Transforming UML State Machine Diagram to High Level Petri Net Using Genetic Algorithm

Aysh Alhroob and Nidal Yousef

Abstract—This paper presents a new methodology to transform UML State diagram to High Level Petri Net (HLPN). Genetic Algorithms (GAs) has been used to generate software specifications set which covers maximum states and transitions in state diagram. Then, these states are used to improve the logical ordering of HLPN. Not all states will be used to represent the HLPN, but those that are selected by GA which reduce the number of HLPN details and avoiding the redundant of states.

Index Terms—UML, HLPN, software development model, model transformation, GA.

I. INTRODUCTION

In UML there is a main limitation for verification and assigning the non-functional parameters of the software. This limitation comes from UML semi-formal characteristic, especially in the first cycle of the software production. In order to avoid its semi-formal problem, a new methodology is proposed in [1] to transform the sequence and class diagrams to HLPN. The main limitation in [1] is how to control the sequencing of states in HLPN. In this paper we are introducing a new methodology to get the sequence states using GA from state diagram to enrich the logical ordering of states in HLPN. HLPN is a graphical diagram for the formal description of the flow of activities in complex systems. HLPN are mainly used to represent in a natural way logical relations among parts or activities in a system. The situations that can be modelled by HLPN are synchronization, sequentially and concurrency [2]. It consists of places, that contain a number of tokens, and transitions which may remove tokens from input places and send it tokens to output places [3]. In HLPN places contain values, and the arc inscriptions of transitions contain expressions and variables.

The rest of the paper is organized as follows: Related work is presented in Section II. Section III covers the proposed methodology. The Experiments are covered in section IV. The conclusion and future work recommendation are discussed in Section V.

II. PREVIOUS WORK

In [4], the authors proposed some rules to transform sequence diagrams into colored Petri nets. Ourdani et al. [5] have transformed sequence diagram to HLPN in simple structures based on mapping of messages as well as conveying them. Z. Hu and S. M. Shatz [6] propose a conversion of the hierarchical representation of UML state diagrams to extended hierarchical automaton (EHA) as an intermediate step and then translate it to PROMELA. More recently it has been recognized that formal methods must be adapted such that developers without deep mathematical knowledge may also use them [7]. Transformation of UML state diagrams to high level Petri nets that use Object Oriented (OO) concepts, has been proposed in [8]. In contrast, an automatic translation of state chart and sequence diagram into generalized stochastic nets is proposed in [9], a composition of the resulting net models appropriate for getting a given analysis object. The transformation is based on message sender and receiver component. Use-case, state and sequence diagrams are used in Emadi and Shams [10] to represent executable model based on Petri nets. Doungsa-Ard et al. [11] proposed an approach to generate test cases from state diagram using GA; this approach detected the states and transitions from state diagram automatically which help us in this paper to trace the state diagram specifications.

Alhroob et al. [1] proposed a methodology to transform sequence diagram and class diagram to HLPN, at the end of that work the problem that raised is about the logical ordering for the states that HLPN should represents. The sequence diagram logical ordering introduced not clear image about these states. For this reason, this work extends the proposed methodology in [1] and use state diagram to enrich the information about the states.

III. PROPOSED METHODOLOGY

In this section, UML state diagram is used as source of software specifications and genetic algorithm as the technique for generating related scenarios. The quality of the generated scenarios is measured by the coverage of transitions and states occurred by each scenarios. The design of our approach is discussed at the end of this section.

A. UML State Machine Diagram

State Machine diagrams are used to illustrate the possible states of an object and the causes and effects of that state [12]. UML state machine contains states in a particular time. Each state in UML state machine is as same as in other finite state transition system. They are connected by transitions. Receiving a trigger associated with the current states causes a change of state of the system. The trigger will turn on the transition which is adjacent with the current states.

Fig. 1 is an example of the UML state machine diagram for a coffee vending machine. The status of machine is changed...
to “on” when user turns on the machine. If money has been added to the machine, the amount of money is increased. If a cup of coffee is requested by the user, the state machine will check if there is money in the machine, and then set up a cup of coffee. After that the money in state machine is decreased. A transition is an edge which connects between states. In each transition, there are four optional components; transition name, trigger, guard condition, and action expression. For example, in transition “t7: dec[money > 1] / money = money - 1”, “t7” is a transition name. “dec” is a trigger. “money > 1” is a guard condition, and “money = money - 1” is an action expression. A transition name is used for the transition ID. A trigger is a member event which can occur within the system.

Fig. 1 contains a set of triggers like “dec”, “inc”, “poweron”, “power-off”, and “coffee.” The trigger can be fired based on the current state of the state machine diagram and the trigger which is input to the diagram. The guard condition is needed to evaluate to be true if the transition is considered to be fired.

A transition is an edge which connects between states. In each transition, there are four optional components; transition name, trigger, guard condition, and action expression. For example, in transition “t7: dec[money > 1] / money = money - 1”, “t7” is a transition name. “dec” is a trigger. “money > 1” is a guard condition, and “money = money - 1” is an action expression. A transition name is used for the transition ID. A trigger is a member event which can occur within the system. The trigger can be fired based on the current state of the state machine diagram and the trigger which is input to the diagram. The guard condition is needed to evaluate to be true if the transition is considered to be fired.

B. Coverage Criteria

The quality of scenarios that are evaluated based on the number of coverage the scenarios can cover. A number of levels of coverage; transition coverage level, full predicate coverage level, transition-pair coverage level, and complete sequence level are proposed by Offutt and Abdurazik [13] as a coverage level for UML diagrams using.

- Transition coverage level is a set of transitions which scenario satisfies.
- Transition-pair coverage level designs coverage from pairs of adjacent transitions.
- Complete sequence level is a meaningful sequence of transitions to which is designed by the software designer.

In this work, we use transition coverage level for measuring the effectiveness of scenario by considering the maximum coverage criteria.

C. Software Specification Scenario

In this section, our aim is to decompose the state machine diagrams to its scenarios. Each scenario should contain start state and end states. Furthermore, each state should be represented by its trigger(s) and transition(s). A proposed approach in [14] generated a sequence of triggers. From the sequence of triggers, the trigger is extracted and used to fire transitions in the state machine diagram. This approach provides us a clear image about the triggers and fired transitions.

Table I shows an example of a sequence of triggers and how the trigger fires the transitions in the state diagram depicted in Fig. 1. A coffee vending machine state machine diagram in Fig. 1 is used by proposed approach in [11] to get a number of transitions that have maximum coverage criteria. The output of the approach gives us these transitions: “t1”, “t2”, “t3”, “t4”, “t5”, and “t8.” In this work, we consider these transitions to generate the scenarios.

<table>
<thead>
<tr>
<th>Trigger</th>
<th>State Executed</th>
<th>Current State</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>Initial state, ti1, off</td>
<td>Off</td>
</tr>
<tr>
<td>power-on</td>
<td>t1, On, Initial state: supply, ti2, Idle</td>
<td>Idle, Empty</td>
</tr>
<tr>
<td>inc</td>
<td>t5, noEmpty</td>
<td>Idle, Empty</td>
</tr>
<tr>
<td>inc</td>
<td>t6, noEmpty</td>
<td>Idle, Empty</td>
</tr>
<tr>
<td>power-on</td>
<td>No change</td>
<td>Idle, NoEmpty</td>
</tr>
<tr>
<td>inc</td>
<td>t6, noEmpty</td>
<td>Idle, NoEmpty</td>
</tr>
<tr>
<td>dec</td>
<td>t8, noEmpty</td>
<td>Idle, NoEmpty</td>
</tr>
</tbody>
</table>

**D. Scenarios Graph (SG)**

In this section, the SG will be defined and we will then present the proposed methodology to generate SG from State Machine Diagram (SMD). SG is defined as follow:

\[ TSG = (RSG; XSG; QSG; FSG; PARSG; BRSG,) \]

where,

- **TSG** is the set of all triggers in SMD.
- **PTSG** is the set of all transitions from one state to another.
- **QSG** is the initial node representing a state from which an operation begins.
- **FSG** is the set of final nodes representing states where an operation terminates.
- **PARSIG** is the set of parallel edges and it represents states where one trigger fired other transition at the same time.

The proposed methodology in this chapter identifies the set of all scenarios Where; \( scns = scn1, scn2, ..., scnm \) and the set of all nodes must be identified. Initially SG contains only the **StartState** and then each node of all **scns**, **scns** should be followed by its corresponding **NextState**, and removes the duplicates, if any. The **StartState** for the different scenarios as illustrated in Fig. 1 is State off and two **FinalStates** (Idle and Empty).

An operation starts with an **InitialState** and undergoes a number of intermediate states due to the occurrence of various transitions. **InitialState** can be easily detected from XML file; the **InitialState** do not contains reference number in XML. The algorithm of the generating SG from SMD uses the XML file as input to get the SMD scenario graph as the output. This process goes through the following steps:

- Find out all states, triggers, transitions and guard conditions.
- Detect the InitialState and check if it was visited before.
- Detect the trigger of the InitialState.
- If it was visited before, means it affects more than one transition.
- Detect the transition(s) that has been affected by trigger(s).
- If the edge leads to FinalState, consider the path from InitialState to FinalState as one scenario.
- If two or more states are related by loop relationship, connect all loop states by one path.
- If two or more states are related by parallel relationship, connect all parallel states by one circle notation at the beginning of process and in the end as well.
- If two states are related by option relationship, connect option edges by one.

E. SMD Scenarios

Now, the SG represents all of the SMD transitions and triggers. The initial state is detected by identifying its preconditions and post-conditions, simply if the precondition of the state is not coming from post-condition of other state, it can be said that this is an initial state.

SMD scenarios generation is the main step towards the generation of correct HLPN. Where the proposed technique enumerates all possible paths from the InitialState to the FinalState of the SG that cover all states. Each path then is visited to determine which transitions and triggers are covered. Fig. 1 represents 4 triggers (PowerOn, PowerOff, Inc and Dec) denote transitions (t1, t2, t3, t4, t5, t6, t7 and t8). Whereas, SG is responsible for matching each state with related triggers and transitions. Proposed technique enumerated two paths:

1) InitialState → Supply → Idle → Busy → Idle, FinalState
2) InitialState → Money → Empty → NotEmpty → FinalState

The main aim in this work is to generate scenarios that have most coverage criteria rather than all and full paths. Next section introduces using GA to generate refined scenarios.

F. SMD Scenarios Refining Using Genetic Algorithms

The proposed approach targets to generate SMD scenarios to cover maximum transitions using genetic algorithms technique [11]. The SD is responsible for generating the states, transitions and triggers. The SD receives a sequence of triggers, and then extracts each trigger to test with the UML state machine diagram. The transition, which is fired by the trigger, is recorded. Data structure is created to represent transition path from the given triggers. This data structure is used for calculating fitness value in FitnessCalculation part. FitnessCalculation part calculates fitness value depending on coverage level. Currently, the transition level coverage is used. Finally, the genetic algorithms part is responsible for the genetic algorithm operation.

GA is iterative procedures which work with chromosomes. The chromosomes are a population of candidate solutions that are maintained by the GAs throughout the solution process [9]. At first a population of chromosomes is generated randomly. A selection operator is used to choose two solutions from the current population. The selection process used the measured goodness of the solutions (Fitness Function). The crossover operator swaps sections between these two selected solutions with a defined crossover probability. One of the chromosomes solutions is then chosen for application of the mutation process. The algorithm is ended, when a defined stopping criterion is reached. Since our approach focuses on finding a set of transitions by triggers firing, a chromosome in our approach is a sequence of triggers itself. UML state machine diagram triggers will be the first population from all possible triggers in the sequence of triggers. The coffee machine state diagram in Fig. 1 is shown in the top row of Table I. The fitness value for each chromosome is calculated from the number of transitions which is fired by the input sequence. If there is any trigger cannot fire any transition, the technique skips it and gets the next trigger in the sequence.

For example, in case of Fig. 1 if a chromosome is shown as a sequence of power-on, inc, tm, dec, after inc trigs, current state is Idle, and notEmpty. From this status tm cannot make any new coverage state or transitions. If dec is considered to unveil new coverage transition which is t7, it is look like power-on, inc, tm, dec can cover transition t7. But tm trigger does not involve any transitions which help to reach transition t7. Omitting that trigger may make the length of chromosome changed, suggesting to use a GA with variable chromosome length. The coverage of state and transition is considered as shown in Table I. A sequence of trigger is power-on - inc - tm - inc, tm, which is generated randomly. The sequence is applied for revealing state and transition coverage as follows. Starting from initial state, there is one transition connected to initial state and there is no event trigger on that transition.

Hence, current state has been move to off. Then, the methodology extracts the first trigger in the sequence, power-on to execute. With event trigger power-on, there is a transition from state off, which needs power-on trigger. That is a transition t1. Current state is now moved to state on. Now there are two sub initial states in state on, therefore, current state has been moved to two initial states. These two initial states bring system to new current states idle and empty. Next event trigger has been taken, inc. Transitions from state idle and empty is checked whether they contains a trigger inc. From the diagram, transition r5 from empty state contains inc trigger, as a consequence, current state is move from empty to notEmpty. Next event trigger is tm. Unfortunately, event tm does not drive any transitions, which are connected state notEmpty and idle.

The search for coverage is stopped, because no new state is covered. Fitness function is calculated by a number of states and transitions covered by the chromosome. In addition, states and transitions, which are covered by a pool of chromosome in each generation, are recorded. If a chromosome in a new generation covers new state or transition has not been covered before, there will be an addition score for that chromosome. These values used for analysis in selecting function and evolutionary operation.

GAs operators used in proposed technique are the two point crossover and random mutation. Based on some experimentation and previous knowledge on GAs application with other problem [11], the parameters of genetic operation
IV. TRANSFORM SMD TO HLPN NODES

Now, to make the conversion from State diagram to HLPN more easily, the proposed methodology in this work depends on the node integration approach that proposed in this work. Every SMD state will represent one node; the edge represents two states connected by one transition. Every node contains the transition with its input state and output state, i.e. edge (One→ti2→ Idle). An operation starts with a starting state and undergoes a number of intermediate states due to the occurrence of various events. Nodes Relationships Table (NRT) that proposed in [1] used to set the relationships between the nods automatically. The intuition of the nodes (NRT) that proposed in [1] used to set the relationships of various events. Nodes Relationships Table is to capture the relationship for every pair of distinct nodes. The conditions that associated with each of the above nodes relationship are commonly exclusive and exhaustive. The final HLPN that is generated from this methodology is represented in Fig. 2.

![HLPN from state diagram.](image)

Fig. 2. HLPN from state diagram.

V. CONCLUSION AND FUTURE WORK

In this paper, we propose a methodology for using genetic algorithm to generate HLPN from UML state machine diagram. The HLPN nodes to be generated are a sequence of transitions which are fed into the system. The fitness function used in the system is the transition coverage. In our experiment we select only the best solution which covers most of transitions. The coverage result of some examples is not good because of the complexity of the design and the approach we use. Since we select only one best solution, not all transitions can be reached in the diagrams with the final state (i.e. an infeasible transition). Our future work will concentrate on fitness value. The fitness value should be calculated from the set of transitions, not only from the individual transitions. We will also like to extend this approach to the other UML diagrams; for example UML class diagram, can be merged with our HLPN to generate more flexible and usable HLPN.

REFERENCES


