Real-Time Systems Modeling and Verification with Aspect-Oriented Timed Statecharts

Xinxiu Wen and Huiqun Yu

College of Information Science and Engineering,
East China University of Science and Technology,
Shanghai 200237, China
{wenxinxiu, yhq}@ecust.edu.cn

Abstract

The modeling and verification of real-time systems is a challenging task in the area of software engineering. This paper proposes a formal method for modeling and verification of real-time systems based on aspect-oriented timed statecharts and linear-time temporal logic. Behaviors of real-time systems are modeled by aspect-oriented timed statecharts, while key properties of systems are specified by linear-time temporal logic. Moreover, aspect-oriented timed statecharts are translated to timed automata with guards to simulate the executable paths of systems and model checking technologies are applied to the verification of models. An elevator example illustrates our modeling and verification method.

Keywords: *Real-time systems; Aspect-orientation; Timed statecharts; Weaving.*

1. Introduction

Real-time systems are complex reactive systems, errors in such systems may lead to fatal results. Therefore, adopting modern technology to guarantee their correctness and improve their qualities at the early phase of software development has become an urgent task.

Aspect-oriented software modeling (AOSM) has attracted much attention in recent years due to separating core concerns from crosscutting concerns successfully and improving software modularity effectively [1]. Moreover, as an important technology that ensures software correctness, formalization verification approach has been widely used in real-time systems [2, 3]. Consequently, modeling real-time systems with AOSM technology and verifying the model with model checking is an effective way to improve the reliability and performance of real-time systems.

In this paper, To separate systems concerns at the beginning phase of software development, real-time systems requirements are divided into base functional requirements, including functional requirements, extended functional requirements, inheriting functional requirements and non-functional properties. Base functional requirements are modeled as base use cases models and are formalized as base timed statecharts(BTSCs), while the others are modeled as aspect use cases models and are formalized as aspect timed statecharts (ATSCs). Next, BTSCs and ATSCs are woven into aspect-oriented timed statecharts (AOTSCs). In the end, AOTSCs are translated to timed automata with guards and systems key properties are described with linear-time temporal logic for model checking. To illustrate our approach vividly, we go through with an elevator example.

The remainder of this paper is organized as follows. Section 2 gives a brief introduction to timed automata with guards and linear-time temporal logic. Section 3 describes our modeling method. Section 4 validates the woven model. Section 5 is the conclusion.

2. Background

A timed automaton is to model the behaviors of real-time systems over time, while a timed automaton with guards is an automaton which is adjoined to a set of continuous variables whose dynamical evolution is time-driven [4].

2.1 Timed Automata with Guards

Definition 1 A Timed Automaton with Guards (*TAG*) is a 6-tuple $(X, E_T, C_T, T_T, I_T, x_0)$, where: X is the finite set of states; E_T is the finite set of events; C_T is the finite set of clocks; T_T is the set of timed transitions of the automaton; $I_T : X \rightarrow \mathcal{C}(C_T)$ is the set of state invariants; $x_0 \in X$ is the initial state.

A transition from state x_{in} to x_{out} in *TAG* is described as $(x_{in}, guard, e, reset, x_{out}) \in T_T$, where: guard $\in \mathcal{C}(C_T)$ is an admissible constraint for the clock in set C_T, $e \in E_T$, and $reset \subseteq C_T$ gives the clocks to be reset with this transition.

When considering the dynamic evolution of a *TAG*, a complete system state at time t is composed of the discrete state $x \in X$ at time t and the value of clock c at time t, $t \in R^+$. Starting from the initial state$(x_0, 0)$, there are two kinds of transitions: event transition and delay transition. All executable paths of *TAG* could always be extended in time by a delay transition or an event transition: $(x_0, 0), (x_1, c_1), (x_2, c_2), \ldots, (x_i, c_i)\ldots$

2.2 Linear-time Temporal Logic

Real-time systems key properties are described as Linear-time Temporal Logic (LTL) [5]. A temporal formula in LTL is constructed out of state formulas to which we apply the boolean connectives $\neg$, $\vee$, $\wedge$ and $\rightarrow$, and the temporal operators: *X(next), F(future), G(globally), U(until)*, etc.

Definition 2 Linear-time temporal logic has the following syntax given in Backus Naur Form (BNF), where p is any propositional atom, ψ and φ are propositional formulas:

$$\psi ::= p \mid (\neg \psi) \mid (\psi \wedge \varphi) \mid (\psi \vee \varphi) \mid (\psi \rightarrow \varphi) \mid X\,\psi \mid F\,\psi \mid G\,\psi \mid \psi\, U\, \varphi \mid \quad (1)$$

A transition system M=<S, R, L> is a model, where S is a set of states, R is the transition relation, L is the labeling function. A path in a model M is an infinite sequence of states $\pi = s_0, s_1 \ldots$ in such that, for each $i \geq 1$, s_i R s_{i+1}. Whether π satisfies a LTL formula or not is defined by the satisfaction relation |=. For a model M, s∈S, and ψ is a LTL formula. If for every execution path π of M starting at s, we have π|= ψ, then we write M, s|= ψ.

When verifying a real-time system which is modeled as a TAG, system key properties are formalized as LTL formula ψ. If there exists a path doesn't satisfy the formula ψ, then checking would be stopped and a counter example would be given; if all paths satisfy formula ψ, it can be concluded that the TAG satisfies the system key properties.

3. Aspect-oriented Modeling with Timed Statecharts

The separation of core requirements and crosscutting requirements at the beginning phase of software development is beneficial to software modularity. Since use cases explore requirements from the user's perspective, they are natural technical candidates to model and organize user concerns during requirements analysis phase. Aspect-oriented requirements analysis with use cases is thus an effective approach to distinguish functional requirements from crosscutting requirements [6].

3.1 Aspect-oriented Use Case Modeling

To separate system corncerns effectively, we specify that base functional requirements of real-time systems are modeled as base use cases, while including functional requirements, extended functional requirements, inheriting functional requirements and non-functional properties are modeled as aspect use cases. To describe our method vividly, we go through with an elevator example.

Elevator control system is a typical case of real-time systems [7]. This paper analyzes it by ignoring some technical details (Here we just consider one elevator and one controller). The elevator should be unlocked before putting into action and there is a control system which is responsible for moving up or moving down, opening or closing the doors. The doors of elevator would be opened automatically in 1 second when arriving at the appointed floor if nobody opens the doors and should be closed automatically in 8 seconds if nobody closes the doors. The elevator could repeatedly response the signals outside or inside the car based on the principle that the elevator will move in the current direction until all passengers who are requesting rides in this direction are picked up and delivered. In the end, if there is no request, it would remain at current floor with its doors closed.

From requirements descriptions of elevator control system we know that, moving up, moving down, opening the doors and closing the doors are base functions of an elevator, they should be modeled as base use cases; timeout handling is non-functional property, passengers' requests scheduling and passengers weighing are including functions, these requirements should be modeled as aspect use cases.

3.2 Aspect-oriented Timed Statecharts Modeling

Aspect-oriented use case model describes the system from an external perspective and its informality is a barrier to the application of automated analysis methods such as simulation and validation, the system should be formalized further during the design phase.

Statecharts were first introduced by David Harel to describe the complex behavior of reactive systems in 1987 [8], while UML statecharts are object-oriented variations of Harel's statecharts with properties as orthogonality, refinement, etc.

Definition 3 A base timed statechart *(BTSC)* is a 7-tuple $(S, E, C, T, r, \rho, type)$, where: S is a finite set of states; E is a finite set of events; C is a finite set of clocks; $T=T_1 \cup T_2$ is a finite set of transitions, T_1 represents immediate event transitions that are triggered by inputs, while T_2 represents timed transitions that depend on time rather than inputs; $r \in S$ is root; $\rho(s)$ specifies the direct offspring of state s; $type(s) \in \{BASIC, AND, OR\}$ means type of state s.

Aspect timed statecharts extend base timed statecharts with pointcut and advices for modeling crosscutting concerns of real-time systems.

Definition 4 An aspect timed statechart *(ATSC)* is a 3-tuple (A, P, tp), where: A is a set of advices, each advice is a base timed statechart; P means pointcut, it is a finite set of join points (join points are certain transitions of BTSC); $tp \in \{before, after, around\}$ is the type of advices.

Base use case model should be formalized as base timed statechart, while aspect use case model should be formalized as aspect timed statechart, the concrete steps of tranlation could be found in [7]. Fig. 1 shows the base timed statechart of elevator, while Fig.2 shows the three aspect timed statecharts of elevator. According to aspect-oriented use case model of elevator, there are three aspect use cases which should be formalized as three aspect timed statecharts separately. Handling timeout replaces the execution of p_open/p_close event of BTSC under certain conditions and should be modeled as an ATSC with around type. From

Fig.2.a we know that the crosscutting position of aspect TIMEOUT should be the join point transition t_8/t_9 of the BTSC, which means when the clock exceed timing constraint (1 second for open p_event or 8 seconds for p_close event), transition t_{ai} will substitute for the base join point transition t_8/t_9. Besides, the including functions of real-time systems should be modeled as ATSCs too. Fig.2.b illustrates the PLAN aspect and from the graph we know that transition t_{ai} (t_{ai} is immediate) would be triggered after base join points transition t_5/t_6. Fig.2.c illustrates the CHECKING aspect and transition t_{ai} (t_{ai} is immediate) would be triggered before base join points transition t_9.

The introduction of ATSC realizes the modularization and formalization of crosscutting concerns at the design phase, the next work is to weave BTSC with ATSC into aspect-oriented timed statechart for model checking.

Definition 5 An aspect-oriented timed statechart *(AOTSC)* is a pair *(BTSC, ATSCi, 1≤i≤n)*, where: *BTSC* represents a base timed statechart; $ATSC_i$ represents an aspect timed statechart, n is the number of aspect timed statecharts.

This paper adopts weaving mechanism like AspectJ, which join points are well-defined points in the execution of BTSC [9]:

(1) Initially, AOTSC=BTSC;

(2) For each aspect timed statechart $ATSC_i$, the weaving process is as follows according to the type of advices: If the type of advice is before or after, then substitute advice of $ATSC_i$ for transitions at join points of BTSC. If the type of advice is around, then interrupt the transitions at join points of BTSC and execute the advice of $ATSC_i$;

(3) If there are several advices which apply to the same join point, it is necessary to plan their weaving sequences. If the advices are in the same $ATSC_i$, then their positions decide their precedence; if they are in different ATSCs, then declaring precedence specifies their weaving sequences with BTSC.

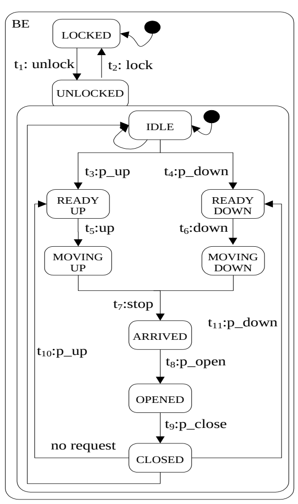

Figure 1 The BTSC of Elevator

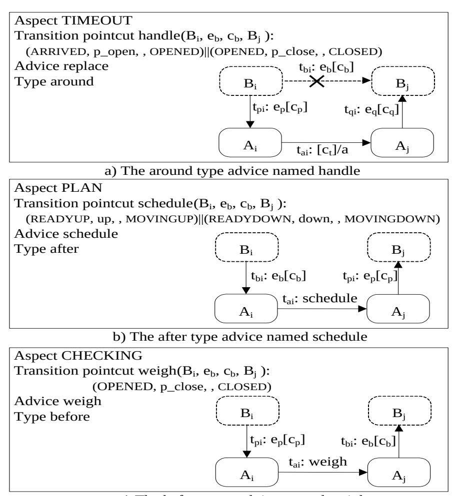

Figure 2 The ATSCs of Elevator

4. Model Checking Timed Statecharts

Model checking is a formal verification technique which checks whether a system satisfies certain properties or not. Given a system model M and a temporal logic formula ϕ, if M

satisfies formula ϕ then *M* is called the model of ϕ and is expressed as $M, s|=\phi$. In this paper, the operational semantics of *AOTSC* is explained by extended hierarchical automaton (*EHA*) [10], which could be described as a transition system $M=(S, s_0, \xrightarrow{STEP})$. $S=Conf(\rho)\times(\Theta E)$ is the set of statuses of *M* ($Conf(\rho)$ denotes the set of all configurations of *EHA*, ΘE denotes the set of all structures of a certain kind (like FIFO queues) over *E*); $s_0=(C_0, \varepsilon_0)\in S$ is the initial status of *M*; $\xrightarrow{STEP}\subseteq S\times S$ is the transition relation of *M*. The *AOTSC* can be translated to an *EHA* (F, E, ρ) by defining *F*, *E* and ρ respectively.

Taking *EHA* as an intermediate format, aspect-oriented timed statechart could be translated into timed automaton with guards to explain its executable paths. The translation process is as follows:

(1) $Conf(\rho)$ in *EHA* is translated to *X* of *TAG*. A configuration in *EHA* describes a state in *TAG*;

(2) (ΘE) in *EHA* is translated to E_T of *TAG*. An event in *TAG* is a set of clock events or environment events;

(3) $\xrightarrow{STEP}$ in *EHA* is translated to T_T of *TAG*;

(4) The initial status C_0 is translated to the initial state x_0 of *TAG*.

According to the translation principle we know that the initial state of elevator *TAG is* x_0= *{WE, LOCKED}*, x_1=*{WE, UNLOCKED, IDLE}…;* e_0=*{unlock}*, e_1=*{p_up}*…. Initially, $c_0=0, c_1=0, c_2=0, c_3=0, c_4=0, c_5=0, c_6=0, c_7=0, c_8=0$.

From section 2.1 we know that a run of *TAG* is sequence of admissible delay and event transitions, starting from $(x_0, 0)$. Therefore, all executable paths of elevator *TAG* could be described as follow:

P_1: $(x_0, 0), (x_1, 0), (x_2, 0), (x_3, 0), (x_4, 0), (x_5, 0), (x_6, 0), (x_7, 0), (x_8, 0)$ …;
P_2: $(x_0, 0), (x_1, 0), (x_2, 0), (x_3, 0), (x_4, 0), (x_5, 0), (x_5, 1), (x_6, 0), (x_7, 0), (x_8, 0)$ …;
P_3: $(x_0, 0), (x_1, 0), (x_2, 0), (x_3, 0), (x_4, 0), (x_5, 0), (x_6, 0), (x_7, 0), (x_7, 8), (x_8, 0)$…;
P_4: $(x_0, 0)$, …, $(x_3, 0), (x_4, 0), (x_5, 0), (x_5, 1), (x_6, 0), (x_7, 0), (x_7, 8), (x_8, 0)$ …;

From the four executable paths above we know that before type advice event *weigh* happens before joint point event *p_close*, after type advice event *schedule* happens after joint points events *up*, around type advice action *open/close* replaces joint points event *p_open*/*p_close* when satisfying the certain timing constraints $t_a=1/\ t_a=8$. The elevator example illustrates the validity of our weaving approach in section 3.2.

The next work is to check whether the woven model satisfies the key properties of elevator control system or not. Two key properties could be described as follows:

System property 1: An upwards traveling elevator does not change its direction if there are above requests coming from the same direction.

$$\phi_1 = G((\neg x_3 \cup x_2)\wedge(x_3 \rightarrow X(x_4))) \quad (2)$$

System property 2: The doors could not be closed before ensuring the weight of passengers in the car. Once having checked the weight of passengers, the doors must be closed within 8 seconds.

$$\phi_2 = G((\neg x_8 \cup x_7)\wedge(x_7 \wedge (c_7 \geq 8) \rightarrow X(x_8))) \quad (3)$$

From the computational paths (*p1, p2, p3, p4*), it could be concluded that elevator control system satisfies the above system properties: ϕ_1, ϕ_2, that means $WE \models \phi_1 \wedge \phi_2$.

5. Conclusions

This paper proposes a formal method for modeling and verifying real-time systems with AOTSCs and LTL. First, real-time systems requirements are captured by aspect-oriented use case models to improve software modularity. Furthermore, aspect-oriented use case models are formalized as AOTSCs and are translated to TAGs, while systems key properties are described by LTL for model checking. In the end, the elevator example illustates that our model is effective.The next work is applying some optimization techniques to our model to solve the question of state space explosion.

Acknowledgments

This work is supported by the National Natural Science Foundation of China under Grant No. 0773094.

References

[1] S. Clarke and E. Baniassad, "Aspect-Oriented Analysis and Design: The Theme Approach", Addison-Wesley, NJ **(2005)**.

[2] T. M. Lam and W. H. K. Lam, "Application of Automatic Vehicle Identification Technology for Real-Time", Journal of Information Fusion, vol. 12, issue. 1, **(2011)**, pp. 11-19.

[3] S. Schliecker and R. Ernst, "Real-time performance analysis of multiprocessor systems with shared memory", ACM Transactions on Embedded Computing Systems, vol.10, no.2, Article. 22, **(2010)**.

[4] C. G. Cassandras and S. Lafortune, "Introduction to Discrete Event Systems", Springer **(2007)**.

[5] E. M. Clarke, O. Grumberg and D. Peled, "Model Checking", MIT Press **(1999)**.

[6] F. D'Amorim and P. Borba, "Modularity analysis of use case implementations", Journal of Systems and Software, vol. 85, issue 4, **(2012)**.

[7] L. Luo, A UML Documentation for an Elevator System, http://www.cs.cmu.edu/~luluo/Courses/18540PhDreport.pdf .

[8] D. Harel, "Statecharts: A visual formalism for complex systems", Sci. Comput. Program, **(1987)**, pp.231–274.

[9] X. Dianxiang, O. E1-Ariss, X. Weifeng and W. Linzhang, "Testing aspect-oriented programs with finite state machines", Journal of Software Testing, Verification and Reliability, in press.

[10] E. Mikk, Y. Lakhnech and M. Siegel, "Hierarchical automata as model for statecharts", In: Third Asian Computing Science Conference. Advances in Computing Sience -ASIAN'97, vol 1345 of Lecture Notes in Computer Science, **(1997)**, pp. 181-196. Springer-Verlag.

Authors

Xinxiu Wen

She received her M.S. degree in Computer Science from Jilin University in 2003. She is now a Ph.D. candidate in Department of Computer Science and Engineering, East China University of Science and Technology. Her main research interests include software architecture, and formal methods.

Huiqun Yu

He received his M.S. degree in Computer Science from East China University of Science and Technology in 1992, Ph.D. in Computer Science from Shanghai Jiaotong University in 1995. He is now a professor in the Department of Computer Science and Engineering. His research interests include information security, software architecture, and formal methods.