ICIC Express Letters ICIC International ©2016 ISSN 1881-803X
Volume **10**, Number **11**, November **2016** pp. 2639–2644

FORMAL MODELING OF RAILWAY SIGNAL SAFETY-CRITICAL SOFTWARE

Yao Li, Jin Guo, Shan Yan and Qian Yu

School of Information Science and Technology
Southwest Jiaotong University
No. 111, North Section 1, Second Ring Road, Chengdu 610031, P. R. China
ly_9967@163.com

Received March 2016; accepted June 2016

Abstract. *Focusing on the problem that the traditional modeling methods fail to satisfy the clock constraint requirement of railway signal safety-critical software well, a formal method of Timed SyncCharts is proposed which extends SyncCharts with clock constraints. First, the formal description of Timed SyncCharts is provided systematically with Z notion; then, the equivalent transformation of Timed SyncCharts into Timed Automata model is demonstrated, which makes it easy to analyze and verify the function and time properties; finally, in order to show the feasibility and effectiveness of the proposed method, the Timed SyncCharts model of switch of computer based interlocking is established, and the corresponding Timed Automata model is presented.*
Keywords: Safety-critical software, Formal modeling, Clock constraint, *Timed SyncCharts*, *Timed Automata*

1. Introduction.

With the rapid development of computer, software is increasingly applied to the control equipments of railway signal system. As a typical safety-critical software, railway signal software performs safety-related function and requires a strict fail-safe and clock constraint requirements [1]. Due to the effects of concurrency, conflict, competition, and realtime of railway signal system, how to describe the function and clock constraint effectively is a key to the safety assurance of this software.

Visual language of formal methods is easy to understand and safety analysis is widely used in safety-critical software design. The main visual language includes *Timed Automata* [2], *Timed Petri Net* [3,4], *Timed UML* [5], etc. Due to the flat structure, these methods easily make model complex and have a huge state space; meanwhile, preemption is not involved. Hsiung et al. presented a method of *SafeCharts* [6,7], which supports hierarchy and includes functional layer and safety layer. *SafeCharts* is appropriate for safety analysis, but it lacks clock constraints, and the functional and safety layers increase model complexity. Harel presented a *Statecharts* [8] method, which supports hierarchy and concurrency. Charles extended *Statecharts* to a *SyncCharts* [9,10] method, which strictly restricts the transition behavior and improves preemption mechanism. As concurrency, hierarchy, synchronization and communication are supported, and *SyncCharts* improves modeling efficiency of complex system. However, temporal behavior as a typical characteristic of safety-critical software is still not involved in *SyncCharts*, which results in *SyncCharts* has to face the problem that it cannot specify the required temporal behavior as *Timed Automata* does. This paper proposes a *Timed SyncCharts* method, which extends *SyncCharts* with real-time constructs, including clocks and clock constraints. The advantages of modeling complex behavior with *Timed SyncCharts* are combined with the advantages of specifying temporal behavior with *Timed Automata*, resulting in the extension of *SyncCharts* to efficiently specify time-critical systems like railway signal software.

The organization of the paper is as follows. The next section introduces *Timed SyncCharts*, and its formal definition is given by Z notion [11]. A model analysis method for *Timed SyncCharts* that translates *Timed SyncCharts* to *Timed Automata* is presented in Section 3. Section 4 presents a case study, and Section 5 concludes the article.

2. ***Timed SyncCharts*** Language.

Statecharts is a tuple (B, S, h, t, Δ, A) [12], where:
B is a finite set of basic events;
S is a finite set of state names;
$h \in S \to 2^S$ is the hierarchy function;
$t \in S \to \{PRIM, AND, OR\}$ is a type function;
$\Delta \in S \to 2^S$ is an initial state function;
$A := \langle Source, Dest, Trigger, Action\rangle$ is a finite set of transitions.

From the definition, *Statecharts* lacks clock constraint, and the hierarchy is defined roughly. *SyncCharts* is a variant of *Statecharts* and inherits this disadvantage. To define a complete *Timed SyncCharts*, this section gives definitions of clock variables and clock constraints first.

Definition 2.1. *A time sequence* $t_c = t_{c1}t_{c2}t_{c3}\cdots t_{ci}\cdots$ *is an infinite sequence of time values* $t_{ci} \in R^+$*, and:*
1). $\forall i \geq 1$, $t_{ci} < t_{ci+1}$;
2). $\forall t \in R^+$, $\exists i \geq 1$, $t_{ci} > t$.

Definition 2.2. *For a clock variables set* C*, the set* $\Delta(C)$ *of clock constraints* δ *is:*

$$\delta := x \propto d | x - y \propto d | \neg\delta_1 | \delta_1 \wedge \delta_2,$$

where δ_1, δ_2 *are clock constraints, and* $x, y \in C$, $d \in N$, $\propto \in \{\leq, <, =, \geq, >\}$.

A *clock valuation* over C is a mapping $v : C \to t_c$, which assigns to each clock a time value. Clock valuation v satisfies a clock constraint δ of C if δ valuates to true using v, denoted by $v(C) \models \delta$.

2.1. ***TSTG.***

After defining clock variables and clock constraints, this section defines the basic unit of hierarchy of *Timed SyncCharts*, *TSTG* (Timed States Transition Graph), using schema language of Z notion.

TSTG is a state transition graph with clock constraints, which contains elements of finite state set, initial state, finite final state set, finite signal set, clock variables set, transition set, state action function, and state invariant function. The Z state schema of *TRANSITION*, *LABEL* and *TSTG* are shown in Figure 1.

TSTG has only one initial state, and its transition contains elements of source state, priority, transition type, label and target state. Given a transition t, the source state and

TRANSITION
source : S
priority : N^+
type : TYPE
label : LABLE
target : S

LABLE
$\delta : \Delta(C)$
$g : G$
$c : \mathbb{F}\, C$

TSTG
init : S
final : $\mathbb{F}\, S$
signal : $\mathbb{F}\, G$
clock : $\mathbb{F}\, C$
T : $\mathbb{F}_1$ *TRANSITION*
ma : $S \twoheadrightarrow Type \times G$
mc : $S \twoheadrightarrow \Delta(C)$

$init \in S \wedge final \subseteq S$
$\forall t : T \bullet target(t) \neq init$
$\forall t_1, t_2 \in T \bullet$ **if** $source(t_1) = source(t_2)$
then $t_1.Priority \neq t_2.Priority$
$T_{STRONG}.Priority > T_{WEAK}.Priority > T_{SYNC}.Priority$

Figure 1. Z schema of *TSTG*

target state is denoted by $source\,(t)$ and $target\,(t)$. Priority eliminates the indeterminacy when more than one transitions of a state are triggered. Transition type is denoted by $TYPE := \{T_{STRONG}, T_{WEAK}, T_{SYNC}\}$, where T_{STRONG} is strong transition, T_{WEAK} is weak transition, and T_{SYNC} is normal termination transition. When a state has no less than one outgoing transition, each has a different priority, and T_{STRONG} has a higher priority than T_{WEAK}, and T_{WEAK} is higher than T_{SYNC}. $LABEL$ is label of transition, where δ is the clock constraint, g is the input signal, and c is the clock that the transition resets. ma is a state action function, which assigns signals to each state, where $Type := \{ENTER, IN\}$ represents the signal that would be triggered when entering or being in a state. mc is a state invariant function, which assigns each state clock constraints that the state must be satisfied.

2.2. ***Timed SyncCharts.*** After defining the *TSTG*, a complete *Timed SyncCharts*, denoted by *TSC*, can be defined.

TSC is an extended structure of nesting *TSTG* in states of *TSTG*, and then states are extended to be simple or macro, denoted by $type : S \to \{\text{BASIC, MACRO}\}$. *TSC* contains elements of finite states set, root state, clock variables set, finite signals set, finite *TSTG* set, and *TSTG* assignment function, denoted by $(S, top, C, G, Tstg, child)$. If $\exists tstg : TSTG \in child\,(s_2)$, then $\forall s_1 \in tstg.S$, s_1 is a child of s_2, denoted by $s_1 \in child\,(s_2)$; s_2 is parent of s_1, denoted by $s_2 = parent\,(s_1)$. The reflexive transitive closure $child^*(s) ::= child(s) \cup \bigcup\{st : child^*(s) \bullet child^*\,(st)\}$ denotes the child relation of s. The Z state schema of *TSC* is shown as Figure 2, and it satisfies the following constraints:

1). state can only be *BASIC* or *MACRO*;
2). transition must be in the same layer;
3). state possesses only one parent, and *top* has no parent;
4). every state is a child of *top*;
5). the *child* function cannot support cyclic assignment;
6). every *TSTG* of source state of transition T_{SYNC} contains $final$ states;
7). each *MACRO* state has no more than one T_{SYNC} transition.

TSC

$top : S$

$Tstg : \mathbb{F}\ TSTG$

$child : S \twoheadrightarrow \mathbb{F}\ Tstg$

$\forall tstg \in Tstg \bullet tstg.S \subseteq S$

$top \in S \wedge type(top) = \text{MACRO}$

$\forall s \in S \bullet type(s) = \text{BASIC} \vee 'type(s) = \text{MACRO}$

$\forall s \in S \bullet \textbf{if } type(s) = \text{BASIC} \textbf{ then } child(s) = \varnothing$

$\quad \textbf{else } child(s) \neq \varnothing$

$\forall t : T \bullet parent(source(t)) = parent(target(t))$

$\forall s, s_1, s_2 : S \bullet \textbf{if } s_1 = parent(s), s_2 = parent(s), \textbf{then } s_1 = s_2$

$top = S \setminus child^*(top)$

$\forall s : S, s \notin child^*(s)$

$\forall t : T \bullet \textbf{if } type(t) = T_{SYNC} \textbf{ then}$

$\quad type(source(t)) = \text{MACRO} \ \wedge$

$\quad \forall tstg : TSTG \in child(source(t)) \bullet tstg.final \neq \varnothing \wedge$

$\quad \forall t_1 : T \bullet \textbf{if } t_1 \neq t, source(t_1) = source(t) \textbf{ then}$

$\quad t_1.TYPE \neq T_{SYNC}$

FIGURE 2. Z schema of *Timed SyncCharts*

3. Model Analysis of *Timed SyncCharts*.

Given a *TSC*, the model analysis is to check whether *TSC* satisfies a safety property ϕ, abbreviated as $TSC \models \phi$.

Timed Automata (*TA*) is an extended automaton to model the behavior of real-time system over time, and has mature model analysis algorithms.

Definition 3.1. *A timed automata TA is a tuple* (L, l_0, C, A, E, inv) *with:*
L *is a finite set of locations;*
$l_0 \in L$ *is an initial location;*
C *is a finite set of clocks;*
A *is a finite alphabet;*
$E \subseteq L \times \Delta(C) \times A \times 2^C \times L$ *is a finite set of edges;*
$Inv : L \to \Delta(C)$ *assigns an invariant to locations.*

TSC model analysis could be performed through *TA* analysis. First, *TSC* should be translated to *TA* structure equivalently, and then *TA* model tools could be used, for example UPPAAL, to analyze *TSC* model indirectly. Thus, model analysis of *TSC* amounts to model analysis of its corresponding *TA*:

$$TSC \models \phi \text{ if and only if } TA \models \phi.$$

A configuration of *TSC* is a maximal set of states that the system could be in simultaneously, and every state should satisfy its invariant. *top* belongs to any configuration, and if a configuration contains a *MACRO* state, every *TSTG* which has exact one state belonged to this configuration. *Z* schema of configuration is shown as Figure 3.

$$\begin{array}{|l}
\underline{config\_ : TSC} \times v(C) \to \mathbb{F}_1 S \\
\quad \forall tsc : TSC; conf : \mathbb{F}_1 S \bullet \\
\quad conf = config(TSC, v(C)) \Leftrightarrow top \in conf \wedge \\
\qquad \forall s : S \in conf \bullet v(C) \models s.\delta \wedge \\
\qquad \textbf{if}\ type(s) = \text{MACRO}\ \textbf{then}\ \forall tstg : TSTG \in child(s), \exists_1 s_1 \in tstg.S \bullet s_1 \in conf
\end{array}$$

FIGURE 3. Configuration of *TSC*

Paths of *TA* are sequences of state transitions in *TA*. The reachable analysis of *TA* paths can be used to verify whether *TA* satisfies its system properties.

Definition 3.2. *A path of a TA is a finite or infinite state sequence* $s_0, s_1, s_2, s_3, \cdots$, $s_0 = l_0$ *and* $\forall i > 0$, $\exists \delta \in \Delta(C)$, $a \in A$, $c \in C$, $(s_i, \delta, a, c, s_{i+1}) \in E$.

A *TSC* path is a configuration sequence of an execution process of *TSC*. $next(cf, \delta, g, c)$ means a reachable configuration from cf with clock valuation δ, signal g and reset clock set c. The initial configuration is denoted by $conf_0(TSC, 0)$.

Definition 3.3. *A path of TSC is a finite or infinite configuration sequence* $cf_0, cf_1, cf_2, \cdots$, $cf_0 = conf_0(TSC, 0)$ *and* $\forall i \geq 0$, $\exists \delta \in \Delta(C)$, $g \in G$, $c \in C$, $cf_{i+1} \in next(cf_i, \delta, g, c)$.

Theorem 3.1. *Let* $\kappa(TSC)$ *be a mapping of* $TSC = (S, top, C, G, Tstg, child)$ *to a tuple* $(L', l'_0, C', A', E', Inv')$, *with:*
$L' = conf(TSC, v(C))$;
$l'_0 = conf_0(TSC, 0)$;
$C' = C$;
$A' = G$;
$E' := \{(s, \delta, a, c, t) | s, t \in L', \delta \in \triangle(C), a \in G, c \in C, t = next(s, \delta, a, c)\}$;
$Inv' : L' \to \Delta'(C')$;
then $\tau = (L', l'_0, C', A', E', Inv')$ *is a TA.*

Proof: According to the definition, the locations set, initial location, clocks set, actions set and location invariant function of τ are mapped to configurations set, initial configuration, clocks set, signals set and state invariant function of TSC respectively. $\forall cf_1, cf_2 \in conf\,(TSC, v\,(C)) \bullet \exists \delta \in \Delta\,(C)$, $g \in G$, $c \in C$, $cf_2 = next\,(cf_1, \delta, g, c)$, by the mapping κ, $\exists s, t \in L'$, $\delta' \in \Delta'\,(C')$, $a' \in A'$, $c' \subseteq C'$, $s = cf_1$, $t = cf_2$, $\delta' = \delta$, $c' = c$, $a' = g$, $(s, \delta', a', c', t) \in E'$, then $\tau = (L', l'_0, C', A', E', Inv')$ is a TA. □

$\kappa\,(TSC)$ implies that the configuration set TSC and state set of TA have equivalence state space and execution path.

Theorem 3.2. *Let ρ be a projection of a TSC path $cf_0, cf_1, cf_2, \cdots$ to a τ sequence $s_0, s_1, s_2, \cdots$, then $\forall \omega \in paths\,(\kappa\,(TSC)) \Leftrightarrow \exists \sigma \in paths\,(TSC) : \rho\,(\sigma) = \omega$.*

Proof: $\Rightarrow$: $\kappa\,(TSC)$ maps a TSC to a TA, and TSC configurations set is mapped to TA states set. $\forall s_1, s_2 \in L'$, $\delta' \in \Delta'\,(C')$, $a' \in A'$, $c' \subseteq C'$, $(s_1, \delta', a', c', s_2) \in E'$, according to Theorem 3.1, in TSC, $\exists cf_1, cf_2 \in conf\,(TSC, v\,(C))$, $g \in G$, $\delta \in \triangle\,(C)$, $c \in C \bullet cf_1 = s_1$, $cf_2 = s_2$, $cf_2 = next\,(cf_1, \delta, g, c)$. Therefore, for any TA sequence $\omega = s_0, s_1, s_2, \cdots$, in TSC, there exists a path $\sigma = cf_0, cf_1, cf_2, \cdots$, and ρ projects σ onto ω.

$\Leftarrow$: By *contradiction*, assume that $\exists \sigma = cf_0, cf_1, \cdots, cf_i, cf_{i+1}, \cdots$, $\sigma \in paths(TSC)$, $\rho(\sigma) = \omega$, $\omega \notin paths(\kappa(TSC))$. According to Theorem 3.1, $\exists i, cf_{i+1} \notin next(cf_i)$; if not, $\omega \in paths\ (\kappa(TSC))$. However, if there exists such a (cf_i, cf_{i+1}), then $\sigma \notin paths(TSC)$. □

Theorem 3.2 implies that TSC and TA are commutative, and run on the same sates sequence and transition relation, as shown in Figure 4.

4. Example.

Computer based interlocking system [13] is a core equipment of urban rail transit signal system, which comprises switch, signal and route components. Taking switch as an example, when it receives switch request, if safety conditions, such as unlocked

$$\begin{tikzcd}
TSC \arrow[r, "\kappa"] \arrow[d, "paths"] & TA \arrow[d, "paths"] \\
paths(TSC) \arrow[r, "\rho"] & paths(TA)
\end{tikzcd}$$

FIGURE 4. Mapping of TSC and TA

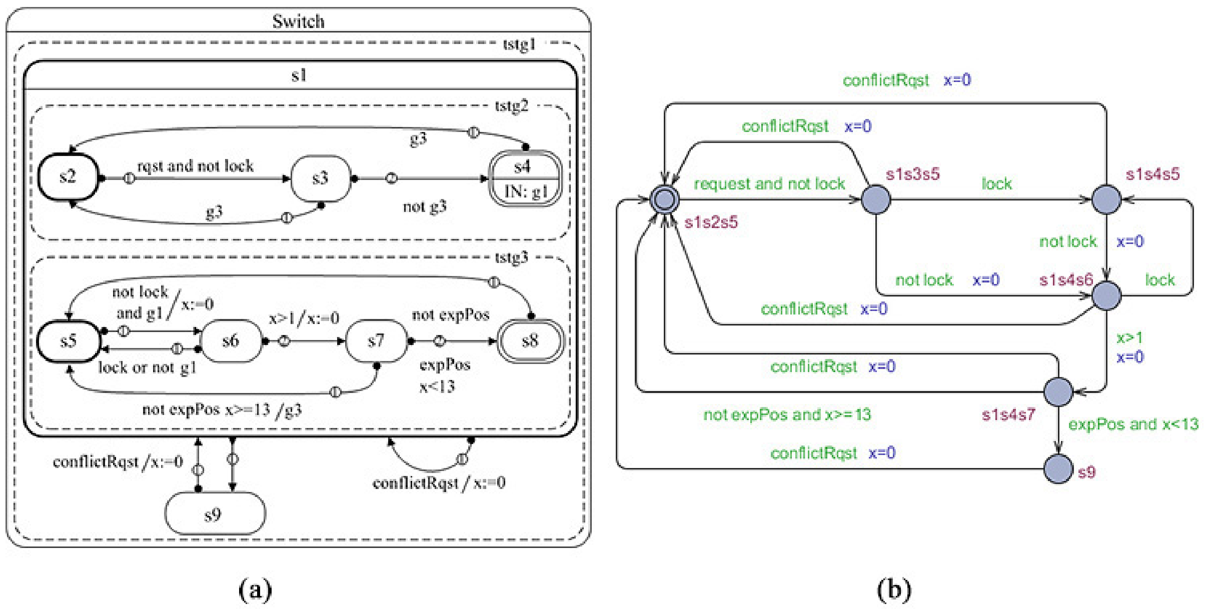

FIGURE 5. TSC and TA models of switch

state, no conflict request signal, keep over 1 second, and the expected position is switched successfully within 13 seconds, switch will enter a request success state.

The switch *TSC* model is shown as Figure 5(a). The hierarchy and priority of *TSC* allows us to model the request and safety conditions independently. The model will reset whenever a conflict request signal is received. It is not necessary to draw all possible combinations explicitly, and this enables engineers to create models of complex system compactly. The switch *TSC* model contains two signals $g1$ and $g3$, which signified respectively that the request is in safety conditions and that switch has a breakdown. A clock x is adopted to specify the required temporal behavior. The only 1 subcomponent and 13 transitions instead of 3 subcomponents and 22 transitions *TA* needed with the same requirements show that *TSC* has a better expressiveness for complex system. The corresponding *TA* model of the *TSC* model is shown as Figure 5(b), which omits the unnecessary states, and can be easily analyzed with UPPAAL.

5. Conclusions.

A critical problem of safety assurance of railway signal software is how to describe the functional and time requirements exactly and clearly. This paper proposed a *TSC* method which extends the *SyncCharts* and supports the functional and clock constraint modeling requirements of railway signal software. Model analysis of *TSC* is performed indirectly through equivalent *TA* model analysis. This method has been applied to designing computer based interlocking software, and the practice shows that the method has a guiding significance for the design and development of railway signal software.

Probabilistic characteristic is another basic aspect of safety critical software, which describes the randomness the system is exposed to, or the randomness the system itself exhibits. In future work, probability element is planned to be incorporated into the formalism to support for modeling of timed and probabilistic aspects.

Acknowledgment. This work is partially supported by the China Railway Corporation under Grants 2015X009-D and 2014X008-A. The authors also gratefully acknowledge the helpful comments and suggestions of the reviewers, which have improved the presentation.

REFERENCES

[1] E. J. Joung, S. C. Oh, S. H. Park and G. D. Kim, Safety criteria and development methodology for the safety-critical railway software, *Proc. of the Telecommunications Energy Conference*, Incheon, Korea, 2009.

[2] J. Bengtsson and Y. Wang, *Timed Automata: Semantics, Algorithms and Tools*, Springer Berlin Heidelberg, 2004.

[3] J. Wang, *Timed Petri Nets: Theory and Application*, Springer Science & Business Media, 2012.

[4] W. Ma, K. Zhao, X. Hei, G. Xie and J. Ma, Research on time petri net oriented UML statechart and its application, *ICIC Express Letters*, vol.9, no.3, pp.929-935, 2015.

[5] A. David, M. O. Moller and Y. Wang, *Formal Verification of UML Statecharts with Real-Time Extensions*, Springer Berlin Heidelberg, 2002.

[6] P. A. Hsiung, Y. R. Chen and Y. H. Lin, Model checking safety-critical systems using safecharts, *IEEE Trans. Computers*, vol.56, no.5, pp.692-705, 2007.

[7] H. Dammag and N. Nissanke, Safecharts for specifying and designing safety critical systems, *Proc. of the 18th IEEE Symposium on Reliable Distributed Systems*, Lausanne, Switzerland, 1999.

[8] D. Harel. Statecharts: A visual formalism for complex systems, *Science of Computer Programming*, vol.8, no.3, pp.231-274, 1987.

[9] C. André, *Semantics of SyncCharts*, University of Nice-Sophia Antipolis, 2003.

[10] A. Charles, Representation and analysis of reactive behaviors: A synchronous approach, *Computational Engineering in Systems Applications*, 1996.

[11] Z Standards Panel, *Formal Specification – Z Notation – Syntax, Type and Semantics*, International Standard, 2002.

[12] J. Philipps and T. Yoneda. Symbolic verification of statecharts, *Technical Report of IEICE*, vol.95, no.212, pp.49-56, 1995.

[13] X. Hei, L. Chang, W. Ma and G. Xie, Using UML and petri nets for the design and verification of a railway interlocking system, *ICIC Express Letters*, vol.5, no.8(B), pp.2825-2830, 2011.