2009 Fourth International Conference on Internet Computing for Science and Engineering

An Operational Semantics for UML RT-Statechart in Model Checking Context

Tao Zhang, Shaobin Huang, and Hongtao Huang
College of Computer Science and Technology
Harbin Engineering University
Harbin, Heilongjiang Province, China, 150001
zhangtaohrbeu@163.com;huangshaobin@hrbeu.edu.cn;hhtdeemail@gmail.com

Abstract**—Model checking UML statechart can detect various errors and inconsistencies of current system design in the early process of development. However, because of classic statechart lacking of real-time operational semantics, it can not be directly used to verify real-time property of current system. This paper introduces related definition of clock to extend UML statechart to real-time UML statechart, defines the main elements of statechart by tuple and proposes a middle expression form of statechart SC. Through defining the operational semantics of SC, the conversion from SC to transition system is achieved and a conversion algorithm is given. Based on the above theories , the general approach for model checking SC is described, and in the end, a classic study case of conversion from SC to TS is given.**

Keywords-modelchecking; statechart; operational semantics; transition system;real time

I. INTRODUCTION

Real-time system as a time-critical system, whose behavior is typically subject to rather stringent time constraints, can be a timely response to the occurrence of random external events, and fast enough to complete the event handling. For a train crossing control system, it is essential that on detecting the approach of a train, the gate is closed within a certain time bound in order to halt car and pedestrian traffic before the train reaches the crossing. For a radiation machine the time period during which a cancer patient is subjected to a high dose of radiation is extremely important; a small extension of this period is dangerous and can cause the patient's death. These show that in real-time system, system correctness depends not only on the accuracy of calculating, but also on the correct time of calculating. Therefore, the key to real-time system design is to ensure the specified time limits to respond.

System development must be related to modeling stage. In the process of modeling real-time system, verifying design model of system is a very important part in order to ensure real-time property and reliability.

Unified Modeling Language (UML)[1] is a visual modeling language which is used to record and exchange the system design in the system development phase. It describes the static structure and dynamic behavior of system model with chart .UML statechart which is based on the classical Harel's statechart[2] is mainly used for modeling the behavior and state changes of the object in the life cycle.

Model Checking[3] is an automatic analysis and verification technology towards finite state concurrent system in the formal verification process, which uses the method of explicit state search or implicit fixed-point computing to verify the modal or temporal property of concurrent system.

Research on the model checking method of UML statechart can detect various errors and inconsistencies of system design in the early process of development, and save a large number of financial, human and material resources for system development.

However, the existing statechart does not have real-time operational semantics, it can not get measure of time performance from the statechart, so the existing model checking method of UML statechart does not support validation of strict real-time property. To solve this problem, many researchers study the model checking method of real-time system with the method of combining UML statechart and sequence diagram. The typical studies are as follows:

Engels[4] gives a method of verifying consistency of dynamic behavior of UML model. It first identifies the consistency problem in the system model and selects an appropriate semantic domain, then gives a reasonable mapping rules, which maps UML behavioral model to the semantic domain. Finally, the problem of model consistency is analyzed and verified in this semantic domain.

Küster[5] uses UML-RT[6] to model the embedded real-time system and to verify the consistency of syntax and semantic of UML statechart and sequence diagram model, and points out that time consistency is a special type of semantic consistency.

A common method is to transform the dynamic model of UML into Timed Automata for formal verification. Firley[7] first transforms the UML sequence diagram into Timed Automata, then uses model checking tool UPPAAL[8] for formal verification of real-time system. Knapp[9] models the dynamic behavior of real-time system with the UML sequence diagram and statechart which is expanded by adding time constraints, then transforms UML sequence diagram and statechart into Timed Automata respectively. Finally it uses model checking tool UPPAAL to verify the consistency of these two models.

Li Xuan-dong[10] uses UML statechart network to model the design model for real-time system, and uses UML sequence diagram with real-time extension to model the requirement based on the scene. Finally, an algorithm to

978-0-7695-4027-6/10 $26.00 © 2010 IEEE
DOI 10.1109/ICICSE.2009.15

verify the consistency between the UML statechart and sequence diagram is given.

The above methods have the following insufficiency: first, although the UML sequence diagram can describe the time sequence of dynamic behavior of real-time system, this time relation can not be directly reflected in the state transition process of real-time system; second, the ability of UML sequence diagram to describe the time constraints is weaker than Timed Computation Tree Logic(TCTL) . Finally, the transition system obtained by the semantics of sequence diagram and statechart is very complicated, this methods greatly increase the difficulty of model checking.

In view of the above all sorts of problems, this paper proposes a method of model checking the UML statechart of real-time system. On the one hand, it introduces related definition of clock to extend UML statechart to real-time UML statechart(RTSC). By defining the real-time operational semantics of the RTSC, it is converted to transition system(TS). On the other hand, the time constraints which is described by Timed Computation Tree Logic is converted to the Timed Automata(TA) . With the obtained TS and TA as the input of model checking algorithm, the behavior of real-time system can be verified whether or not to meet the time constraints.

Furthermore, the composite states, hierarchy process and historical transformation of UML statechart destroy the modularity of model structure, the statechart can not be directly mapped to transition system, which makes model checking statechart very difficult . The current method of model checking UML statechart is first to convert statechart to middle expression form, which flattens the hierachical structure of statechart, then to convert middle expression form to transition system for model checking. For the past decade, there are many research results about this:

Li Liu-ying[11] proposes a kind of operational semantics of UML statechart and Dong Wei[12] gives a method of model checking UML statechart. The operational semantics of UML statechart is defined by searching for the largest conflict-free set of state transition for system which has infinite calculating and is mapped to Büchi automata. Zhou Ying[13] proposes an operational semantics for UML state machine in model checking context. First the syntax and static semantics of UML state machine are analyzed, then the dynamic semantics of UML state machine is defined as a transition system and related concepts and algorithms are given.

In the above-mentioned methods, most of the conversion processes of statechart only deal with the minimal subset of the composition elements of the statechart and much redundancy is introduced in the process of model checking.

The rest of the paper is organized as follows: Section 2 introduces the definition of syntax and static semantics of RTSC; Section 3 defines the operational semantics of RTSC; Section 4 introduces the method for model checking RTSC; Section 5 gives a case study on converting RTSC to TS; Finally, Section 6 concludes the paper.

II. Definition of Syntax and Static Semantics of RTSC

This paper uses tuple to express the state and transition of UML statechart. For modeling real time system by UML statechart, the related concepts of clock are introduced in the definition of elements of UML statechart.

The clocks are used to formulate the real-time assumptions on system behavior[14]:

Definition 1. Clock Constraint. A clock constraint over set C of clocks is formed according to the grammar:

$$\sigma ::= x < c \mid x \le c \mid x > c \mid x \ge c \mid \sigma \wedge \sigma$$

Where, $C \in N$ and $x \in C$. Let $CC(C)$ denote the set of clock constraint over C . Clock constraint that does not contain any conjunctions is atomic. Let $ACC(C)$ denote the set of all atomic clock constraints over C .

Definition 2. Clock Valuation. A clock valuation η for a set C of clocks is a function $\eta : C \to R_{\geq 0}$, assigning to each clock $x \in C$ its current value $\eta(x)$.

We can formally define as a clock constraint to hold for a clock valuation or not. The satisfaction relation $|=$ is a relation between clock valuation and clock constraint.

Definition 3. Satisfaction Relation for Clock Constraint. For $x \in C$, $\eta \in Eval(C)$, $c \in N$ and g, $g' \in CC(C)$, let $|= Eval(C) \times CC(C)$ be defined by:

- $\eta \mid= \text{true}$;
- $\eta \mid= \text{x<c}$ iff $\eta(\text{x})<\text{c}$;
- $\eta \mid= \text{x} \le \text{c}$ iff $\eta(\text{x}) \le \text{c}$;
- $\eta \mid= \neg\text{g}$ iff η not hold g ;
- $\eta \mid= g \wedge g'$ iff $\eta \mid= \text{g} \wedge \eta \mid= \text{g}'$.

Let η be a clock valuation on C . For positive real d , $\eta + d$ denotes the clock valuation where all clocks of η are increased by d .

Formally, $(\eta + d)(x) = \eta(x) + d$ for all clocks $x \in C$. reset x in η denotes the clock valuation which is equal to η except that clock x reset. Formally:

$$(\text{reset}\, x \text{ in } \eta)(y) = \begin{cases} \eta(y) & \text{if}\, y \neq x \\ 0 & \text{if}\, y = x \end{cases} \quad (1)$$

There are two possible ways in which a statechart can proceed: by taking a transition in the statechart, or by letting time progress while staying in a state. In the underlying transition system, the former is represented by a discrete transition and the latter by a delay transition. In the former case, the corresponding transition of the underlying transition system is labeled with the action of the transition. In the latter case, it is labeled with a positive real number indicating the amount of time that has elapsed.

The abstract syntax of UML statechart can be expressed by metamodel. The elements of abstract syntax of UML statechart contain: State machine, State, Transition, guard, event and action. They are defined in this paper as follows:

(1) State machine: $SM = (S, T, Stop, sub, type)$, where, S is a set of all states; T is a set of all transitions; $Stop$ is a state in outermost layer of SM; $sub : S \to 2^S$ denotes the subset of state s and for $\forall s \in S : Stop \notin Sub(s)$, $type : S \to \{$Basic,Or, And,Pseudo$\}$ is a type function.

(2) Basic State: $s = (sn, en, ac, ex, inv(s))$. where, sn denotes the name of state with the unique expression; en, ac, ex respectively denotes the entry, stay, exit action occurred in the state; $inv(s)$ is a time invariant which denotes the time stay in the state, i.e., State s should be leave before the $inv(s)$ become ineffective, $type(s) = \text{Basic}$.

(3) Or State: $s = (sn, (s_1, s_2, ..., s_k), l, ts, en, ac, ex, inv(s))$. Where, sn denotes the name of state, $(s_1, s_2, ..., s_k)$ is the subset of s; the active or state only has one active substate, $l \in [1, k]$ is the serial number of the active substate and its default value is 1 ; ts is the set of all transition among all the substate, $inv(s) \subseteq inv(s_1) \vee ... \vee inv(s_k)$, and $type(s) = \text{Or}$.

(4) And State: $s = (sn, (s_1, s_2, ..., s_k), en, ac, ex, inv(s))$, where sn denotes the name of state; $(s_1, s_2, ..., s_k)$ is the subset of s, all substates of the active And State are active state; $inv(s) \subseteq inv(s_1) \vee ... \vee inv(s_k)$.

(5) Least Common Ancestor (LCA) of the set of state: the LCA of the non-empty sets $U \subseteq S$ is expressed as $LCA(U)$, where $LCA(U) = u$ and $u \in S$, *iff*:

1) $U \subseteq sub(u)$;

2) $\forall s \in S : U \subseteq sub(s) \Rightarrow u \in sub(s)$.

(6) Two states $s, s' \in S$ are orthogonal, written $s \perp s'$. *iff* : $s = s'$ or $type(LCA(s, s')) = \text{And}$. If $U \subseteq S$ and for $\forall (s, s') \in U$, $s \perp s'$ holds, then U is orthogonal states set. If $s' \in sub(s)$, there are preorder relation between s and s', written $s' \le s$. If $s' \le s$ and $s' \ne s$, then $s' < s$ is a preorder relation.

(7) Transition in the statechart can be defined as follows: $\forall t \in T, t = (tn, ms, sr, event, guard, action, td, mt, D)$ w here tn denotes the name of transition; ms and mt are the main sources state and main target state of t respectively; sr is the set of source restriction and td is the set of target determinator; the $event$ specifies the single event which activates t ; the $guard$ is a boolean expression that may be attached to a transition in order to determine whether that transition is enabled or not; the $action$ denotes a set of action occurred in the transition; $D(t)$ denotes the set of the clock variables which are reset in the transition.

(8) Least Common Ancestor(LCA) of transition: for $\forall u \in S$ and $t \in T$, if $LCA(t) = u$, *iff*:

1) $sr \subseteq sub(u) \wedge td \subseteq sub(u)$;

2) $\forall s \in S : u \in sub(s) \Rightarrow sr \subseteq sub(s) \wedge td \subseteq sub(s)$.

(9) Configuration $Confi \subseteq S$ denotes the global state in State Machine. In the same time there may be many active states in state machine, which constitute a global state of State Machine, such that :

1) $\exists_1 s \in sub(Stop)$ and $s \in Confi$;

2)if $s \in Confi$ and $type(s) = \text{Or}$, then

$\exists_1 s' \in sub(s) : s' \in Confi$;

3)if $s \in Confi$ and $type(s) = \text{And}$,then

$\forall s' \in sub(s) : s' \in Confi$.

This paper uses Con_{all} to denote the set of all configuration of the State Machine, con_{cur} to denote the current configuration of the State Machine and con_{ini} to denote the initial configuration of the State Machine.

(10) $defcon$ denotes the default configuration of Composite State .The calculation rules are as follows:

1) If $type(s) = \text{Or}$, then $defcon(s) = \{sn\} \cup defcon(s_l)$;

2) If $type(s) = \text{And}$, $defcon(s) = \{sn\} \cup \{ \bigcup_{i=0}^{k} defcon(s_i) \}$.

(11) ds is the set of the innermost state of configuration, $ds(con) = \{s \mid s \in con \wedge sub(s) \cap con = \varnothing\}$.

(12) $PE(con)$ denotes the set of all possible events of configuration con , it is defined as:

$PE(con) = \{event(t) \mid sr(t) \subseteq con \wedge guard(t)\}$

(13) $enable(con, e)$ is the function which is used to calculate all enable transitions when the event e occurs in the configuration con :

$enable(con, e) = \{t \in T \mid sr(t) \subseteq con \wedge e \in event(t) \wedge guard(t)\}$

Where $sr(t)$ denotes the set of source restriction of transition t ; $event(t)$ denotes the $event$ set of t and $guard(t)$ denotes the $guard$ of t .

In the UML semantics documentation, the execution semantics of the state machine are expressed in terms of the operations of a hypothetical machine that implements a state machine specification. In the general case, the key components of this abstract machine are:

I) An event queue which accepts incoming event instances;

II) A dispatcher which selects and dequeues event instances for processing;

III) An event processor which processes dispatched event instances according to the general semantics of UML state machines and the specific form of the state machine in question.

Event processing by a state machine is partitioned into steps, each of which is caused by an event instance directed to the state machine. The fundamental semantics assumes that events are processed in sequence, where each event stimulates a run-to-completion (RTC) step. The next external event is dispatched to the state machine after the previous RTC step has completed. This assumption simplifies the transition function of the state machine since the incoming event is processed only after the state machine has reached a well-defined (stable) state configuration.

Once an event instance is dispatched, it may result in one or multiple transitions being enabled for firing. By default, if no transition is enabled, the event is discarded without any effect. In case where one or more transitions are enabled, the state machine selects a subset and fires them, moving the state machine from one active state configuration to a new active state configuration. This basic transformation is called a step. Actions that result from taking the transition may cause event instances to be generated for this and other objects.

In the RTC step, the generated events by the action in a transition are kept in the event queue $Event$, the completion events are kept in the event queue E_{com}, the events which are generated for other objects are kept in the event queue E_{env}.

In a given state, it is possible for several transitions to be enabled within a state machine. The issue then is which ones can be fired simultaneously without contradicting each other.

(14) $confl(t1,t2)$ is used to denote transition $t1$ and $t2$ are contradicting each other:

$$\forall t1, t2 \in S, \forall con \in Confi, confl(t1,t2) = true \Leftrightarrow$$
$$(sr(t1) \bigcap con) \bigcap (sr(t2) \bigcap con) \neq \varnothing .$$

Priorities can resolve transition conflicts, but not all of them, we use the state hierarchy to define priorities among conflicting transitions, a transition emanating from a substate has higher priority than a conflicting transition emanating from any of the containing states.

(15) $FireT$ is used to denote the set of maximal non-conflicting transition of configuration. If in the configuration con, the enable transitions which are triggered by event e can be defined as：

$$enable(con, e) = \{t \mid e \in e(t) \wedge \eta \models g(t)\}$$

Then the corresponding $FireT_{con}$ is defined as follows:

1) $\forall t1, t2 \in FireT_{con} : t1 = t2 \vee \neg confli(t1, t2)$;

2) $\forall t1 \in enable(con, e) : t1 \in FireT_{con} \vee$

$(\exists t \in FireT_{con} : t1 \neq t \wedge confli(t1, t))$.

(16) $D(FireT_{con}) = \{\bigcup D(t) \mid \forall t \in FireT_{con}\}$ is used to denote the clock variables which is reset in the $FireT_{con}$.

In summary, the real-time UML statechart over the set of clock variable C is defined as a tuple:

Definition 4. Real-time UML state chart, $RTSC$: $RTSC = (Con_{all}, Event, Action, Guard, con_{ini}, g_0, Inv, T)$, where:

- Con_{all} is a set consisting of all configuration;
- $Event$ denotes the set of all events of statechart;
- $Action$ is the set of action occurred in all transitions of statechart;
- $Guard = \{g \mid g \in CC(C)\}$ is the set of conditions which clock variable holds;
- $con_{ini} \in Con_{all}$ is initial configuration of statechart;
- $g_0 \in Guard$ is initial condition which clock variable holds;
- $Inv(con) : con \rightarrow CC(C)$ is time invariants of configuration con ;
- T is set of all transition in statechart and have: $T \subseteq Con_{all} \times Event \times Guard \times Action \times 2^{C} \times Con_{all}$.

III. THE OPERATIONAL SEMANTICS OF RTSC

Transition system is often used in computer science as models to describe the behavior of systems. The transition system we used is defined as follows[14]:

Definition 5. Transition System (*TS*). A transition system *TS* is a tuple: $TS = (S, Act, \rightarrow, I, AP, L)$ where:

- S is a set of states and Act is a set of actions;
- $\rightarrow \subseteq S \times Act \times S$ is a transition relation;
- $I \subseteq S$ is a set of initial states;
- AP is a set of atomic propositions;
- $L : S \rightarrow 2^{AP}$ is a labeling function.

The labeling function L relates a set $L(s) \in 2^{AP}$ of atomic propositions to any state s. $L(s)$ intuitively stands for exactly those atomic propositions $a \in AP$ which are satisfied by state s. Given that Φ is a propositional logic formula, then s satisfies the formula Φ if the evaluation induced by $L(s)$ makes the formula Φ true; that is: s |=Φ iff L(s) |=Φ.

This paper defines the operational semantics of *RTSC* as *TS*:

Definition 6. The operational semantics of *RTSC*. For $RTSC = (Con_{all}, Event, Action, Guard, con_{ini}, g_0, Inv, T)$ over the set of clock variables C, its operational semantics is defined as a $TS = (S, Act, \rightarrow, I, AP, L)$:

- $S = Con_{all} \times Event \times Eval(C)$;
- $Act = Action \bigcup \mathrm{R} \geq 0$;
- $I = (con_{ini}, 0, \eta_0)$, where 0 denotes no event in con_{ini} ;
- $AP = Con_{all} \bigcup ACC(C)$;
- $L((con, e, \eta)) = \{con\} \bigcup \{g \in ACC(C) \mid \eta \models g\}$;

The transition relation $\rightarrow$is defined by the following two rules:

I) discrete transition: $< con, e, \eta > \xrightarrow{\alpha} < con', e', \eta' >$ if the following conditions hold:

(a) there is a transition $con \xrightarrow{e \times g \times \alpha \times D} con'$ in *RTSC*;

(b) $\eta \models g$;

(c) $\eta' =$ reset D in η ;

(d) $\eta' \models Inv(con')$.

II)delay transition: $< con, \eta > \xrightarrow{d} < con, \eta + d >$ for $d \in R \geq 0$:

(e) if $\eta + d \models Inv(con)$.

The definition 6 entails that how to convert a statechart to a transition system, Algorithm 1 in TABLE I shows the pseudocode for this basic process.

In Algorithm 1, the next configuration con_{next} is consisted of all the next states which are generated by every transition caused by all states in the current configuration con_{cur} :

$$nextC(con_{cur}, FireT_{con}) = \bigcup_{i=1}^{n} nextS(s_i, t_i) \tag{2}$$

Where $s_i \in con_{cur}, t_i \in FireT_{con}$, and the computation rules for $nextS(s,t)$ are as follows:

$$\frac{s{=}(sn,sub(s),l,ts,en,ac,ex),t{\in}ts}{nextS(s,t){=}(sn,sub(s),l{=}lab(mt(t)),ts,en,ac,ex)} \tag{3}$$

Where $lab(s)$ denotes the label of s in its parent state.

$$\frac{s{=}(sn,sub(s),l,ts,en,ac,ex){\in}td(t){\wedge}s'{=}sub(s){\bigcap}td(t)}{nextS(s,t){=}(sn,sub(s),l{=}lab(s'),ts,en,ac,ex)} \tag{4}$$

not revise any of the current designations.

TABLE I. THE ALGORITHM OF CONVERT *RTSC* TO *TS*

```
Algorithm1 Convert RTSC to TS
Input: the middle expression form of UML statechart, RTSC and the set of clock constraints, $CC(C)$
output: the corresponding transition system, TS

Begin
construct the initial state $I = (con_{ini}, 0, \eta_0)$ by $con_{ini}$ and $\eta_0$ ;
$con_{cur} = con_{ini}, \eta_0 = \eta$ ;
determine $PE$ which is the set of all possible events of $con_{cur}$ ;
construct $CSS = \{cs_i = (con_{cur}, e_i, \eta) \mid \forall e_i \in PE\}$
which is the set of current state of TS ;
connect the initial state $I$ with all the current state $cs_i \in CSS$ ;
While( $PE \neq \varnothing$ ) do{
for $\forall cs_i \in CSS$ , determine $FireT_{con}$ which was triggered by $e_i \in cs_i$ ;
determine the next configuration of $con_{cur}$ ;
$con_{next} = nextC(con_{cur}, FireT_{con})$ ;
determine $\eta' = Eval(C)$ after the transition triggered by $e_i$ is completed;
determine $PE'$ which is the set of all possible events of $con_{next}$ ;
construct $NSS = \{ns_i = (con_{next}, e_i, \eta) \mid \forall e_i \in PE'\}$
which is a set of all next sates of TS;
connect $cs_i$ with all the corresponding
$\forall ns_i \in NSS$ ;
$CSS = NSS$ ;
$PE = PE'$ ;
}od
end
```

IV. THE METHOD OF MODEL CHECKING RTSC

The UML statechart with clock can be regarded as complicated timed automata which has hierarchical structure. *RTSC* is actually a timed automata form of flattening statechart. After converting *RTSC* to *TS*, the Timed Computation Tree Logic(TCTL)[14] is used to describe the time constraints. Then the *TS* and TCTL formula are used as the input of algorithm for model checking *SC*.

Definition 7. Syntax of Timed CTL. Formulae in TCTL are either state or path formulae. TCTL state formulae over the set AP of atomic propositions and set *C* of clocks are formed according to the following grammar:

$$\Phi ::= true \mid a \mid g \mid \Phi \wedge \Phi \mid \neg\Phi \mid \exists\varphi \mid \forall\varphi$$

where $a \in AP, g \in ACC(C)$ and φ is a path formula expressed by: $\varphi ::= \Phi \bigcup^{J} \Phi$,where $J \in IR_{\geq 0}$ is an interval whose bounds are natural numbers.

The TCTL model-checking problem is to check for a given timed automata TA and TCTL formula Φ whether TA $\models \Phi$. The main difficulty of the TCTL model-checking problem is that a transition system with uncountable states has to be analyzed, A naive graph analysis in the state graph of *TS*(TA) is therefore not feasible. Instead, the basic idea is to consider a finite quotient of this transition system, the so-called region transition system, which is obtained from the transition system of timed automata TA and the TCTL formula Φ [14]. In essence, the region transition system $RTS(\text{TA},\Phi)$ is the quotient of *TS*(TA) with respect to a bisimulation relation. The states in the region transition system are equivalence classes of states in *TS*(TA) that all satisfy the same atomic clock constraints, and from which " similar " time-divergent paths emanate, i.e., such states are TCTL equivalent. As the number of equivalence classes is finite, this provides a basis for TCTL model checking. In fact, rather than checking the TCTL formula Φ , it is checked whether a derived CTL formula holds in $\text{RTS}(\text{TA},\Phi)$.

In this paper, the *RTSC* is regarded as a complicated time automata and is used to take the place of TA above-mentioned. Algorithm 2 in TABLE II shows the basic recipe of TCTL model checking.

TABLE II. BASIC RECIPE OF TCTL MODEL CHECKING

```
Algorithm2 Basic recipe of TCTL model checking
Input: RTSC and TCTL formula $\Phi$
Output: TA $\models \Phi$

$\Phi'$ := eliminate the timing parameters from $\Phi$ ;
construct the transition system $TS(SC)$
determine the equivalence classes under $\cong$ ;
construct the region transition system $TS' = RTS(SC)$ ;
apply the CTL model-checking algorithm to check $TS' \models \Phi'$;
$SC \models \Phi$ if and only if $TS' \models \Phi'$.
```

In Algorithm 2, the $TS(RTSC)$ can be obtained from Algorithm 1. Φ' is a CTL formula that is obtained from the TCTL formula by eliminating the time parameters from Φ and $\cong$ denotes the equivalence used to obtain the quotient $RTS(SC,\Phi)$, the classic CTL model checking method can be used to determine whether $TS' \models \Phi'$ can be satisfied?, these approaches can be obtained from literature [14].

V. A CASE STUDY ON COVERTING RTSC TO TS

Considering a railroad crossing control system, see the schematic representation in Fig. 1[14]. For this railroad crossing a control system needs to be developed that closes the gate on receipt of a signal indicating that a train is approaching and only opens the gate once the train has signaled that it entirely crossed the road. The safety property that should be established by the control system is that the gates are always closed when the train is crossing the road.

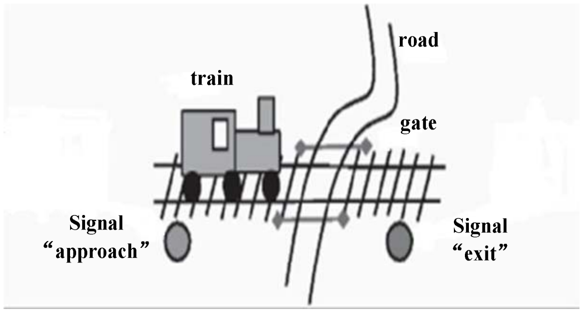

Figure 1. Railroad crossing (time abstract)

The RT-UML statechart of railroad crossing control system is illustrated in Fig. 2. The complete system consists of the three components: Train, Gate, and Controller. The Train is depicted in Fig. 2(left). Let us assume that the train signals its approaching of the gate at least two time units before it enters the railroad crossing. Besides, it is assumed that the train has sufficient speed such that it leaves the crossing five time units after approaching it, at the latest. On approaching the gate, clock y is set to zero, and only if y > 2 is the train allowed enter the crossing.

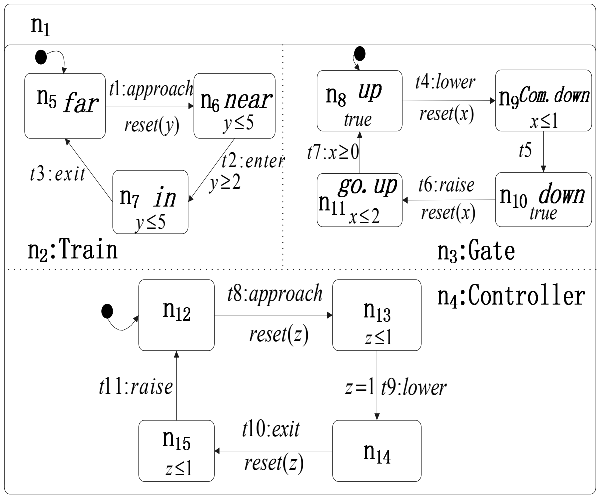

Figure 2. The RT-UML statechart of railroad crossing control system

The Gate is depicted in Fig. 2(right). Assuming that lowering the gate takes at most a single time unit, and raising the gate takes at least one and at most two time units. The state n_9 coming down with invariant $x \leq 1$ has been added to model that the maximal delay between the occurrence of action lower and the change to location down is at most a single time unit. Clock x is set to zero on the occurrence of action lower and thus "measures" the elapse of time since that occurrence. By restricting the residence time of coming down to $x \leq 1$, the switch to down must be made within one time unit. Note that this would not have been established by having a direct edge between state n_8 up and down with guard $x \leq 1$, as the value of x would not refer to the time of occurrence of lower. In a similar way, the purpose of state n_{11} goingup with invariant $x \leq 2$ is to model that raising the gate takes at most two time units. In the initial state n_8, no constraints are imposed on the residence time, i.e., inv(n_8)=true. The same applies to location down.

The Controller is depicted in Fig. 2(below) and is forced to send the signal "lower" exactly after one time unit after the Train has signaled its approaching.

Then the transition system of $RTSC$ is obtained by Definition 6 and Algorithm 1, see the corresponding $TS(RTSC)$ depicted in Fig. 3.

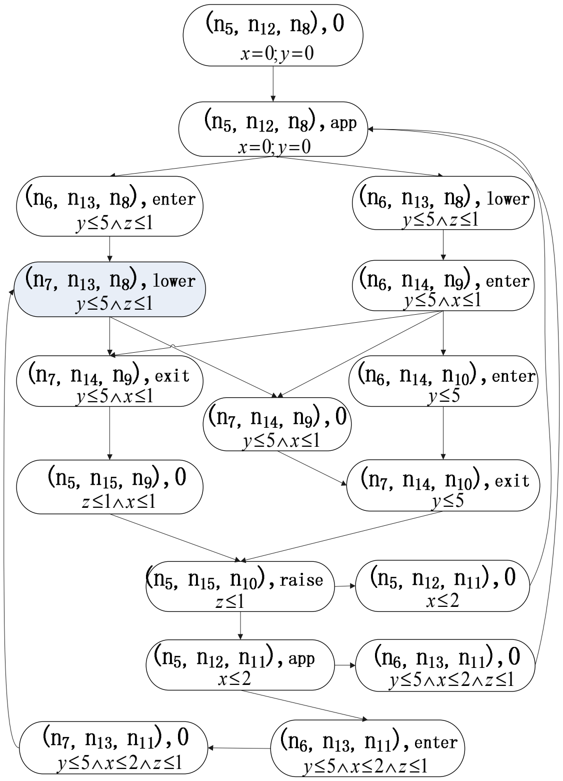

Figure 3. The TS of railroad crossing control system

There are eleven states and five events in RT-UML Statechart of railroad crossing control system. Firstly, we use *RTSC* to represent the statechart, for example, the parts of states and transition are as follows:

For $s_i \in S, i \in [1,11]$,

$s_1 = (n_1, (s_2, s_3, s_4), <>, <>, <>)$

$s_2 = (n_2, (s_5, s_6, s_7), 5, < t1, t2, t3 >, <>, <>, <>)$

For $j \in T, j \in [1,11]$,

$t1 = (t1, 5, (), approach, null, <>, 6, y)$.

Note that this transition system contains the state <(n7,n13,n8),lower, $y \le 5 \wedge z \le 1$>. In this state, the train is at the crossing while the gate is still open. However, this location turns out to be unreachable. It can only be reached when $y > 2$, but as y and z are reset at the same time, $y > 2$ implies $z > 2$, which is impossible due to the location invariant $z \le 1$.

VI. Conclusions

This paper introduces related concepts of clock to extend the definition of UML statechart. Through defining the operational semantics of *RTSC*, the conversion from *RTSC*,to transition system is achieved and a conversion algorithm is given. Based on the above theories, the general approach for model checking *RTSC*, is described, and in the end, a classic study case of conversion from *RTSC*, to TS is given.

In the process of defining the operation semantics of *RTSC* , the sole statechart of real time system is considered, this will make the representation of concurrent action of system become relatively complex. Even if there are many concurrent objects in the real time system, they are also regarded as many concurrent states of sole statechart. So in the future research, we will consider to separately model each individual object in the concurrent systems and define the composition operation of the operational semantics of multiple concurrent objects in the real-time concurrent systems.

Acknowledgment

This work is sponsored by the National Natural Science Foundation of China under grant number 60873038.

References

[1] OMG unified modeling language specification (Version 1.4). Needham: Object Management Group, Inc., 2001.

[2] Mikk E, Lakhnech Y, Petersohn C, Siegel M. On formal semantics of Statecharts as supported by STATEMATE. In: Duke D, Evans A, eds. Proceedings of the 2nd BCS-FACS Northern Formal Methods Workshop. London: Springer-Verlag, 1997.

[3] E.M. Clarke, O. Grumberg, D.A. Peled. Model Checking [M].Cambridge, MA:MIT Press.1999.

[4] G. Engels, J. Kuster, R. Heckel et al. A Methodolgy for Specifying and Analyzing Consistency of Object-Oriented Behavioral Models. In:Proc of the 8th European software engineering conference, ACM Press, 2001, 186-195.

[5] J.M. Küster, J. Stroop. Consistent Design of embedded Real-Time Systems with UML-RT. In: 4th International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'01) , Magdeburg, Germany, IEEE Computer Society Press, 2001, 31-40.

[6] B. Selic. Using UML for modeling complex real-time systems. Springer-Verlag, 1999, LNCS 1474, 250-262.

[7] T. Firley, M. Huhn, K. Diethers et al. Timed Sequence Diagrams and Tool-Based Analysis-A Case Study. Springer-Verlag, 1999, LNCS1723, 645-660.

[8] G. Behrmann, A. David, J. Hakansson et al. UPPAAL4.0. In: 3rd International Conference on the Quantitative Evaluation of Systems (QEST'06), University of California, USA, IEEE Computer Society Press, 2006, 125-126.

[9] A. Knapp, S. Merz, C. Rauh. Model Checking Timed UML State Machines and Collaborations. Springer-Verlag, 2002, LNCS 2469, 395-416.

[10] Li Xuandong, Zhao Jian-hua, Gong Jia-yu. Verifying Compositional Designs for Scenario-Based Timing Specifications. In: Proceedings of the 7th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC2004), Vienna, Austria, IEEE Computer Society Press, 2004, 253-256.

[11] Li Liu-ying, Wang Ji, Qi Zhi-chang. An Operational Semantics for UML Statechart Diagrams. Journal of Software, 2001, 12(12) : 1864-1873.

[12] Dong Wei, Wang Ji, Qi Zhi-Chang. An Approach of Model Checking UML Statecharts. Journal of Software, 2003, 14(4): 750-756.

[13] Zhou Ying, Zheng Guo-liang, Li Xuan-dong. An Operational Semantics for UML State Machines in Model Checking Context. ACTA ELECTRONICA SINICA, 2003, 12A : 2091-2095.

[14] Christel Baier, Joost-Pieter Katoen, Kim Guldstrand Larsen. Principles of Model Checking. [M]. The MIT Press. May 31, 2008.