterexample, the system is proven to satisfy its requirements [8]. In this context, the Model-Based Software Engineering (MBSE) approach enables the formalization of semi-formal models into formal models through either direct or indirect translation using translation rules. In the case of indirect translation, the model is first transformed into an intermediate modeling language. Examples of such intermediate languages include the Common Modeling Language (CML) [8], Gamma statechart [9], and Abstract Rule-based [10]. CML is an intermediate modeling language for formal verification, serving as a bridge between UML Statechart and the model checkers.

Modern systems are rapidly evolving and increasing in complexity to satisfy growing requirements. Such systems often incorporate multiple hierarchical statecharts within their behavior modeling diagram, which significantly complicates the verification process. Therefore, recent research has proposed numerous methods that can formalize the UML Statechart [11], [12], [13], transforming it into a formal model that can be used for the verification process. Nevertheless, with the complexity of the systems developed nowadays, numerous complex interactions are involved [13]. In [9] and [14], the authors mentioned that integrative approaches are limited to models with fewer modules or lack efficient compatibility with formal verification and validation frameworks. Moreover, reference [15] stated that the CML supports modeling only a single hierarchical statechart, which limits its applicability to complex systems. Therefore, both statements show that recent

approaches have focused on verifying a single hierarchical statechart. However, modeling complex systems involves multiple hierarchical statecharts. It is crucial to address CML's limitations and resolve this problem to ensure the correctness of complex systems. Therefore, this study aims to propose an enhancement of the CML that is capable of verifying systems with multiple hierarchical statecharts, which represent the complex behavior of the systems.

The remainder of this study is organized as follows: Section II reviews related works on formal modeling, semi-formal modeling, theorem proving, model checking, MBSE, and formal verification of UML Statechart, particularly for multiple hierarchical statecharts. Section III presents the research framework and the proposed method, which includes the formal definition of E-CML and the development of new translation rules. Section IV discusses the results and discussion, illustrating the case study using the proposed method. Section V provides a conclusion, summarizes the findings, and outlines potential directions for future work.

II. Related Works

A. Formal Modeling

Formal modeling is an unambiguous language with precise specification built using mathematical and formal methods [16]. Therefore, the formal modeling can specify the requirements, designs, and behavior of the system. It provides the formal semantics of the systems. Examples of formal modeling are Petri Nets, Vienna Development Method (VDM), and Kripke structures. Petri Nets are directed, bipartite multigraphs consisting of places, transitions, and arcs. Fuzzy Petri Nets (FPN) are extensions of Petri Nets and are effective in modeling uncertain biological systems. In [17], the authors reviewed the FPN and categorised it into three types: basic Fuzzy Petri Nets, Fuzzy Quantitative Petri Nets, and Petri Nets with Fuzzy Kinetic Parameters. In [18], the authors proposed an extension of Coloured Petri Nets (CPN) known as catalog-nets (CLog-nets) to integrate business processes with different data types. In [19], the authors utilized Petri Nets to generate secure smart contracts in blockchain systems, enabling the detection of deadlock at an early stage.

Recent studies have used the VDM language to model and verify complex systems. VDM Specification Language (VDM-SL) is a formal specification language with an extensive executable subset and a toolbox that verifies models. In [20], the authors mentioned that the VDM-SL is a robust specification that can define the system's functionality without ambiguity. They proposed an approach that serves as a pre-formal notation for the natural language specification before it is implemented in VDM-SL. In [21], the authors proposed an Internet of Things (IoT) based formal model for vehicle ad hoc networks (VANETs) using VDM-SL. The proposed formal model will be transformed into VDM-SL and verified using the VDM toolbox to check the system's correctness and absence of deadlocks. Meanwhile, reference [22] introduced the first approach to developing the formal semantics of a simulation language in a VDM-SL. They proposed a RE: MODIYC that can describe and examine the formal specification of VDM-SL after initial implementation.

The Kripke structures are a mathematical model used in model logic to represent the possible states and the relationships among them. However, the Kripke structures formalize model logic by extending classical propositional logic with model operators such as necessity and possibility. In [23], the authors proposed an alternative method that transforms a given Kripke structure and the corresponding LTL formal properties into a Buchi automaton, where the number of accepting states is determined based on the LTL formal properties. Meanwhile, reference [24] introduced an approach to verify mobile-interactive systems. They model the properties of dynamic interaction using Variable Petri Nets (VPN). The VPN is transformed into the Kripke structures using a proposed algorithm and verified with a model checker. In [25], the authors proposed formalizing natural language using the UML use case diagram with the Kripke structures. They use the NuSMV model checker to verify whether the formal specification is consistent with the Kripke structures model. However, there are some previous studies that use semi-formal modeling to model the systems.

B. Semi-Formal Modeling

Semi-formal modeling is an ambiguous language in which the modeling lacks formal semantics, but it provides a better understanding of modeling. Therefore, semi-formal modeling is more popular than formal modeling because it does not require as much time and effort to understand the model. Examples of semi-formal modeling are UML and System Modeling Language (SysML). The UML is a graphical notation language used to define and document a system during software development [26]. In [27], the authors introduced a hierarchical modeling formalism based on UML Statechart with generally distributed (GEN) durations, aimed at ensuring modeling simplicity and efficient evaluation of steady-state or transient behavior up to absorption. In contrast, reference [28] used the new semantics of the UML Statechart to model the digital watch. In [29], the authors proposed using a restricted version of the Statechart (Revised Statechart) to model the behavior of autonomous robotic surgery. In [30], the authors used the UML Sequence Diagram to express the requirements of safety-critical systems incrementally.

However, another semi-formal modeling is SysML, which describes a hybrid system of hardware and software. SysML is a graphical modeling language that is intuitive and widely used for modeling complex systems, comprising hardware and software. In [31], the authors proposed an approach to analyze the safety of complex systems. They model the structure of the system using the SysML Block Definition Diagram (BDD) while utilizing the SysML State Machine Diagram to describe the system's behavior. In [32], the authors introduced the transformation of the UML/SysML Activity Diagram to Petri Nets. Meanwhile, reference [33] used the SysML Activity Diagram to model the system's behavior and transform the model into the input language of the Next-generation Unifying Symbolic Model Verifier (nuXmv) model checker. Therefore, there are two types of modeling language, which are formal modeling and semi-formal modeling. However, both models require a formal verification approach to ensure their correctness. Formal verification is the process of ensuring that

a system satisfies certain specifications, typically using two approaches: theorem proving and model checking [34].

C. Theorem Proving

Theorem proving is a formal method that can establish the validity of theorems. The model and specification of the systems are described in a mathematical statement. The verification process of theorem proving involves proving theorems about the system using the rules of the inference process. In [35], the authors introduced the automated prover and proof assistant, GPT-f, to analyze the performance of the Metamath formal language. In [36], the authors proposed the transformation rules from the UML Activity Diagram to Formal Calculus for Liveness and Zero state (FoCaLiZe), a proof-based formal language, and verified the proof-based formal language with the Zenon theorem prover. In [37], the authors identified a barrier for researchers using machine learning for theorem proving in Large Language Models (LLMs). Therefore, they introduced LeanDojo, an open-source Lean framework designed to overcome this barrier, which enables researchers to utilize machine learning with LLMs to enhance the effectiveness of theorem proving.

D. Model Checking

Model checking is one of the formal verification approaches and an automated process for verifying the behavior of a software system. The model checking approach requires a formal model and a formal temporal logic specification to check the correctness of the systems. In [11], the authors enhanced their previous work by formalizing the concurrent element of the UML Statechart, which involves multiple hierarchical statecharts. In [38], the authors used Maude as the formal language for the Internet of Things (IoT) software systems. They use Maude's model checker and Linear Temporal Logic (LTL) specification to verify the IoT software system. Meanwhile, reference [33] utilized the nuXmv to verify Autonomous Artificial Pancreas Systems (AAPS). They used the LTL specification in the nuXmv model checker to verify the APPS system. The model checking approach is widely used due to the MBSE technique, which enables the formalization of modeling language through a process. The transformation process has two types: direct transformation and indirect transformation, which require an intermediate language for the indirect transformation.

E. Model-Based Software Engineering

MBSE provides a platform that enables the definition of complex, multidisciplinary systems. However, the MBSE approach requires formal verification to ensure the correctness of complex systems. Therefore, the MBSE approach provides transformation rules to transform the model, either direct transformation or indirect transformation. Some previous studies use the direct transformation in their approaches.

In [39], the authors identified that the UML State Machine Diagram lacks formal semantics. Therefore, they proposed direct transformation rules from UML State Machine Diagram to Petri Nets and verified them with the Time Petri Nets Analyzer (TINA) to check correctness, safety, and liveness. In [31], the authors verified the Integrated Modular Avionic (IMA) system by proposing the direct transformation rules. They transform the SysML BDD and SysML State Machine Diagram into the input language of the NuSMV model checker. Meanwhile, reference [13] also introduced direct transformation rules from the SysML State Machine Diagram to the input language of the NuSMV model checker for verifying the driving control system. Although both research utilize the SysML State Machine Diagram, their approaches and direct transformation rules differ slightly.

However, some previous studies use the indirect transformation, which introduces an intermediate language to solve their problems. In [8], the authors introduced the CML, an intermediate language to formalize the UML Statechart. They proposed transformation rules to transform the UML Statechart into the CML and translation rules to translate the CML into the input language of the SMV model checker. In [9], the authors proposed the Gamma Statechart, an intermediate language that uses the formal semantics of Yakindu Statechart. They also proposed translation rules to translate the Gamma Statechart into Timed Automata (TA). In [14], the authors proposed transformation rules to transform the SysML State Machine Diagram to the Gamma Statechart. They also proposed the translation rules to translate the Gamma Statechart into the Uppsala Time Automata Laboratory (UPPAAL). Meanwhile, reference [10] used the Abstract Rules-based language as an intermediate language to formalize the UML State Machine Diagram. Previous studies show that most researchers utilize the UML Statechart and SysML State Machine Diagram. However, they may not encompass all elements of the UML Statechart that can represent a complex system.

F. Formal Verification of UML Statechart

Table I illustrates previous studies related to the formal verification of UML Statechart. We consider approaches that use either direct transformation or indirect transformation. We also examine which elements of the UML Statechart are addressed in these approaches. Our focus is on the system complexity, which can be decomposed into multiple modules. Therefore, we consider whether the approaches handle a single hierarchical statechart or multiple hierarchical statecharts. The symbols are indicated as "/" and "X" refer to "have" and "have not", respectively.

TABLE I. PREVIOUS STUDIES RELATED TO THE FORMAL VERIFICATION OF UML STATECHART

Journals	Intermediate Language	Approach	Transformation Rules	Single Hierarchical Statechart	Multiple Hierarchical Statechart
[15]	CML	Indirect	/	/	X
[11]	X	Direct	/	/	X
[12]	X	Direct	/	/	/
[9]	Gamma Statechart	Indirect	X	/	/
[13]	Abstract Rule-based	Indirect	X	/	X
[31]	X	Direct	/	/	X

Some previous studies [15], [9], [13] used an intermediate language to formalize UML Statechart. They employed the indirect transformation approach to verify the models and proposed a few transformation rules to transform the UML Statechart into the intermediate language and subsequently translate it into the model checker's formal model. However, two of these studies [9], [13] used an intermediate language without specifying whether they applied the translation rules in their approaches. The remaining studies [11], [12], [31] did not use an intermediate language; instead, they employed the direct transformation approach, which transforms the UML Statechart directly into the formal model. They also proposed a few transformation rules to guide this transformation.

As mentioned earlier, we focus on system complexity. Most previous studies have addressed a single hierarchical statechart, which represents a relatively simple system. However, only two studies have considered multiple hierarchical statesharts, despite modern systems being built with considerable complexity. Therefore, this observation motivates us to focus on verifying systems with multiple hierarchical statecharts using an indirect transformation approach. Specifically, we aim to enhance the CML by introducing new elements into its formal definition and by creating new translation rules that define multiple hierarchical statecharts and complex systems. The following section will explain the research framework and proposed method.

III. Proposed Method

A. Research Framework

Fig. 1 illustrates the research framework for the proposed method. It involves four processes: 1) Modeling the UML Statechart. 2) Exporting to XML Metadata Interchange (XMI) format. 3) Transforming XMI to Enhancement Common Modeling Language (E-CML). 4) Translating E-CML to Input Symbolic Model Verifier (I-SMV).

Process (1) Modeling the UML Statechart: The behavior of the system is modeled with the UML Statechart by using the Altova UModel software. The Altova UModel software enables modeling of UML Statechart features, including composite state, orthogonal state, history pseudostate (deep and shallow), and inter-level transitions. Process (2) Exporting the UML Statechart to XMI format: The Altova UModel software provides features for exporting the UML Statechart into the XMI format. In the XMI format, there are <region>, <subvertex>, <transition>, and <trigger> tags that refer to the regions, states, transitions, and triggers of the UML Statechart, respectively. Process (3) Transforming XMI to E-CML: The transformation process is based on the tag elements from the XMI format and the formal definition of E-CML. The <region> tags are transformed into the region. The <subvertex> tags are transformed into the state. The <transition> tags are transformed into the transition, and the <trigger> tags are transformed into the trigger. Process (4) Translating E-CML to I-SMV: The translation process is based on the formal definition of I-SMV. Four translation rules are involved in this process, which translate the elements of E-CML into the elements of I-SMV.

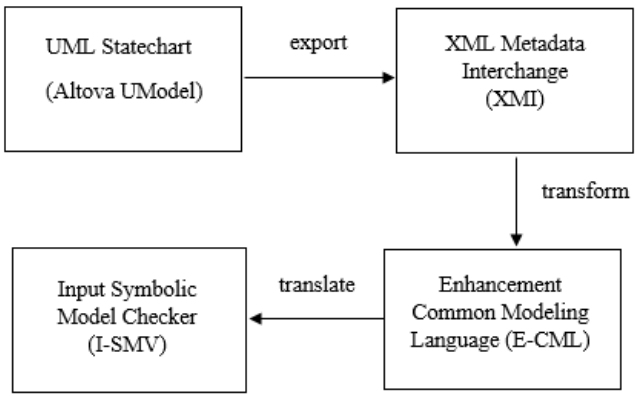

Fig. 1. Research framework.

B. Enhancement of Common Modeling Language

The E-CML is an extension of the CML based on the features of the UML Statecharts. Definition 1 defines the formal definition of E-CML as follows:

Definition 1: Enhancement Common Modeling Language

E-CML = <S, S_0, S_c, G, T, Regions, L, Gp, R>

Where:

S = finite set of states, where each state, s, is declared one of the two types: {AND, OR}.

S_0 = set of initial states ($S_0 \subseteq S$). S_0 forms a valid initial transition relation.

Sc = set of states that form a valid state configuration.

G = finite set of triggers.

T = finite set of transition relations, T = S x G x S'.

Regions = finite set of regions.

Gp = finite set of group components. Each group component consists of states, an initial state, triggers, transitions, and a region. Gp = <S, S_0, S_c, G, T, Regions>.

L = $S \rightarrow Gp'$ is the group component-level function. If $Gp' \subseteq L(S)$, then Gp' is an immediate descendant of S. The function of L describes the hierarchical state of the model.

R = relation between the superstate and group components, Gp.

There are two types of E-CML states: AND and OR. The AND state models concurrency by composing multiple simultaneous group components, Gp. The AND state is a superstate for the group components, Gp, of the statechart that are concurrently active. The descendants of the AND state must always be the OR state. The OR state is a state that supports one of the states inside another, providing a hierarchy in the behavior model. The OR state has states related to each other by an exclusive OR relationship. The leaf states of the E-CML must always be OR states. Each OR state has its region.

Each region is represented as the group component, Gp, in E-CML, consisting of the initial state, S_0, states, S, transitions, T, triggers, G, and the region. This also includes the outermost region represented as a group component, Gp. At runtime, E-CML allows multiple active states, referred to as a state configuration, S_c. A state configuration, S_c, always contains one state from the AND state and all states from the OR states.

Fig. 2 illustrates an example of the UML Statechart model that consists of multiple hierarchical statecharts. It shows there are basic states (A, D, G, H, I, J, M, N, O, and P), AND states (B and C), and OR states (E, F, K, and L). B state contains two OR states (E and F), E state contains two basic states (G and H), and the F state contains two basic states (I and J). Meanwhile, the C state contains two OR states (K and L), the K state contains two basic states (M and N), and the L state contains two basic states (O and P).

The operation of the E-CML is described using the step semantics. The state configuration, S_c, of E-CML starts with the initial state, S_0. In E-CML, when the state configuration, S_c, includes a composite AND state, all of its regions are active concurrently. Each region typically behaves like an OR state, meaning exactly one substate within each region is active at any time.

For example, the state configuration for the AND states in Fig. 2 should be (B, G, I) or (B, H, J) and (C, M, O) or (C, N, P). In E-CML, a transition will always occur at each step in each active state configuration, S_c. An implicit transition is triggered if no explicitly modeled transitions are enabled. The synchronization of E-CML allows it to be flattened into sequential automata, preserving the formal model semantics. Each single flattened component E-CML is equivalent to a sequential automaton [6].

Fig. 3 illustrates the flattened hierarchical E-CML structure of the UML Statechart model in Fig. 2.

Therefore, the single flattened component E-CML is known as the group component, Gp, in Fig. 3 as follows:

Gp_1 = ({A, B, C, D}, A, {E1, E2, E7, E8, E13, E14}, {(A, E1, B), (B, E2, A), (B, E7, C), (C, E8, B), (C, E8, B), (C, E13, D), (D, E14, C)}, Region 1)

Gp_2 = ({G, H}, G, {E3, E4}, {(G, E3, H), (H, E4, G)}, Region 2)

Gp_3 = ({I, J}, I, {E5, E6}, {(I, E5, J), (J, E6, I)}, Region 3)

Gp_4 = ({M, N}, M, {E9, E10}, {(M, E9, N), (N, E10, M)}, Region 4)

Gp_5 = ({O, P}, O, {E11, E12}, {(O, E11, P), (P, E12, O)}, Region 5)

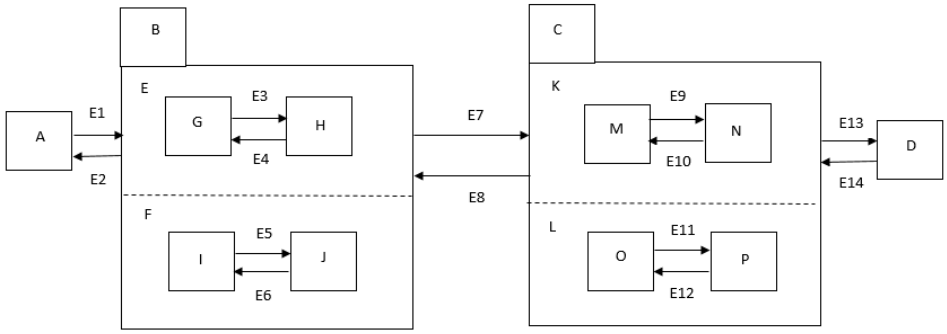

Fig. 2. UML Statechart model.

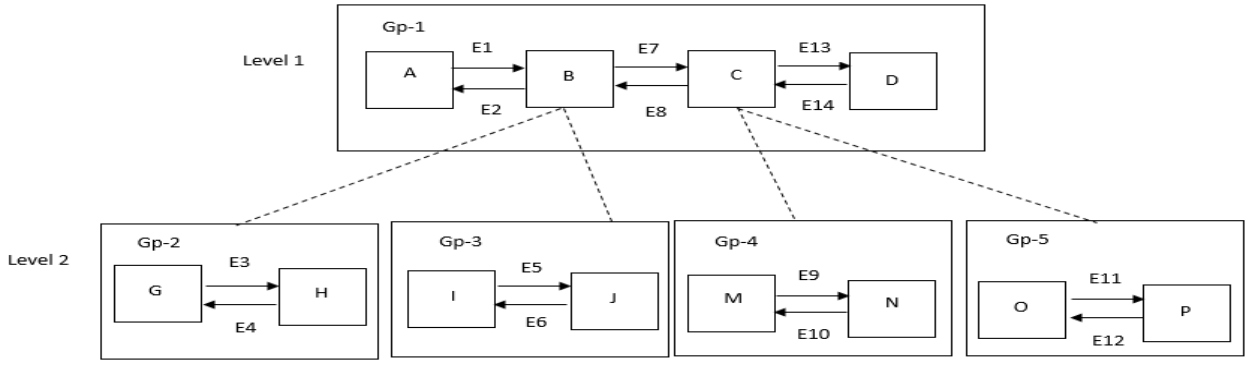

Fig. 3. Flattened hierarchical E-CML structure.

The sequence of elements in the group component, Gp, follows the sequence of variables in the formal definition of the group component, Gp, as given in Definition 1. The group component, Gp, interacts through events. In the following step, an event may trigger a transition in the system's synchronous group component, Gp. If the event triggers a transition from a state and the result is a group component, Gp, then the state is referred to as a superstate. For example, B is the superstate of Gp_2 and Gp_3, while C is the superstate of Gp_4 and Gp_5. This situation creates inter-level transitions. The inter-level

transitions cross hierarchy boundaries. If a transition leaves a superstate, the firing of all transitions containing the group components, Gp, of the superstate is suppressed, as represented by the dotted line in Fig. 3. This creates the relations, R, between the superstate and group components, Gp. Therefore, the group component level function, L, is created. The superstate's group component level function, L, is Level 1. The descendant of the superstate's group component level function, L, is Level 2. Therefore, the relation, R, between superstate and group components, Gp, is defined as follows:

Gp_2: receive a message from Gp_1

Gp_3: receive a message from Gp_1

Gp_4: receive a message from Gp_1

Gp_5: receive a message from Gp_1

Gp_1: receive a message from Gp_2, Gp_3, Gp_4, and Gp_4

In the E-CML, there are two processes involved: transforming the UML Statechart to the E-CML and translating the E-CML to the I-SMV. Therefore, we use the formal definition of I-SMV that is defined by [6]. The formal definition of I-SMV is as follows:

Definition 2: Input Symbolic Model Verifier

I-SMV = <M, V, N, Y>

Where:

M = set of finite modules

V = set of finite state variables

N = set of next states

Y = relation between modules

The I-SMV is modular. The high-level module is called the main module, while the other modules are known as sub-modules. In a module, M, there are state variables, V, to describe the module. A state evolves from one state to another through a next operator, N. The relation, Y, between one module and another is described by using a set of parameters. A set of rules guides the translation from E-CML to I-SMV.

C. Translation Rules of Enhancement of Common Modeling Language

Based on Definition 2, several rules of translation are created to map the E-CML, which is the source model, to the I-SMV, the target model. The translation from the source model to the target model can be defined by using a set of rules [31]. In the I-SMV, the group component, Gp, of E-CML corresponds to the module, M. The set of states, S, and triggers, G, corresponds to the state variables, V. The transitions, T, correspond to the next state, N. Lastly, the relation, R, between the superstate and group components, Gp, corresponds to the relation between modules, Y. The translation rules for mapping the E-CML to the I-SMV are defined as follows:

Rule 1 (Module): Let Grp be the set of group components in the E-CML. Each $Grp_i \in Grp$ is modeled as a module declaration in I-SMV as follows:

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

where, i = 1, ..., n, and n is the maximum integer.

If $Grp_i \in Grp$ does not exist, then the execution must be terminated. In I-SMV, arg_i refers to the actual parameter of a module within the main module.

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

$St_i : s_1, \ldots., s_{n+1}$; if St is an integer type

$St_i : \{s_1, \ldots., s_n\}$; if St is enumerated type

$St_i : \{\}$; if St is a Boolean type

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

$Gr_i : g_1, \ldots., g_{n+1}$; if Gr is an integer type

$Gr_i : \{g_1, \ldots., g_n\}$; if Gr is enumerated type

$Gr_i : \{\}$; if Gr is a 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): Tr be the set of transitions. In E-CML, state changes may occur with or without a trigger, Gr. This implies changes between the source state, Ss, and the target state, St, with or without the presence of the trigger. The state changes in I-SMV are defined as follows:

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

The first statement defines the state changes caused by triggered transitions, while the second statement defines the state changes caused by null-triggered transitions.

Rule 4 (Relation between modules): Let $St\text{-}Grp_b.R_b$, $St\text{-}Grp_c.R_c$, $St\text{-}Grp_d.R_d$, and $St\text{-}Grp_e.R_e$ are state variables for Grp_a. Let $St\text{-}Grp_a.R_a$ is the state variable for Grp_b, Grp_c, Grp_d, and Grp_e. The relation between those levels is defined as follows:

Module main()

$St\text{-}Grp_b : Grp_b (St\text{-}Grp_a.R_a)$;

$St\text{-}Grp_c : Grp_c (St\text{-}Grp_a.R_a)$;

$St\text{-}Grp_d : Grp_d (St\text{-}Grp_a.R_a)$;

$St\text{-}Grp_e : Grp_e (St\text{-}Grp_a.R_a)$;

$St\text{-}Grp_a : Grp_a (St\text{-}Grp_a.R_a)$;

$St\text{-}Grp_a$, $St\text{-}Grp_b$, $St\text{-}Grp_c$, $St\text{-}Grp_d$, and $St\text{-}Grp_e$ are state variables in the main module. In I-SMV, the arguments to a module are defined by the state variable of the destination message, followed by the state variable of the source message.

IV. Results and Discussion

In this section, we implement the E-CML for the Surgical Consultation in the Outpatient Clinic Health System as the case study. In general, doctors will refer patients for surgical consultation if they believe the underlying health condition can be effectively treated through surgery. Following the referral, the outpatient clinic schedules an appointment with the surgeon and arranges for samples to be collected if further diagnostic tests are required. The surgeon evaluates the necessity for an operation by analysing symptoms and test results during the consultations. Following a decision to proceed, the patient is registered on a surgical wait list to ensure an appropriate time is booked for use of the operating room. Before surgery, the patient may be educated by the surgeon about the procedure. When the operating room is already booked, the patient waits for their scheduled surgery date. Fig. 4 illustrates the UML Statechart Surgical Consultation in the Outpatient Clinic Health System.

Fig. 4 illustrates that there are two AND states ("Receiving referral" and "Receiving treatment") and four OR states ("Appointment", "Diagnostic Test", "Surgical Waiting List", and "Education"). There are three basic states at the outermost region ("Waiting for referral to clinic", "At consultation", and "Operation booked"), two basic states at the "Appointment" state ("Pending 1" and "Booked"), two basic states at the "Diagnostic Test" state ("Pending 2" and "Sample taken"), two basic states at the "Surgical Waiting List" state ("Pending 3" and "On waiting list"), and two basic states at the "Education" state ("Pending 4" and "Educated"). This model has five regions; each OR state has one region, and the other region is the outermost region. Fig. 5 illustrates the flattened hierarchical E-CML structure for the Surgical Consultation in the Outpatient Clinic Health System.

Based on Fig. 5, each flattened is referring to the group component, Gp, and is defined as follows:

Gp_1 = ({Waiting for referral to clinic, Receiving referral, At consultation, Receiving treatment, Operation booked}, Waiting for referral to clinic, {referral to clinic, patient referral at clinic, treatment decided, booked operation}, {(Waiting for referral to clinic, referral to clinic, Receiving referral), (Receiving referral, patient referral at clinic, At consultation), (At consultation, treatment decided, Receiving treatment), (Receiving treatment, booked operation, Operation booked)}, Region 1).

Gp_2 = ({Pending 1, Booked}, Pending 1, {make booking}, {(Pending 1, make booking, Booked)}, Region 2)

Gp_3 = ({Pending 2, Sample taken}, Pending 2, {take sample}, {(Pending 2, take sample, Sample taken)}, Region 3)

Gp_4 = ({Pending 3, On waiting list}, Pending 3, {register}, {(Pending 3, register, On waiting list)}, Region 4)

Gp_5 = ({Pending 4, Educated}, Pending 4, {educate}, {(Pending 4, educate, Educated)}, Region 5)

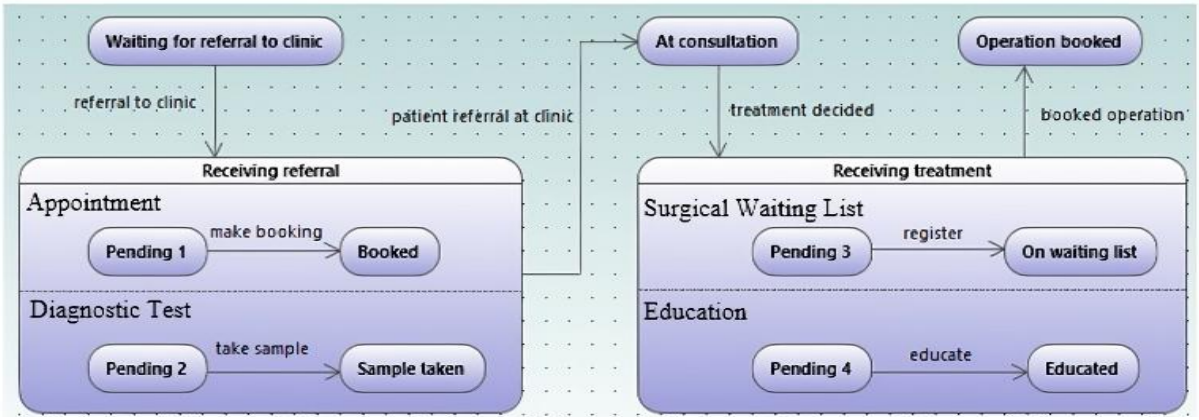

Fig. 4. UML Statechart surgical consultation in the outpatient clinic health system.

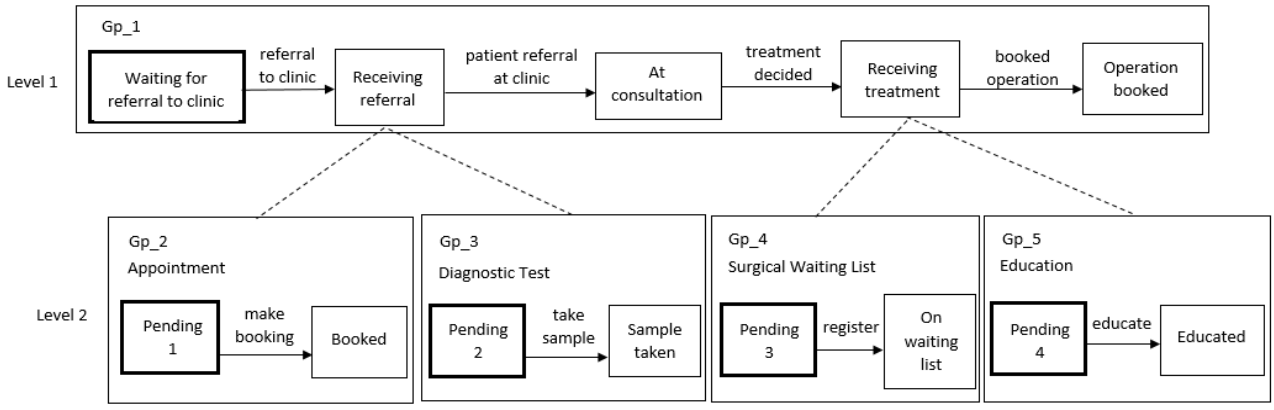

Fig. 5. Flattened hierarchical E-CML structure for surgical consultation in the outpatient clinic health system.

The relation, R, between superstate and group components, Gp, is defined as follows:

Gp_2: receive a message from Gp_1

Gp_3: receive a message from Gp_1

Gp_4: receive a message from Gp_1

Gp_5: receive a message from Gp_1

Gp_1: receive a message from Gp_2, Gp_3, Gp_4, and Gp_4

We use the translation rules of E-CML (Rule 1-4) to translate the E-CML into the I-SMV. Rule 1: Each group component, Gp, in E-CML is translated to the module's name in the I-SMV, followed by the list of arguments. Each module's arguments are based on the relation, R, which is translated in Rule 4. Therefore, the modules are defined as follows:

```
MODULE Gp_2 (gp_1)
MODULE Gp_3 (gp_1)
MODULE Gp_4 (gp_1)
MODULE Gp_5 (gp_1)
MODULE Gp_1 (gp_2, gp_3, gp_4, gp_5)
```

Rule 2 is regarding the variables declared for each module. All states, S, and triggers, G, in E-CML are defined as follows:

```
MODULE Gp_2 (gp_1)
VAR
    state:{Pending 1, Booked};
    trigger2:{make booking};
MODULE Gp_3 (gp_1)
VAR
    state:{Pending 2, Sample taken};
    trigger3:{take sample};
MODULE Gp_4 (gp_1)
VAR
    state:{Pending 3, On waiting list};
    trigger4:{register};
MODULE Gp_5 (gp_1)
VAR
    state:{Pending 4, Educated};
    trigger5:{educate};
MODULE Gp_1 (gp_1)
VAR
    state:{Waiting for referral to clinic, Receiving
    referral, At consultation, Receiving treatment,
    Operation booked};
    trigger6:{referral to clinic, patient referral at clinic,
    treatment decided, booked operation};
```

Rule 3 is regarding the transitions, T, in the E-CML, which are translated to the next (state) in the I-SMV. Therefore, the next (state) in the I-SMV is defined as follows:

```
MODULE Gp_2 (gp_1)
…..
next (state):= case
    ((state = Pending 1) & (trigger2 = make
    booking)): Booked;
1: state;
esac;
MODULE Gp_3 (gp_1)
…..
next (state):= case
    ((state = Pending 2) & (trigger3 = take sample)):
    Sample taken;
1: state;
esac;
MODULE Gp_4 (gp_1)
…..
next (state):= case
    ((state = Pending 3) & (trigger4 = register)): On
    waiting list;
1: state;
esac;
MODULE Gp_5 (gp_1)
…..
next (state):= case
    ((state = Pending 4) & (trigger5 = educate)):
    Educated;
1: state;
esac;
MODULE Gp_1 (gp_2, gp_3, gp_4, gp_5)
…..
next (state):= case
    ((state = Waiting for referral to clinic) & (trigger6 =
    referral to clinic)): Receiving referral;
    ((state = Receiving referral) & (trigger7 = patient
    referral at clinic)): At consultation;
    ((state = At consultation) & (trigger8 = treatment
    decided)): Receiving treatment;
```

((state = Receiving treatment) & (trigger9 = booked operation)): Operation booked;

1: state;

esac;

Rule 4 is regarding the relation, R, between superstate and group components, Gp, in E-CML. In -SMV, Rule 4 is translated into MODULE main. The state variable of the MODULE main is defined as a call module for the corresponding group component. The arguments in each call module are the state variable of the source state, followed by its message passing. The relation between modules is defined as follows:

MODULE main:

VAR

gp_2: Gp_2 (gp_1);

gp_3: Gp_3 (gp_1);

gp_4: Gp_4 (gp_1);

gp_5: Gp_5 (gp_1);

gp_1: Gp_1 (gp_2, gp_3, gp_4, gp_5);

Based on the relation between modules above, gp_2, gp_3, gp_4, and gp_5 are state variables used as arguments for the module in Gp_1. The gp_1 is the state variable for the modules in Gp_2, Gp_3, Gp_4, and Gp_5. We use the SMV model checker to verify the UML Statechart in Fig. 4. The formalized properties need to be specified on the "SPEC" keyword in the I-SMV file. Table II shows the list of formalized properties, descriptions, and the results.

A. Discussion

In this section, we discuss and explain the results that were obtained from the proposed method. All the formalized properties, which cover the multiple hierarchical statecharts, are verified as TRUE. P1 is verified as TRUE because the specification defines that the patient can book an appointment and take the diagnostic test in the future, even though the patient is not waiting for the referral. P2 is verified as TRUE because the specification states that the patient always receives a consultation from the doctor after completing the booking appointment and taking the diagnostic sample test. P3 is verified as TRUE because the specification defines that the patient can always book the operation date after registering on the waiting list and getting education about the surgery from the surgeon. P4 is verified as TRUE because the specification defines that the patient can be on the waiting list for an operation and can receive education about the surgery from the surgeon after the patient receives a consultation from the surgeon.

According to reference [15], correctness is defined as adherence to the specifications and the way software behaves when these correctness specifications are applied. The results indicate that E-CML can serve as an effective intermediate language, as the verification of the specifications returned a result of TRUE. Moreover, E-CML is able to verify complex systems that have multiple hierarchical statecharts, which are a common feature for modern complex systems.

TABLE II. FORMALIZED PROPERTIES, DESCRIPTIONS, AND RESULTS

Property	Description	CTL	Result
P1	A patient who is not waiting for a referral may be able to book an appointment and undergo the diagnostic tests in the future.	AG (!(gp_1.state = Waiting_for_referral_to_clinic) → EF (gp_2.state = Booked & gp_3.state = Sample_taken))	True
P2	A patient who completes a booking and has taken a sample for a diagnostic test is eligible for a consultation.	AG (gp_2.state = Booked & gp_3.state = Sample_taken) → EF (gp_1.state =At_consultation)	True
P3	A patient registered for the operation and received education from the surgeon can book the operation date.	AG ((gp_4.state = On_waiting_list & gp_5.state = Educated) → EF (gp_1.state = Operation_booked))	True
P4	The patient whom the surgeon has consulted can be placed on the waiting list for the operation and receive education on the operation.	AG ((gp_1.state = At_consultation) → EF (gp_4.state = On_waiting_list & gp_5.state = Educated))	True

TABLE III. COMPARISON OF THE E-CML WITH THE OTHER APPROACHES

Method	Approach	Intermediate Language	Substate	Orthogonal State	History State (depp/shallow)
E-CML	Indirect transformation	E-CML	/	/	X
Colored Petri Nets [11]	Direct transformation	-	/	/	/
Gamma Statechart [9]	Indirect transformation	Gamma Statechart	/	X	X

Table III illustrates a comparison between E-CML and other approaches. We compare E-CML with the approaches proposed in [11] and [9]. We also consider the UML Statechart elements covered by each approach during the verification process. The symbols "/" and "X" indicate "supported" and "not supported", respectively.

In [11], the authors proposed a method based on a direct transformation approach, which transforms one model into another without using an intermediate language. The authors utilized UML Statechart to express the system's behavior and formalized it into the CPN. This approach uses the transformation rules to map the elements of the UML Statechart to CPN. In contrast, E-CML provides a formal definition that represents the elements of the UML Statechart. It also includes transformation rules to transform E-CML into an I-SMV file, which is used as the input language for the verification process with the SMV model checker. The E-CML helps and encourages the user to apply the model checking method without requiring knowledge of SMV syntax or CPN.

As discussed, both approaches are capable of verifying multiple hierarchical statecharts. However, some UML Statechart elements are not included in E-CML, whereas they are included in the approach proposed in [11], such as history state (deep and shallow).

Meanwhile, reference [9] proposed a method based on an indirect transformation approach, which same approach used by E-CML and requires an intermediate language. The authors utilized UML Statechart to express the system's behavior and transformed it into the TA. This approach also considers the transformation of ports in the physical components. In contrast, E-CML focuses only on behavioral aspects and does not transform or include physical components. As discussed, both approaches are capable of verifying multiple hierarchical statecharts. However, some UML Statechart elements are not included in the approach proposed in [9], whereas they are included in E-CML, such as orthogonal states. Nevertheless, both approaches do not include the history state (deep and shallow).

V. Conclusion

This study presents a proposed solution for verifying complex systems that are inherently more intricate and difficult to verify. Modern systems often incorporate multiple hierarchical statecharts within their behavior modeling diagram, which significantly complicates the verification process. Therefore, E-CML, an enhancement of CML, helps the users verify complex systems with multiple hierarchical statecharts and ensure their correctness. E-CML introduces group components comprising an initial state, a set of states, transitions, triggers, and a region, to formally differentiate the group components from superstates. It creates new translation rules to define multiple hierarchical statecharts and complex systems that relate to the group components. E-CML demonstrates that it can verify the formalized properties of multiple hierarchical statecharts model. As a result, all of the formalized properties are verified as TRUE. Thus, E-CML enables verification of unverified complex systems. However, E-CML has a limitation in that it cannot verify the history states, including either the deep or shallow pseudostate. This feature is an important element of the UML Statechart that can represent the behavior of complex systems. Some modern systems also include a history state in the behavioral models to store the last active state, such as in air conditioning systems. Therefore, it is important to enhance E-CML to include all UML Statechart elements capable of representing complex systems. This enhancement will enable the verification of all types of complex systems built today.

Acknowledgment

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

References

[1] Harel, D. (1987). Statecharts: A visual formalism for complex systems. *Science of Computer Programming, 8*, 231–274. https://doi.org/10.1016/0167-6423(87)90035-9.

[2] 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. https://doi.org/10.3844/jcssp.2011.225.233.

[3] Souri, A., Ali, M., & Norouzi, M. (2012). Analyzing SMV & UPPAAL model checkers in real-time systems. *AWERProcedia Information Technology & Computer Science, 1*, 631–639.

[4] Vasilakis, C., Lecznarowicz, D., & Lee, C. (2008). Application of UML to the modeling of health care system: An introduction and literature survey. *International Journal of Healthcare Information Systems and Informatics, 3*(4), 39–52. https://doi.org/10.4018/jhisi.2008100103.

[5] Khalid, M., Zafar, N. A., Afzaal, H., Latif, S., Hassan, S., & Rehman, A. (2019). Automated UML-based formal model of E-Health system. In *2019 13th International Conference on Mathematics, Actuarial Science, Computer Science and Statistics (MACS)* (pp. 1–6). https://doi.org/10.1109/MACS48846.2019.9024830.

[6] Muhamad, Z. H., Abdulmonim, D. A., & Alathari, B. (2019). An integration of UML use case diagram and activity diagram with Z language for formalization of library management system. *International Journal of Electrical and Computer Engineering, 9*(4), 3069–3076. https://doi.org/10.11591/ijece.v9i4.pp3069-3076.

[7] Mikk, E., Lakhnech, Y., & Siegel, M. (1997). Hierarchical automata as model for statecharts (extended abstract). In R. K. Shyamasundar & K. Ueda (Eds.), *Advances in computing science—ASIAN'97* (*Lecture Notes in Computer Science*, Vol. 1345). Springer. https://doi.org/10.1007/3-540-63875-X_52.

[8] Samat, P. A., & Zin, A. M. (2012). Common modeling language for model checkers. *Journal of Computer Science, 8*(1), 99–106. https://doi.org/10.3844/jcssp.2012.99.106.

[9] Molnár, V., Graics, B., Vörös, A., Majzik, I., & Varró, D. (2018). The Gamma statechart composition framework: Design, verification, and code generation for component-based reactive systems. In *Proceedings of the 40th International Conference on Software Engineering: Companion* (pp. 113–116). https://doi.org/10.1145/3183440.3183489.

[10] Grobelna, I. (2020). Formal verification of control modules in cyber-physical systems. *Sensors, 20*(18), Article 5154. https://doi.org/10.3390/s20185154.

[11] André, É., Benmoussa, M. M., & Choppy, C. (2015). Formalising concurrent UML state machines using coloured Petri nets. In V.-H. Nguyen, A.-C. Le, & V.-N. Huynh (Eds.), *Knowledge and Systems Engineering* (pp. 473–486). Springer. https://doi.org/10.1007/978-3-319-11680-8_38.

[12] Guo, C., Ren, S., Jiang, Y., Wu, P., Sha, L., & Jr, R. B. B. (2016). Transforming medical best practice guidelines to executable and verifiable statechart models. In *2016 ACM/IEEE 7th International Conference on Cyber-Physical Systems (ICCPS)* (pp. 1–10). https://doi.org/10.1109/ICCPS.2016.7479121.

[13] Mahani, M., Rizzo, D., Paredis, C., & Wang, Y. (2021). Automatic formal verification of SysML state machine diagrams for vehicular control systems. *SAE Technical Paper*. SAE International. https://doi.org/10.4271/2021-01-0260.

[14] Horváth, B., Molnár, V., Andolfato, L., & Gomes, I. (2020). Model checking as a service: Towards pragmatic hidden formal methods. In *ACM/IEEE 23rd International Conference on Model Driven Engineering Languages and Systems (MODELS '20 Companion)* (pp. 1–5). https://doi.org/10.1145/3417990.3421407.

[15] Samat, P. A., Azwarrudin, M. A., Mohd, N., & Admodisastro, N. (2021). Verifying the correctness of UML Statechart outpatient clinic based on Common Modeling Language and SMV. *The International Journal of Integrated Engineering, 13*(5), 137–145. https://doi.org/10.30880/ijie.2021.13.05.015.

[16] Singh, A., Parizi, R. M., Zhang, Q., Choo, K.-K. R., & Dehghantanha, A. (2020). Blockchain smart contracts formalization: Approaches and challenges to address vulnerabilities. Computers & Security, 88, Article 101654. https://doi.org/10.1016/j.cose.2019.101654.

[17] Liu, F., Heiner, M., & Gilbert, D. (2020). Fuzzy Petri nets for modelling of uncertain biological systems. *Briefings in Bioinformatics, 21*(1), 198–210. https://doi.org/10.1093/bib/bby118.

[18] Ghilardi, S., Gianola, A., Montali, M., & Rivkin, A. (2020). Petri nets with parameterised data: Modelling and verification. In D. Fahland, C. Ghidini, J. Becker, & M. Dumas (Eds.), *Business Process Management*

(BPM) 2020. Lecture Notes in Computer Science, 12168 (pp. 55–74). Springer. https://doi.org/10.1007/978-3-030-58666-9_4.

[19] Zupan, N., Kasinathan, P., Cuellar, J., & Sauer, M. (2020). Secure smart contract generation based on Petri nets. In R. da Rosa Ringhi, A. M. Alberti, & M. Singh (Eds.), *Blockchain Technology for Industry 4.0* (pp. 73–98). Springer. https://doi.org/10.1007/978-981-15-1137-0_4.

[20] Oda, T., Kusakabe, S., Chang, H.-M., & Larsen, P. G. (2022). VDM-SL in action: A FRAM-based approach to contextualise formal specification. In H. D. Macedo & K. Pierce (Eds.), *Proceedings of the 20th International Overture Workshop* (pp. 5–18). https://doi.org/10.48550/arXiv.2208.10233.

[21] Iqbal, S., Zafar, N. A., Ali, T., & Alkhammash, E. H. (2022). Efficient IoT-based formal model for vehicle-life interaction in VANETs using VDM-SL. *Energies, 15*(3). https://doi.org/10.3390/en15031013.

[22] Oda, T., Dur, G., Ducasse, S., & Macedo, H. D. (2023). *Implementation-first approach of developing formal semantics of a simulation language in VDM-SL* (arXiv:2303.14944). arXiv. https://doi.org/10.48550/arXiv.2303.14944.

[23] Rungsetthaphat, N., & Vatanawood, W. (2023). Transformation of Kripke structure with linear temporal logic formula to Büchi automata. In 2023 27th International Computer Science and Engineering Conference (ICSEC) (pp. 1–5). IEEE. https://doi.org/10.1109/ICSEC59635.2023.10329762.

[24] Yang, R., Ding, Z., Guo, T., Pan, M., & Jiang, C. (2022). Model checking of variable Petri nets by using the Kripke structure. *IEEE Transactions on Systems, Man, and Cybernetics: Systems, 52*(2), 7774–7786. https://doi.org/10.1109/TSMC.2022.3163741.

[25] Zaman, Q. uz, Nadeem, A., & Sindhu, M. A. (2020). Formalizing the use case model: A model-based approach. *PLoS ONE, 15*(4). https://doi.org/10.1371/journal.pone.0231534.

[26] Koç, H., Erdogan, A. M., Barjakly, Y., & Peker, S. (2021). UML diagrams in software engineering research: A systematic literature review. *Proceedings 2021*. https://doi.org/10.3390/proceedings2021074013.

[27] Carnevali, L., German, R., Santoni, F., & Vicario, E. (2022). Compositional analysis of hierarchical UML statecharts. IEEE Transactions on Software Engineering, 48(12), 4762–4788. https://doi.org/10.1109/TSE.2021.3125720.

[28] Exelmans, J., Van Mierlo, S., & Vangheluwe, H. (2022). A statecharts interpreter and compiler with semantic variability. In *ACM/IEEE 25th International Conference on Model Driven Engineering Languages and Systems (MODELS '22 Companion)*. ACM. https://doi.org/10.1145/3550356.3561569.

[29] Falezza, F., Piccinelli, N., De Rossi, G., Roberti, A., Kronreif, G., Setti, F., Fiorini, P., Fellow, L., & Muradore, R. (2021). Modeling of surgical procedures using statecharts for semi-autonomous robotic surgery. *IEEE Transactions on Medical Robotics and Bionics, 3*(4), 888–899. https://doi.org/10.1109/TMRB.2021.3110676.

[30] Chen, X., Mallet, F., & Liu, X. (2020). Formally verifying sequence diagrams for safety critical systems. In *International Symposium on Theoretical Aspects of Software Engineering (TASE) 2020* (pp. 217–224). https://doi.org/10.1109/TASE49443.2020.00037.

[31] Wang, H., Zhong, D., Zhao, T., & Ren, F. (2019). Integrating model checking with SysML in complex system safety analysis. *IEEE Access, 7*, 16561–16571. https://doi.org/10.1109/ACCESS.2019.2892745.

[32] Huang, E., McGinnis, L. F., & Mitchell, S. W. (2020). Verifying SysML activity diagrams using formal transformation to Petri nets. *Systems Engineering, 23*(1), 118–135. https://doi.org/10.1002/sys.21524.

[33] Staskal, O., Simac, J., Swayne, L., & Rozier, K. Y. (2022). Translating SysML activity diagrams for nuXmv verification of an autonomous pancreas. *2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC)* (pp. 1637–1642). https://doi.org/10.1109/COMPSAC54236.2022.00260.

[34] Mahmoud, A. T., Mohammed, A. A., Ayman, M., Medhat, W., Selim, S., Zayed, H., Yousef, A. H., & Elaraby, N. (2024). Formal verification of code conversion: A comprehensive survey. *Technologies, 12*(12), 1–28. https://doi.org/10.3390/technologies12120244.

[35] Polu, S., & Sutskever, I. (2020). Generative language modeling for automated theorem proving. *arXiv*. https://doi.org/10.48550/arXiv.2009.03393.

[36] Abbas, M., Rioboo, R., Ben-Yelles, C., & Snook, C. F. (2021). Formal modeling and verification of UML activity diagrams (UAD) with FoCaLiZe. *Journal of Systems Architecture, 114*(September 2020), 101911. https://doi.org/10.1016/j.sysarc.2020.101911.

[37] Yang, K., Swope, A. M., Gu, A., Chalamala, R., Song, P., Yu, S., Godil, S., Prenger, R., & Anandkumar, A. (2023). LeanDojo : Theorem proving with retrieval-augmented language models. *37th Conference on Neural Information Processing System (NeurIPS 2023)*, 1–40. https://doi.org/https://doi.org/10.48550/arXiv.2306.15626.

[38] Fortas, A., Kerkouche, E., & Chaoui, A. (2022). Formal verification of IoT applications using rewriting logic: An MDE-based approach. *Science of Computer Programming, 222*, 102859. https://doi.org/10.1016/j.scico.2022.102859.