Translating Activity Diagram from Duration Calculus for Modeling of Real-Time Systems and its Formal Verification using UPPAAL and DiVinE

MUHAMMAD ABDUL BASIT UR RAHIM*, AND FAHIM ARIF**

RECEIVED ON 09.02.2015 ACCEPTED ON 14.12.2015

ABSTRACT

The RTS (Real-Time Systems) are widely used in industry, home appliances, life saving systems, aircrafts, and automatic weapons. These systems need more accuracy, safety, and reliability. An accurate graphical modeling and verification of such systems is really challenging. The formal methods made it possible to model such systems with more accuracy. In this paper, we envision a strategy to overcome the inadequacy of SysML (System Modeling Language) for modeling and verification of RTS, and illustrate the framework by applying it on a case study of fuel filling machine. We have defined DC (Duration Calculus) implementaion based formal semantics to specify the functionality of RTS. The activity diagram in then generated from these semantics. Finally, the graphical model is verified using UPPAAL and DiVinE model checkers for validation of timed and untimed properties with accelerated verification speed. Our results suggest the use of methodology for modeling and verification of large scale real-time systems with reduced verification cost.

Key Words: Formal Semantics, Formal Modeling, Real-Time System, Verification, UPPAAL, DiVinE.

1. INTRODUCTION

The software engineers graphically model the functional requirements of large scale real-time system during analysis and design phases of software development life cycle. For this purpose, they use the graphical modeling languages those are SysML and UML (Unified Modeling Language). However these modeling languages are not very suitable for modeling of critical systems due to their limited support. The graphical modeling of large scale embedded and real-time systems are really challenging due to system's complex nature. An error in its model can cause loss of money or even loss of precious lives. Therefore, these systems must be formally specified and analyzed in the earlier phase of system development cycle in order to avoid such loss.

SysML is a graphical modeling language that is used to model the industrial applications [1]. The software engineers graphically model the industrial applications as per gathered user's requirements. These user requirements must be validated and verified in earlier design phase otherwise the errors found in later phases will cause failure of software or it will increase software's cost. Moreover, the errors in industrial applications can also cause loss of precious human lives. The SysML

* Military College of Signals, National University of Science and Technology, Islamabad/National Institute of Electronics, Islamabad.
** National Institute of Electronics, Islamabad.

consists of different behavioral and interactive diagrams. The activity diagram is one of the behavioral diagram of SysML however it is not rich enough to model the RTS and concurrent systems. It is difficult to define the type of flows, controls and other elements w.r.t. RTS.

The industry uses RTS to provide services and manufacture high-quality products with optimized production processes [2]. These are also used in home appliances, automatic vehicles, trains, and signaling system. Such systems require high degree of precision which is possible through formal methods.

Duration Calculus is used to formally model the functionality of real-time systems and many authors have generated graphical models (automata or state chart diagram) from these formal models for verification of functionality of RTS [3]. Such formal models are more reliable and useful to model the functionality of discrete time based RTS. The activity diagram is used to describe the functionality in a more prescribed way that is why it is more suitable to model the functionality of concurrent and component based application. However, the activity diagram has more elements as compared to automata or state chart diagram that is why it is typical to generate the activity diagram from *DC implementable* based formal model. Therefore, it is desirable to update the formal semantics that may be more useful to generate more SysML based diagrams with more accuracy.

The verification and validation is an important phase of software development life cycle. Before the modern tools and techniques, the verification and validation of requirements were only possible after the development of the software. Now different model checking tools are available which are useful to validate and verify the user requirements in earlier design phase that reduce the chances of software failure rate and also save from loss of precious human lives [4].

1.1 Motivation

The real motivation of this paper is an accurate modeling of industrial application and its verification in earlier design phase to prevent the application from later hazards. For this purpose, we have adopted a methodology to formally specify the user requirements using *Duration Calculus implementables* and generate the activity diagram from these formal semantics. The generated activity diagram is further verified using a discrete time based model checker (UPPAAL) and a parallel model checker (DiVinE) to verify the timed and untimed properties of real-time system. The DiVinE model checker is used to verify the case study with an accelerated speed of verification [5].

1.2 Framework

The proposed methodology is more useful for modeling and verification of real-time systems. The methodology is also helpful to formally define the functionality of the system and generated more accurate graphical model from formal specification. The activity diagram is more important diagram as it is capable to model the complete functionality of individual devices and also the interaction among the devices. DC is more useful for RTs, for that reason, the patterns of DC implementables (initialization, bounded stability, unbounded stability, synchronization, progress) are used to describe the functionality of RTS application. The UPPAAL model checking tool is used to verify the generated activity diagram against the functional and non-functional requirements specified in ECTL temporal format. The selected tool is more useful to show the step-by-step functionality of the system where the temporal property is satisfied and a counter example to describe the state where the temporal property does not satisfy. Our results prove that the formal methods show more accurate results then graphical modeling as there is still issues of understanding [4], concurrency and specifying the behavior of RTS w.r.t. time.

1.3 Paper organization

The remainder of the paper contains following: Section 2 presents the related work. Section 3 describes the *DC implementables*. Section 4 demonstrates the proposed formal semantics for elements of activity diagram. Section 5 shows the case study and Section 6 presents the analysis of result and finally Section 7 is the conclusion round trip of the paper.

2. RELATED WORK

Hansen, et. al. [6] have introduced the formal syntax and semantics for DC and proved its correctness and completeness. The methodology is still considered more useful due to its rich formal semantics; however, it is only useful for generating automata or state chart diagram. The other diagrams of SysML cannot be generated from these formal semantics. Pandya, et. al. [7] has proposed an extension of DC which is named as QDDC (Quantified Discrete-Time Duration Calculus) which is more useful to specify the functionality of RTS using automata only. The methodology is not useful to generate the other diagram of SysML. Hansen, et. al. [8] have proposed an algorithm for model checking of DC based formal model. A very complex mathematical modeling is involved for modeling and verifying the functionality of RTS. Guelev, et. al. [9], have proposed semantics to capture the probabilistic requirement of RTS. The authors only focus on reasoning about requirements that can be done in an infinite-interval-based system of probabilistic duration calculus. Rahim, et. al. [3], authors have proposed a methodology to formally model the functionality of RTS using DC implementables. Authors have further generated state chart diagram from formal specification and validated the functionality of the system using UPPAAL and DiVinE model checkers to reduce the verification time. However, the defined semantics are not suitable for activity diagram.

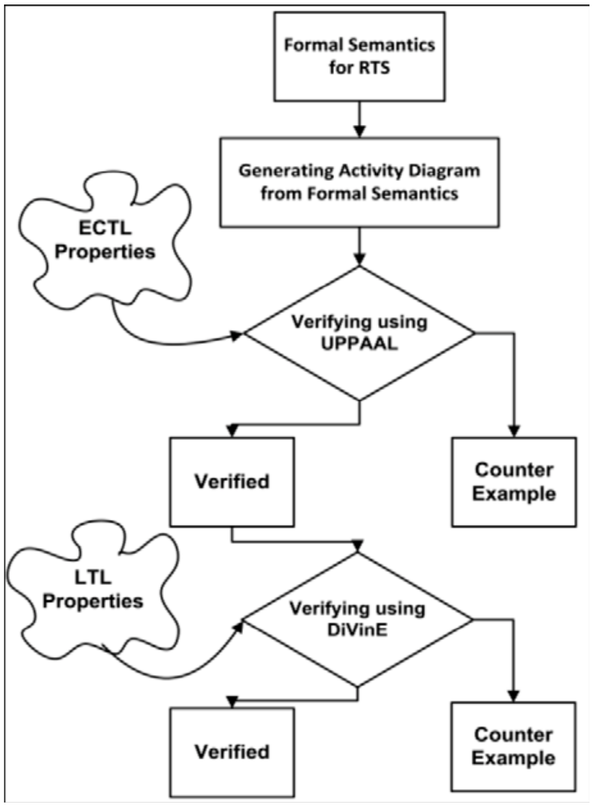

FIG 1. PROPOSED METHODOLOGY

Ouchani, et. al. [10-13], authors have verified the activity diagram using PRISM model checking tool. The authors have defined semantics using NuCalculus for all elements of activity diagram. The functionality of the system is verified against PCTL temporal properties using PRISM model checker. The authors have mainly focused probability aspect for modeling the real-time system and performed the validation using sequential model checkers.

In our previous research [4-5,14-17], , we proposed methodologies for verification of SysML by accelerating the verification speed and reducing the verification cost using parallel model checker. The authors have classified the timed and untimed properties and validated the functionality against these requirements using UPPAAL and DiVinE model checkers. However, the SysML diagrams are not formally specified in our earlier work that is why we have initially specified only the activity diagram.

We have defined DC formal semantics for all elements of activity diagram. These semantics define the type of flow among the activities that can be synchronization, progress, stability, bounded stability, unbounded stability, bounded initial stability, or unbounded initial stability. The activity diagram translated from DC implementables is more illustrative, comprehensive and reliable. The software engineers can easily model a critical large scale system. In related work, authors performed the validation using sequential model checkers that take more time for verification of large scale system and increase the verification cost. However, the proposed framework accurately models the RTS and verify the requirements in less time.

3. DC IMPLEMENTABLES

DC implementables are the subset of DC formula and these are certain patterns of DC formula that are more suitable for specifying the behavior [15]. We have used these patterns for generating activity diagram of real-time system.

In the following definitions of implementables, the $x,...,x_n$ are considered as phases where $n \geq 0$, α is considered as assertion and ε denotes as rigid.

(i) Definition (Initialization)

$$\lceil\,\rceil v \lceil x \rceil;\ true \tag{1}$$

Equation (1) describes that each control automaton is either empty or in phase x.

(ii) Definition (Bounded Stability)

$$\lceil \neg x \rceil ; \lceil x \wedge \alpha \rceil \xrightarrow{\leq \varepsilon} \lceil x \vee x_1 \vee ... x_n \rceil. \tag{2}$$

Equation (2) illustrates that when the control changes its phase to x with the condition á being true and the time since this change not exceeding ε seconds, it stays in x or it moves to one of phases $x,....,x_n$.

(iii) Definition (Unbounded Stability)

$$\lceil \neg x \rceil ; \lceil x \wedge \alpha \rceil \longrightarrow \lceil x \vee x_1 \vee ... x_n \rceil. \tag{3}$$

Equation (3) illustrates that when the control changes its phase to x with the condition α being true, it stays in x or it moves to one of phases .

(iv) Definition (Synchronization)

Equation (4) expresses that the control stayed for ε time units in phase x and with true condition.

(v) Definition (Progress)

Equation (5) demonstrates that the control stayed for ε seconds in phase x, it leaves this phase and progress accordingly.

4. PROPOSED FORMAL SEMANTICS FOR ELEMENTS OF ACTIVITY DIAGRAM

In this section, we have proposed the formal semantics for activity diagram which are based on *duration calculus implementables and it is the main contribution of this paper*.. As the DC Implementable are used to define the functionality of individual modules and its type of interaction among other modules that is why it is more efficiently utilized with activity diagram.

The formal semantics for an action is defined using ceiling brackets $\lceil\,\rceil$ that contains the name of action e.g. $\lceil$Ready$\rceil$. In the proposed formal semantics, the flow of control from an action to another action is also depicted as an arrow, moreover, this arrow is also used in different capacity for real-time perspective. All the patterns of activities are tagged on left side of pattern and a coloun (:) is marked at the end of each tag (*Init*: $\lceil\,\rceil \vee \lceil x \rceil$; *true*,). The initial node of activity diagram is defined as *Init*., however, an action node which is the final action node of the activity is tagged as fin e.g.

($Cn-Flow-Fin: \lceil x_m \rceil \longrightarrow \lceil x_n \rceil.$) where Cn-Flow presents the flow or node type and *Fin* describes it as a final activity (end of activity diagram). The control flow node is described in detail in next section. The flow final tag is described as FFin e.g. ($Cn-Flow-Fin: \lceil x_m \rceil \longrightarrow \lceil x_n \rceil.$) The action constraint node supports most of the DC implementables as this node may contain the timed or untimed constraints. There are two ways to describe the patterns for decision, merge, and fork nodes. If the time interval is not involved then the formal pattern can described on a single line, however, if time interval is involved then the patterns are defined on multiple lines. The conjunction (^) and disjunction (V) are used to define the patterns for merge, join and fork nodes. The exception handler and interruptible region along with its actions are described using tags. However, partition and their respective actions are described by mentioning both name in ceiling brackets ($\lceil$ *Card Read.Read Data* $\rceil$). For sending signal, the channel (!) is used to define the formal semantic e.g $\lceil Card\ Read.Send\ Data \rceil \xrightarrow{!\varepsilon} \lceil Card\ Read.Send\ Data \rceil.$ DC implemenable based formal semantics for individual node are defined in detail in next section.

4.1 Initial Node

An initial node is depicted by round cornered-rectangle that is preceded by a black spot. The system starts its functionality through this action.

Formal Semantic for Initial Node.

$$\lceil \rceil \vee \lceil x \rceil; true, \tag{6}$$

Equation (6) expresses that, initially, each control is either empty or in phase x.

Example: As the initial node is depicted by a filled black circle and an action node is depicted by a round-cornered rectangle, however in formal semantics, the action node connecting with initial node is considered as initial activity. In Fig 2, the initial activity "ready" is graphically presented and its formal semantics is defined in Table 1 which states that either the control is initially in ready state or it is not staying in any state.

4.2 Action Constraint Node

In activity diagram, an action constraint is depicted as dotted arrow that contains the timed or untimed constraint. It provides support for following DC implementables: progress, bounded stability, unbounded stability and synchronization.

Formal Semantic for Action Constraint: The following patterns from Equations (7-10) express the possible functionality of action constraints.

$$Prog: \quad \lceil x \rceil \xrightarrow{\varepsilon} \lceil \neg x \rceil. \tag{7}$$

or

$$Bd-Stab: \quad \lceil \neg x \rceil; \lceil x \wedge \alpha \rceil \xrightarrow{\leq \varepsilon} \lceil x \vee x_1 \vee \ldots x_n \rceil. \tag{8}$$

or

$$Ub-Stab: \quad \lceil \neg x \rceil; \lceil x \wedge \alpha \rceil \longrightarrow \lceil x \vee x_1 \vee \ldots x_n \rceil. \tag{9}$$

or

$$Sync: \quad \lceil x \wedge \alpha \rceil \longrightarrow \lceil \neg x \rceil. \tag{10}$$

Example-1: Table 2 shows the implantation of action constraint node along with *progress DC implementable* where the control flows from *Ready* to *Switch On Valve* after staying 10 time units in *Ready* action node. Fig. 3 is graphical presentation of defined formal semantics in Table2.

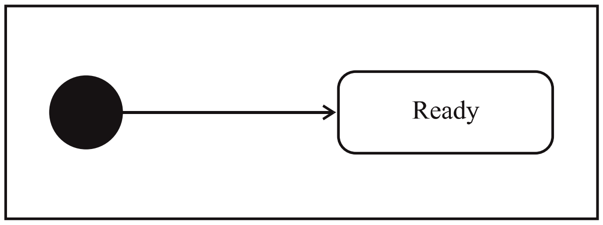

FIG. 2. INITIAL NODE

TABLE 1. FORMAL SEMANTIC FOR INITIAL NODE

$Init: \lceil \rceil \vee \lceil Ready \rceil; true,$

Example-2: Table 3 shows the implantation of action constraint node along with *bounded stability DC implementable* where the control flows from *Prompt for Amount* to *Insert Amount.* The action node *Prompt for Amount* also contains a constraint and on satisfaction of this constraint, the control stays maximum up to 10 time units in *Ready* action node and . Fig. 4 is graphical presentation of defined formal semantics in Table 3.

Example-3: Table 4 shows the implantation of action constraint node along with *unbounded stability DC implementable* where the control flows from *Prompt for Amount* to *Insert Amount* on satisfaction of constraint. Fig. 5 is graphical presentation of defined formal semantics in Table 4.

Example-4: Table 5 shows the implantation of action constraint node along with *synchronous stability DC implementable* where the control flows from *Prompt for Amount* to *Insert Amount.* The action node *Prompt for Amount* also contains a constraint and when this constraint satisfies then control stays for a fix time (10 time units) in *Ready* action node. Fig. 6 is graphical presentation of defined formal semantics in Table 5.

TABLE 2. FORMAL SEMANTIC FOR PROGRESS BASED ACTION CONSTRAINT NODE

$$Prog : \lceil Ready \rceil \xrightarrow{10} \lceil Switch\ On\ Valve \rceil .$$

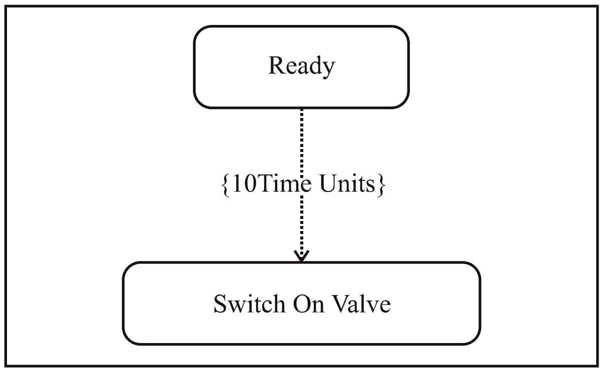

FIG. 3. PROGRESS BASED ACTION CONSTRAINT NODE.

TABLE 3. FORMAL SEMANTIC FOR BOUNDED STABILITY BASED ACTION CONSTRAINT NODE

$$Bd - Stab : \lceil \text{Re}ady \wedge amount < 10000 \rceil \xrightarrow{\le 10} \lceil SwitchOnValve \rceil .$$

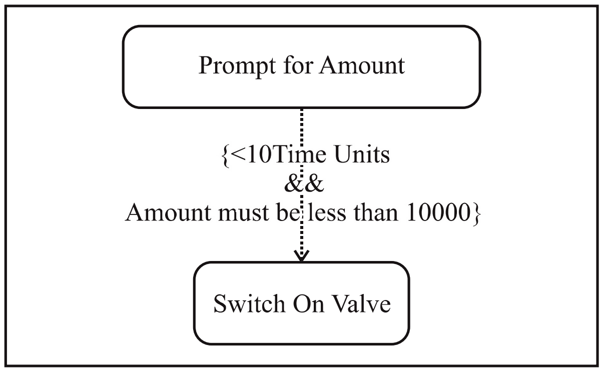

FIG. 4. BOUNDED STABILITY BASED ACTION CONSTRAINT NODE

TABLE 4. FORMAL SEMANTIC FOR UNSTABILITY BASED ACTION CONSTRAINT NODE

$$Ub - Stab : \lceil \text{Re}ady \wedge amount < 10000 \rceil \longrightarrow \lceil SwitchOnValve \rceil .$$

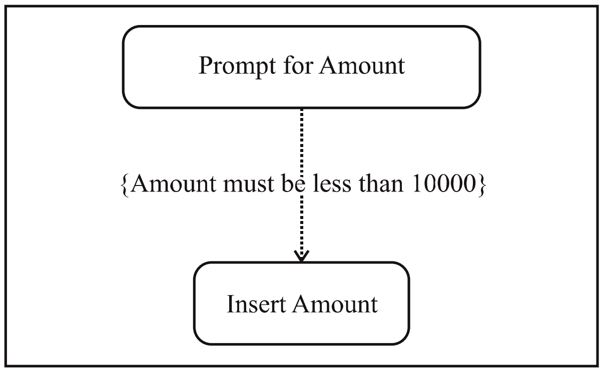

FIG. 5. UNSTABILITY BASED ACTION CONSTRAINT NODE

TABLE 5. FORMAL SEMANTIC FOR SYNCHRONIZE BASED ACTION CONSTRAINT NODE

$$Sync : \lceil \text{Re}ady \wedge amount < 10000 \rceil \xrightarrow{10} \lceil SwitchOnValve \rceil .$$

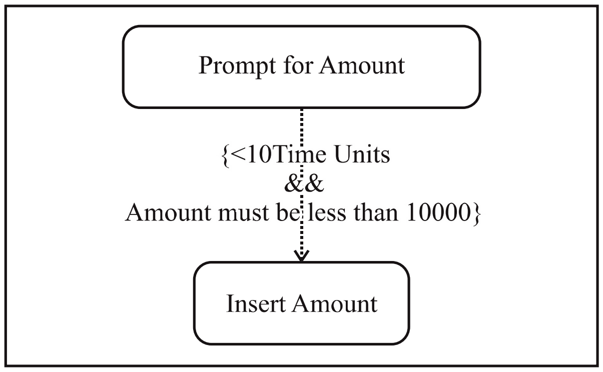

FIG. 6. SYNCHRONIZE BASED ACTION CONSTRAINT NODE

5. CONTROL FLOW NODE

It simply shows the flow of control from one action to subsequent action and it is depicted as an arrow in activity diagram.

Formal Semantic for Action Constraint.

$$\lceil x \rceil \longrightarrow \lceil \neg x \rceil \qquad (11)$$

Equation (11) expresses that if the control is in phase x then it subsequently stays in x or moves to one of the other phase.

Example: The following example simply presents the flow of control from one action node (*Prompt for Pin Code*) to other action node (*Pin Code Inserted*). The Table 6 shows the formal semantics for given example and Fig. 7 is its graphical presentation.

5.1 Decision Node

The control flows coming away from a decision node contain conditions which allow control to flow if the condition is satisfied. The decision node is depicted as diamond.

TABLE 6. FORMAL SEMANTIC FOR CONTROL FLOW NODE

$Cn-Flow: \lceil Prompt\ for\ Pin\ Code \rceil \longrightarrow \lceil Pin\ Code\ Inserted \rceil.$

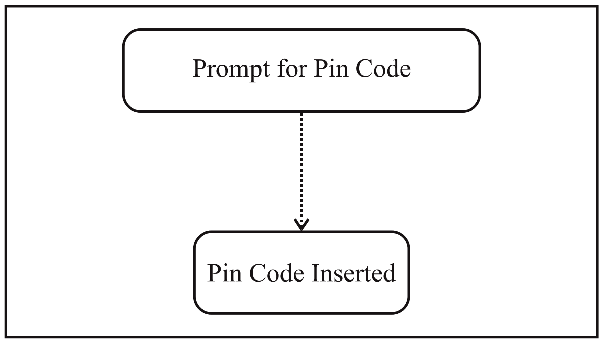

FIG 7. CONTROL FLOW NODE

Formal Semantic for Action Constraint.

$$\lceil x \wedge \alpha \rceil \longrightarrow \lceil x_m \rceil \vee \lceil x_{m+1} \rceil. \qquad (12)$$

Equation (12) expresses that if the condition á is satisfied then control will flow to phase x_m otherwise control will flow to x_{m+1} where m is a discrete number and m < n. The decision node can also be used for verification of time intervals, for that reason, the pattern will be expressed explicitly which is as under:

$$\lceil x \rceil \xrightarrow{>\varepsilon} \lceil x_m \rceil, \qquad (13)$$
$$\lceil x \rceil \xrightarrow{<\varepsilon} \lceil x_{m+1} \rceil.$$

The described pattern in Equation (13) is the time based decision node that also looks like the same pattern for constraint flow. The only difference between patterns of constraint flow node and decision node is the difference of number of flows coming away from their respective node as the decision node provides support for more than one flow coming away from decision node, however, there is always one constraint flow coming away from action node.

Example: The following example presents the conditional flow of control from an action node *Pin Code Inserted.* In the following example if control stays more than ten time units then control flows to *Eject Card* otherwise it flows to *Validate Pin*. Table 7 describes the formal semantics for decision node and Fig. 8 is the graphical presentation of described formal semantics.

5.2 Merge Node

Merge node is also graphically presented as a diamond. The control flows from merge node as it receives control from one or all connected nodes.

Formal Semantic for Action Constraint.

$$\lceil x_m \rceil \vee \lceil x_{m+1} \rceil \longrightarrow \lceil x_{m+2} \rceil. \qquad (14)$$

The Equation (14) expresses that the control moves to phase x_{m+2} when control comes away from x_m or x_{m+1} where m is a discrete number and m < n. The merge pattern can also be explicitly mentioned which is expressed using Equation (15):

$$\lceil x_m \rceil \longrightarrow \lceil x_{m+2} \rceil, \qquad \lceil x_{m+1} \rceil \longrightarrow \lceil x_{m+2} \rceil. \tag{15}$$

Example: The following example presents the unconditional flow of control to action node *Prompt for Pin Code* as merge node receive flow from one or both of action nodes *Read Data* and *Invalid Pin Code*. Table 8 describes the formal semantics for merge node and Fig. 9 is the graphical presentation of described formal semantics.

5.3 Join Node

Join node is depicted as a thick black bar which is connect with more than one incoming flows and one coming away flow. The join synchronizes two incoming flows and generate a single outflow and this outgoing flow cannot execute until all inflows have been received.

Formal Semantic for Action Constraint.

$$\lceil x_m \rceil \wedge \lceil x_{m+1} \rceil \longrightarrow \lceil x_{m+2} \rceil. \tag{16}$$

TABLE 7. FORMAL SEMANTIC FOR DECISION NODE

$Bd-Stab-Dec:$	$\lceil Pin\ Code\ Inserted \rceil \xrightarrow{<10} \lceil Validate\ Pin \rceil,$
$Bd-Stab-Dec:$	$\lceil Pin\ Code\ Inserted \rceil \xrightarrow{>10} \lceil Eject\ Card \rceil.$

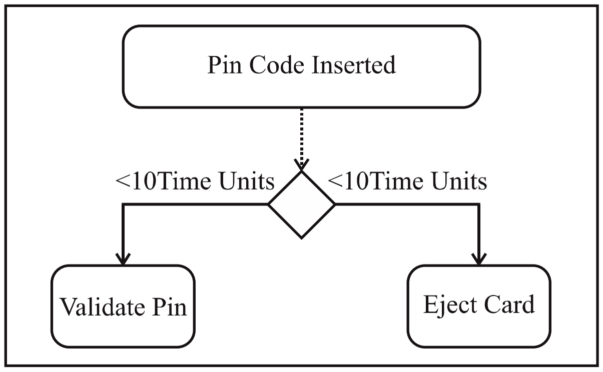

FIG. 8. DECISION NODE

The Equation (16) expresses that the control moves to phase x_{m+2} when control comes away from both x_m and x_{m+1} where m is a discrete number and m < n.

Example: An example is presented in Fig. 10 where control stays for ten time units in action node *Switch On Valve 1* and twenty time units in second in action node *Switch On Valve 2*. The control flows from *Switch On Valve 1* to join node and the join node waits for other incoming before executing the flow toward action node *Mix Both Chemical*. Table 9 describes the formal semantics for join node and Fig 10 is the graphical presentation of described formal semantics.

5.4 Fork Node

It looks like a join node however there is a difference of incoming and outgoing flows. The fork node is used for concurrent processing for that reason it has one incoming flow and multiple outgoing flows.

Formal Semantic for Action Constraint.

$$\lceil x_m \rceil \longrightarrow \lceil x_{m+1} \rceil \wedge \lceil x_{m+2} \rceil. \tag{17}$$

TABLE 8. FORMAL SEMANTIC FOR MERGE NODE

$Mrg: \lceil Read\ Data \rceil \vee \lceil Invalid\ Pin\ Code \rceil \longrightarrow \lceil Prompt\ for\ Pin\ Code \rceil.$
OR
$Mrg: \lceil Read\ Data \rceil \longrightarrow \lceil Prompt\ for\ Pin\ Code \rceil,$
$Mrg: \lceil Invalid\ Pin\ Code \rceil \longrightarrow \lceil Prompt\ for\ Pin\ Code \rceil.$

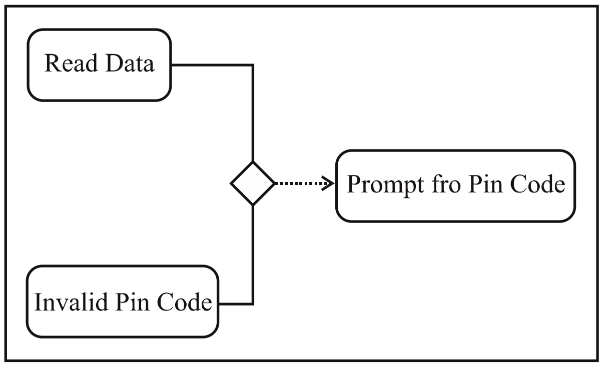

FIG. 9. MERGE NODE

This Equation (17) describes the concurrency that the control moves to phase x_{m+1} and x_{m+2} when control comes away from x_m where m is a discrete number and m < n. If time factor is involved then fork pattern can also be explicitly mentioned which is as under in Equation (18):

$$\begin{aligned} &\lceil x_m \rceil \xrightarrow{\varepsilon} \lceil x_{m+1} \rceil. \\ &\lceil x_m \rceil \xrightarrow{\varepsilon} \lceil x_{m+2} \rceil. \end{aligned} \tag{18}$$

Example: The following example describes the formal semantics and graphical presentation of fork node. In this example, the fork node has one inflow *Mixing Both Chemical* and two outflows which are *Filling Bottle 1 (Switch On Valve 3)* and *Filling Bottle 2 (Switch On Valve 4).* The control stays for ten time units in inflow of fork and both outflows are the concurrent activities which starts at the time. Table 10 describes the formal semantics for fork node and Fig. 11 is the graphical presentation of described formal semantics.

5.5 Exception Handler Node

It is used to attach with an activity where the exception handling is required. It is depicted as an angled arrow.

TABLE 9. FORMAL SEMANTIC FOR PROGRESS BASED JOIN NODE

$$\begin{aligned} &Prog-Join: \ \lceil Switch\ On\ Valve\ 1 \rceil \xrightarrow{10} \lceil Mix\ Both\ Chemical \rceil, \\ &Prog-Join: \ \lceil Switch\ On\ Valve\ 2 \rceil \xrightarrow{20} \lceil Mix\ Both\ Chemical \rceil. \end{aligned}$$

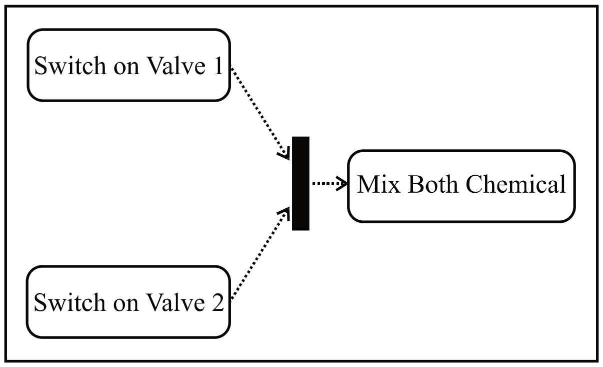

FIG 10. PROGRESS BASED JOIN NODE

Formal Semantic for Action Constraint.

$$\lceil x \rceil \xrightarrow{\sim} \lceil \neg x \rceil. \tag{19}$$

The pattern of Equation (19) expresses that the control flows to exception handler as an exception occurs in phase x.

Example: The example presents exception handling for an action node *Pin Code Validation* where an action node *Error State* is plays a role as an exception handler. Table.11 describes the formal semantics for fork node and Fig. 12 is the graphical presentation of described formal semantics.

5.6 Interruptible Active Region

A region that contains multiple activities can be interrupted at any time. It is depicted with dotted or dashed boundaries.

Formal Semantic for Action Constraint.

$$\lceil x \rceil \xrightarrow{\varepsilon} \lceil \neg x \rceil. \tag{20}$$

The Equation (20) illustrates that the control moves to interruptible region as an interrupt occurs in phase x.

TABLE 10. FORMAL SEMANTIC FOR FORK NODE

$$\begin{aligned} &Prog-Frk: \lceil Mixing\ Both\ Chemical \rceil \xrightarrow{10} \lceil Filling\ Bottle\ 1\ (Switch\ On\ Valve\ 3) \rceil, \\ &Prog-Frk: \lceil Mixing\ Both\ Chemical \rceil \xrightarrow{10} \lceil Filling\ Bottle\ 2\ (Switch\ On\ Valve\ 4) \rceil. \end{aligned}$$

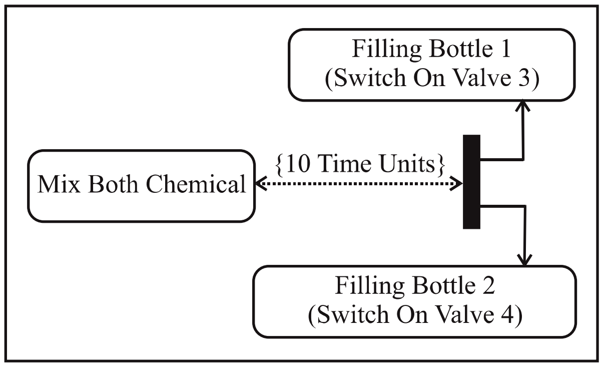

FIG. 11. PROGRESS BASED FORK NODE

Example: The example shows that the plant will mix two chemicals till and then it will close mixing. An interrupter *Cancel Mixing Request* can interrupt the mixing process during mixing and control can not stay for more then five time units in *Cancel Mixing Request*. Table 12 describes the formal semantics for fork node and Fig. 13 is the graphical presentation of described formal semantics.

5.7 Partition Node

Partition is a swim line that differentiates the working of individual parts a module.

Formal Semantic for Action Constraint.

$$\lceil y.x \rceil \longrightarrow \lceil z.\neg x \ \rceil \ . \qquad (21)$$

TABLE 11. FORMAL SEMANTIC FOR EXCEPTION HANDLER NODE

EXC: $\lceil$ PIN CODE VALIDATION $\rceil \xrightarrow{\leadsto} \lceil$ ERROR STATE $\rceil$.

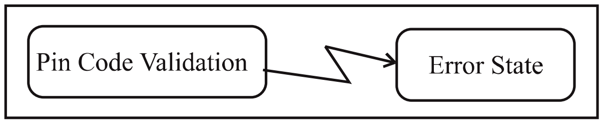

FIG. 12. EXCEPTION HANDLER NODE

The Equation (21) describes that the control moves from phase x of y partition to some other phase of z partition.

Example: In the following example, the functionality of an input device (Card Reader) and a controller (Control Unit) is presented. Initially, the card reader reads the data available on card and then control unit prompts for enter amount. The activity of individual part is defined by conncating it with parts name e.g. *Card Reader. Read Data* where *Card Reader* is the device name and *Read Data* is the name of its activity. Table 13 describes the formal semantics for fork node and Fig. 14 is the graphical presentation of described formal semantics.

5.8 Send Signal Node

It is an action that is used to send data from one activity to other activity.

Formal Semantic for Action Constraint.

$$\lceil x \rceil \xrightarrow{!} \lceil \neg x \rceil \ . \qquad (22)$$

The Equation (22) shows that the control moves from phase x to some other phase by sending a signal.

TABLE 12. FORMAL SEMANTIC FOR INTERRUPTIBLE ACTIVE REGION NODE

$Init - I - Reg:$	$\lceil \ \rceil \vee \lceil Request\ for\ Mixing\ Both\ Chemicals \rceil; true,$
$Cn - Flow - I - Reg:$	$\lceil Mixing\ Both\ Chemicals \rceil \longrightarrow \lceil Mixing\ Both\ Chemicals \rceil,$
$Intr - I - Reg:$	$\lceil Cancel\ Mixing\ Request \rceil \xrightarrow{<5} \lceil Cancel\ Mixing\ Both\ Chemicals \rceil,$
$Bd - Stab - Fin:$	$\lceil Cancel\ Mixing\ Both\ Chemicals \rceil \longrightarrow \lceil Fin \rceil,$
$Fin:$	$\lceil Mixing\ Both\ Chemicals \rceil.$

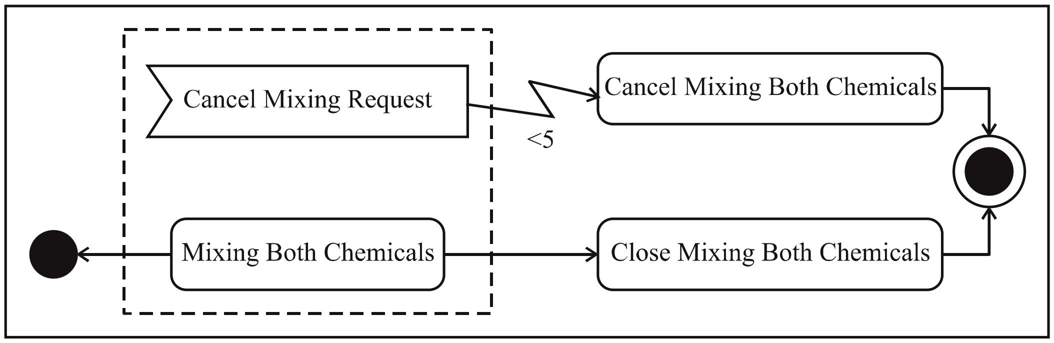

FIG. 13. INTERRUPTIBLE ACTIVE REGION NODE

Example: The example presented in previous section is extended to describe the functionality of send signal, as shown in Fig. 15. Initially, the card reader reads the data and sends it to the control units. The control unit receives the data and further prompts for amount. In this example, the channel (!) used to present communication among the devices that contains a message (*data)* as well, as shown in Table 14 and graphically in Fig 15.

6. CASE STUDY

The fuel filling machine contains following devices: keypad, card reader, display screen, control unit, and filler. When the customer inserts card in card reader then the card reader takes ten time units to read information available on card. The display screen prompts to insert pin code using keypad and then prompts for amount if pin code is valid otherwise prompts for incorrect pin code. On validation of pin code, the display screen prompts to insert amount using keypad. The control unit takes ten time units for validation of pin code. The control unit keeps record of invalid attempts and ejects the card on three consecutive invalid attempts. The control unit again validates the amount in maximum ten time units. After successful validation of amount, the control unit switches on the filler for desired amount otherwise prompts again to insert the amount. If user has inserted the card and he is unable to insert pin or amount within ten time units, in either case, the control unit will eject the card. Fig. 16 shows the functionality of fuel filling machine that contains two inputs, two outputs and a controller. The input devices communicate with controller by sending respective data and then controller controls the output devices accordingly.

Table 15 shows the formal semantics to describe the functionality of the fuel filling machine. The semantics define the initial and final actions of fuel filling machine. Moreover, the flow of control is also shown that is either bounded stable or unbounded stable. The label on left side of semantics also shows the use of node to represent the functionality.

Fig. 16 is the activity diagram generated from semantics defined in Table 15 under the rules defined in section 5. The dotted line shows the time of flow of control from one node to other node. The DC implementable based activity diagram more precisely represents the functionality of real-time systems.

TABLE 13. FORMAL SEMANTIC FOR PARTITION NODE

$$\begin{aligned}
&Init: && \lceil\,\rceil \vee \lceil Card\ Reader.\ Read\ Data \rceil;\ true, \\
&Cn\text{–}Flow: && \lceil Card\ Reader.\ Read\ Data \rceil \longrightarrow \\
& && \lceil Control\ Unit.\ Prompt\ for\ Amount \rceil.
\end{aligned}$$

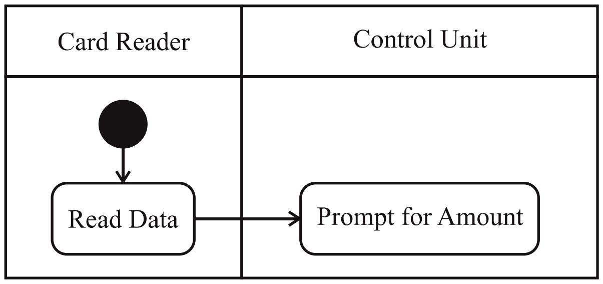

FIG. 14. PARTITION NODE

TABLE 14. FORMAL SEMANTIC FOR SEND SIGNAL NODE

$$\begin{aligned}
&Init: && \lceil\,\rceil \vee \lceil Card\ Reader.\ Read\ Data \rceil;\ true, \\
&Cn-Flow: && \lceil Card\ Reader.\ Read\ Data \rceil \longrightarrow \lceil Control\ Unit.\ Send\ Data \rceil, \\
&Sgnl: && \lceil Card\ Reader.\ Send\ Data \rceil \xrightarrow{data!} \lceil Control\ Unit.\ Receive\ Data \rceil, \\
&Cn-Flow: && \lceil Control\ Unit.\ Receive\ Data \rceil \longrightarrow \lceil Control\ Unit.Prompt\ for\ Amount \rceil.
\end{aligned}$$

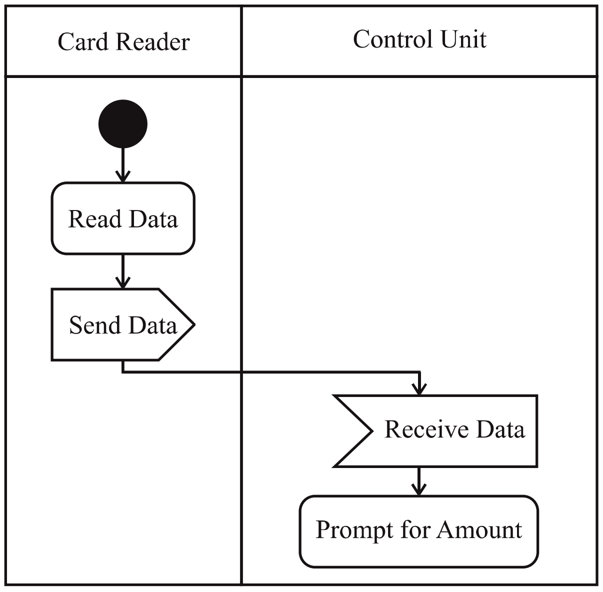

FIG 15. SEND SIGNAL NODE

TABLE 15. FORMAL SEMANTIC FOR FUEL FILLING MACHINE

$Init:$	$\lceil\ \rceil \vee \lceil Card\_Reader.Ready \rceil; true,$
$Bd.Stab-Cn-Flow:$	$\lceil Card\_Reader.Card\_Inserted \rceil \xrightarrow{<10} \lceil Card\_Reader.Read\_Data \rceil,$
$Cn-Flow-Mrg:$	$\lceil Card\_Reader.Read\_Data \rceil \longrightarrow \lceil Control\_Unit.Prompt\_for\_Pin \rceil,$
$Cn-Flow:$	$\lceil Control\ Unit.Prompt\ for\ Pin \rceil \longrightarrow \lceil Display\ Screen.Prompt\ For\ Pin \rceil,$
$Cn-Flow:$	$\lceil Display\ Screen.Prompt\ for\ Pin \rceil \longrightarrow \lceil Keypad.Pin\ Code\ Inserted \rceil,$
$Dec:$	$\lceil Keypad.Pin\ Code\ Inserted\ \wedge Validatepin \rceil \longrightarrow \lceil Display\ Screen.Prompt\ for\ Amount \rceil,$
$Dec-Dec-Mrg:$	$\lceil Keypad.Pin\ Code\ Inserted \wedge !validatepin\ No.\ of\ att < 3 \rceil \longrightarrow \lceil Control\ Unit.Prompt\ for\ Pin \rceil,$
$Dec-Dec:$	$\lceil Keypad.Pin\ Code\ Inserted\ \wedge !validatepin\ No.\ of\ att > 3 \rceil \longrightarrow \lceil Control\ Unit.Card\ Ejected \rceil,$
$Cn-Flow:$	$\lceil Display\ Screen.Prompt\ for\ Amount \rceil \longrightarrow \lceil Keypad.Amount\ Inserted \rceil,$
$Dec-Bd.Stab-Dec:$	$\lceil Keypad.Amount\ Inserted\ \wedge validateamount \rceil \xrightarrow{>10} \lceil Control\ Unit.Card\ Ejected \rceil,$
$Dec-Bd.Stab-Dec-Fin:$	$\lceil Keypad.Amount\ Inserted\ \wedge validateamount \rceil \xrightarrow{<10} \lceil Filler.Filling \rceil.$

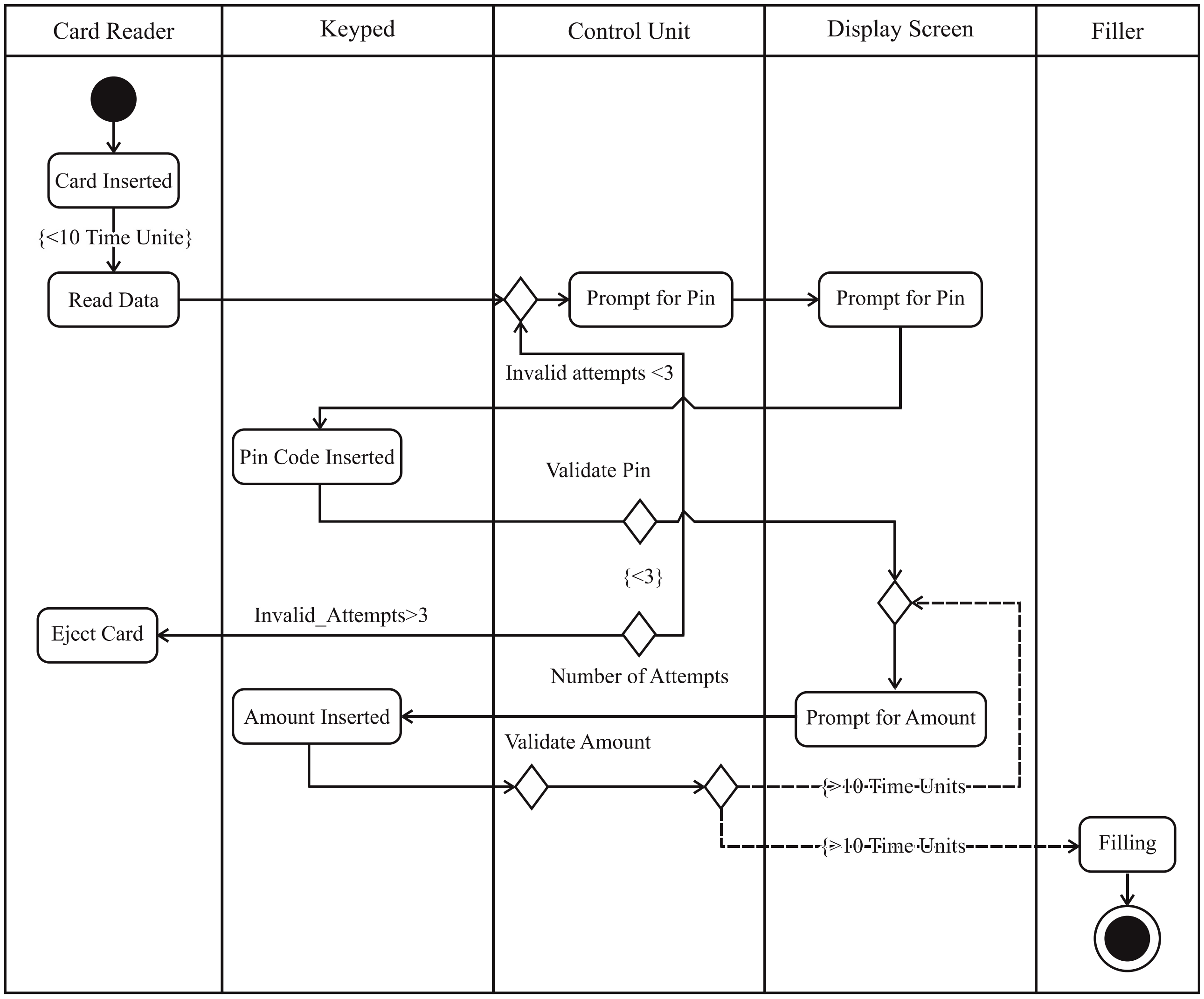

FIG. 16. ACTIVITY DIAGRAM OF FUEL FILLING MACHINE

6.1 Properties

This section present the properties using duration calculus and its equivalent temporal logic format (extended computational tree logic) that is used by model checker for validation of properties. These properties are verified against the functionality of the system and model checker is used for this purpose which analyzes the functionality at individual transaction level. This individual transaction can also be seen using simulator of model checkers. The model checker also shows the counter example if a property is not satisfied against the system's functionality. The following are the properties which are formally specified and verified against the functionality of the system.

P1 The card reader can take maximum ten time units to read the card's data.

P2 The control unit takes maximum ten time units to validate entered amount.

P3 If user enters the amount in ten time units then start filling.

P4 If user is unable to enter the amount in ten time units then eject the card.

The functionality of the case study is formally specified using DC implementables and the properties are specified using DC formulas. The previous section shows the properties which are verified against the functionality of the case study. Table16 presents the both formats of requirement specification which are DC formula and its ECTL (Equivalent Temporal Logic Format). Table 16 also illustrates the status of property that either the property is satisfied or not. All the properties specified are satisfied except the last one.

7. ANALYSIS OF RESULTS

The proposed framework is more useful in the field of software and system engineering. The software engineers mostly rely on graphical models while computational engineers depend on complex mathematical models. The graphical models are easy to understand, however, it provides limited support for modeling of RTS; therefore, we have defined the formal semantics to overcome the limitations of graphical model. Furthermore, these semantics are used to generate an activity diagram with more accuracy.

As the activity diagram is unable to describe the type of flow among the actions and also lacks in specifying the

TABLE 16. REAL-TIME PROPERTIES FOR FUEL FILLING MACHINE

Property No	Property Specification using		Status
P1	DC	$\lceil Read\_data \rceil \Rightarrow l \le 10$	Satisfied
	ECTL	$E <> card\_reader.Card\_Inserted\ and\ (card\_reader.y <= 10)$	
P2	DC	$\lceil Validate\_Amount \rceil \Rightarrow l \le 10$	Satisfied
	ECTL	$E <> controller.Validate\_Amount\ and\ (controller.z <= 10)$	
P3	DC	$(\lceil Validate\ Amount \rceil\ l < 10) \Rightarrow \lceil Filling \rceil$	Satisfied
	ECTL	$E <> controller.Validate\_Amount\ and\ (controller.z < 10)\ and\ (filler.Filling)$	
P4	DC	$(\lceil Validate\ Amount \rceil \wedge l > 10) \Rightarrow \lceil Card\ Ejected \rceil$	Not Satisfied
	ECTL	$E <> Controller.Validate\_Amount\ and\ (controller.z > 10)\ and\ (card\_reader.Card\_Inserted)$	

functionality of concurrent and component based system, for such reason, formal semantics for activity diagram have been proposed which describe the type of interactions e,g. *bounded stable*, *unbounded stable*, *synchronous* or *progress*. The proposed semantics are based on *DC implementables* which are very helpful to demonstrate the functionality of RTS. One of the major benefits of these semantics is effectively specifying the time for all nodes of activity diagram. Moreover, by using these semantics the time for concurrent actions and components based systems can be more precisely specified for individual actions. Although the activity diagram uses fork and swim line nodes for graphical modeling these systems, however, it lacks in specifying different type of time for individual action. The presented case study is a component based system that is more efficiently specified which proves its usefulness for component based applications.

The proposed formal semantics describes all type of nodes, interactions among the nodes, timed constraints and *DC implementable* based control flow. The individual pattern of proposed semantic is tagged on left side of the pattern that presents the type of control flow and the element used other than an action e.g. merge, fork, join, decision, signal, interrupt region, exception handler and control flow.

As the duration calculus is based on discrete time therefore the methodology is only useful for such systems. For verification of discrete time based systems, a discrete model checker UPPAAL is used with ECTL temporal logic. In this paper, the UPPAAL model checker is used to model the functionality of each component individually which communicates among the other components using communication channels and timed functional requirements are verified against ECTL temporal logic.

As the methodology is verified using UPPAAL model checker that is a sequential model checker and it takes more time for verification of large industrial applications. DiVinE is a parallel model checker that uses multiple processers for verification of industrial application which ultimately accelerate verification speed [5,16] . DiVinE is also compatible with UPPAAL model checker, for that reason, we have used DiVinE model checker to verify the UPPAAL's model with accelerated speed. The timed properties are verified using UPPAAL and untimed properties is verified using DiVinE model checking tool.

8. CONCLUSIONS

With increase in demand, dependency and complexity, the real-time systems are required to be more reliable and efficient. The proposed methodology is more useful to fulfill such industrial requirements with more accuracy, safety and efficiency in a very cost effective way. The methodology validates user's requirements in earlier design phase and reduces the verification cost by accelerating verification speed. In this paper, we have extended the *DC implementable* based formal semantics to effectively model the functionality of RTS and generate an accurate activity diagram. The generated diagram is further verified against time and untimed properties using UPPAAL and DiVinE model checkers. The DiVinE model checker accelerates the verification speed and saves the time for verification of large scale real-time industrial applications.

In this paper, the formal semantics have been defined for all nodes of activity diagram which are more useful to precisely describe the functionality of RTS. A component based case study of fuel filling machine is efficiently specified using defined semantics and generated more accurate activity diagram.

9. FUTURE WORK

In future, other SysML diagrams will be generated using *DC implementable* based formal semantics and a tool will be developed for specification and verification of user requirements.

ACKNOWLEDGEMENT

Authors would like to acknowledge the technical support provided by Research Centre for Modeling and Simulation, National University of Science & Technology, Islamabad, Pakistan, for implementing and running the case study.

REFERENCES

[1] Object Modeling Group, "Specificaiton System Modeling Language (OMG SysML)", Object Modeling Group, June 2012. [Online]. Available: www.omgsysml.org/INCOSE-OMGSysML-Tutorial-Final-090901.pdf. [Accessed 28 September 2014].

[2] Olderog, E.R., and Dierks, H., "Real-Time Systems - Formal Specification and Automatic Verification", Cambridge University Press, New York, 2008.

[3] Rahim, M.A.B, Arif, F., Mehmood, R., and Ahmad, J., "Modeling and Validation of Real-time Systems", Design Automation of Embedded System, pp. 18, November, 2014.

[4] Rahim, M.A.B., Ahmad, J., and Arif, F., "Parallel verification of UML using DiVinE tool", 5th International Conference on Computer Science and Information Technology, Amman, Jordan, 27-28 March, 2013.

[5] Rahim, M.A.B., Arif, F., and Ahmad, J., "Modeling of Embedded System Using SysML and Its Parallel Verification Using DiVinE Tool", Computational Science and Its Applications, Guimarães, pp. 541-555, Portugal, Springer, July, 2014.

[6] Hansen, M.R., and Chaochen, Z., "Semantics and Completeness of Duration Calculus", Real-Time: Theory in Practice, pp. 209-225, The Netherland, Springer Berlin Heidelberg, June, 1992.

[7] Pandya, P.K., "Specifying and Deciding Quantified Discrete-Time Duration Calculus Formulae using DCVALID", Proceedings of Real-Time Tools, Aalborg, 2000.

[8] Hansen M.R., Phan, A.D., and Brekling, A.W., "A Practical Approach to Model Checking Duration Calculus using Presburger Arithmetic", Annals of Mathematics and Artificial Intelligence, Volume 71, Nos. 1-3, pp. 251-278, July, 2014.

[9] Guelev, D., and Hung, D.W., "Reasoning about QoS Contracts in the Probabilistic Duration Calculus", 5th International Workshop on Formal Foundations of Embedded Software and Component-Based Software Architectures, Budapest, Hungary, June, 2010.

[10] Ouchani, S., Mohamed, O.A., and Debbabi, M., "A Formal Verification Framework for SysML Activity Diagrams", Expert Systems with Applications, Volume 41, No. 6, pp. 2713-2728, May, 2014.

[11] Ouchani, S., Mohamed, O.A., and Debbabi, M., "A Security Risk Assessment Framework for SysML Activity Diagrams", IEEE 7th International Conference on Software Security and Reliability, Gaithersburg, MD, 18-20 June, 2013.

[12] Ouchani, S., Mohamed, O.A., and Debbabi, M., "A Property-Based Abstraction Framework for SysML Activity Diagrams", Knowledge-Based Systems, Volume 56, pp. 328-343, 2014.

[13] Ouchani, S., "A Probabilistic Verification Framework for SysML Activity Diagrams", Proccedings of 11th Conference on New Trends in Software Methodologies, Tools and Techniques, Volume 24, pp. 108, 2012.

[14] Rahim, M.A.B., Arif, F., and Ahmad, J., "Modeling of Real-Time Embedded Systems using SysML and its Verification using UPPAAL and DiVinE", 5th IEEE International Conference on Software Engineering and Service Science, Beijing, China, June, 2014.

[15] Olderrod, E.R., and Dierks, H., "Real-Time Systems", Oldenburg, Cambridge University Press, 2008.

[16] Barnat, J., Brim, L., Havel, V., Havlíèek, J., Kriho, J., Lenèo, M., Roèkai, P., Štill, V., and Weiser, J., "DiVinE 3.0 - An Explicit-State Model Checker", Computer Aided Verification, pp. 863-868, Springer, 2013.

[17] Rahim, M.A.B.,, Arif, F., and Ahmad, J., "Parallel Verificaiton of UML using DiVinE Tool", 5th International Conference on Computer Science and Information Technology, Amman, Qatar, March, 2013.