Timed Automata Semantics of Spatial-Temporal Consistency Language STeC And Its Related Verification Techniques

Yuanrui Zhang
Institute of Software Engineering,
East China Normal University,
Shanghai, China
Polytech Nice Sophia,
University Nice Sophia Antipolis,
Nice, France
Email: yuanrui1990.zhang@gmail.com

Frederic Mallet
Univ. Nice Sophia Antipolis,
CNRS, I3S, UMR 7271,
06900 Sophia Antipolis, France
(NOTE: Member of Aoste team (INRIA/I3S)
Email: Frederic.Mallet@unice.fr

Yixiang Chen
MoE Engineering Research Center for
Software/Hardware Co-design
Technology and Application,
East China Normal University,
Shanghai, China
Email: yxchen@sei.ecnu.edu.cn

***Abstract*—Intelligent Transportation Systems (ITS) are a class of quickly evolving modern safety-critical embedded systems. Dealing with their growing complexity demands a high-level formal modeling language along with adequate verification techniques. STeC has recently been introduced as a process algebra that deals natively with both spatial and temporal properties. Even though STeC has the right expressive power, it does not provide a direct support for verification both in theory and application. We propose to encode STeC models as Timed Automata(abbreviated as TA) and introduce CCSL as its specification language on its abstract level to provide such a support. We illustrate our verification strategy on a simple ITS example.**

I. Introduction

As computer systems become more and more complex, high-level formal modeling language is more and more needed to model the high-level behaviours of system and as well as the related verification techniques. A great variety of languages have been developed over the last two decades to describe untimed or timed models (like CSP ([1], [2], π-calculus[3], Timed-CSP[4], Timed Automata[5]) and properties (LTL, CTL, TCTL or PSL[6]).

Timed Automata(TA), introduced by Alur and Dill, have inspired a lot of works as they introduced real-valued clocks thus making 'time' a first-class citizen of the specification languages. Sound and mature verification tools(like uppaal) have also been developed around Timed Automata, which makes them a good 'assembly language' for other high-level modeling languages in search for a verification support.

CCSL—the Clock Constraint Specification Language is firstly proposed by AOSTE team as a companion language of the UML profile for MARTE [7]. It is intended to describe the temporal order of interactions between components of a distributed software system. CCSL is a clock-oriented specification language where each clock represents a sequence of occurrence of movements(or say events) of a single action(distinguished from the concept of clock in Timed Automata. It is firstly proposed on the usage of describing logical relationship between actions of UML profile, but now has been developed and improved as a independent formal specification language with standard mathematical expression. [8] gives its formal syntax and semantics. And [9] introduces more about its background technique it origins from—MARTE. The best advantage of CCSL as a specification language is that it provides a direct support for verification. The state-base representation of CCSL is considered a lot in [10]. [11] gives a transformation from CCSL to Timed Automata, through which we can implement the verification tools for CCSL by transforming it into Timed Automata. It is also a good reason that we use CCSL as our specification language in our new purposed verification framework, which is discussed later. [12] gives the denotational semantics for CCSL.

In embedded systems, time is acknowledged as being of central importance. In the domain of Intelligent Transportation Systems (ITS), which is developing rapidly and is gaining a growing attention, we believe that both models and properties need to consider not only time but also spatial aspects. For example, in a 'Railroad Crossing System' that will be introduced later, a train is always required to receive a message from gate at right location and time. Such an example can be found many places in Intelligent Transportation Systems. That is why we consider STeC (Spatial-Temporal Consistence Language) as a good candidate process algebra to build spatial-temporal models([13], [14]). Indeed, STeC makes (physical) location a first-class citizen of the specification. In this paper, we propose to build a high-level verification framework for spatial-temporal specifications. We use STeC to capture the models and CCSL to specify properties. We rely on Timed Automata to provide the foundational verification support. After discussing a global high-level modeling framework to capture both models and properties, we focus on the definition of CCSL on STeC, using CCSL to express high-level property-specifications of STeC models, and the transformation from STeC to Timed Automata, where we also analysis the bisimilarity between STeC and its corresponding transformed TA. We

* Yixiang Chen: The Communication author

illustrate our work through a simple intelligent transportation system—Railroad Crossing System.

In order to provide a direct verification support for the spatial-temporal models built by STeC. We propose a global verification framework to illustrate our goal and main works. In the framework we use a specification language(called CCSL [7]) to specify 'logical' properties of STeC-models, which is the clock-oriented and easy-understandable language and easy to use in computer industry without any lose of expressive power comparing with LTL and CTL. And we also show globally how to transform a STeC formula into Timed Automata on which verification techniques can be applied. After that we focus on the details of our main work in this paper.

Section 2 gives some background knowledge and definitions, which are essential for introducing the contribution of this paper. In Section 3 we introduce a possible verification framework for verifying STeC-models. In Section 4, we give the details of how to import CCSL as a specification language into STeC. In Section 5, we focus on the transformation from STeC into Timed Automata. And we formally analysis the bisimilarity between STeC and transformed TA in Section 6. In Section 7, we use a simple example of intelligent transportation system to illustrate the modeling and verification technique under the new verification framework. Section 8 summarizes the results.

II. Background and Definitions

In this section, some basic definitions about STeC, CCSL and TA are introduced. These definition are necessary for ~~the contribution part~~. For the traditional definitions and more details, see [5], [13], [15], [16] and [8].

A. An Introduction of STeC

Yixiang Chen introduced the Spatial-Temporal Consistence Language in 2010 [13] and set up its formal semantics with his colleagues in [14]. Unlike the traditional formal modeling languages, it stresses the location and the time of an event of real-time systems, especially ITS. An event (or action) is executed at a location and time. For example, in a Train Scheduling System (see [13], [14]), trains travel from one platform to another with very strict time restriction. If a train do not arrive at a station at a 'correct' time, ~~the~~ collisions might happen. This characteristic of STeC makes it a powerful and ~~an easy understand~~ formal modeling language for ITS (for more examples modeled by STeC, see [13], [14]). However, STeC does not support for verification ~~considering there is no specification language used to describe the spatial-temporal properties and we can not apply traditional verification techniques based on automata directly to the STeC model~~.

The syntax and operational semantics of STeC are given below.

1) The Syntax of STeC: The syntax of STeC is as follows, in Backus-Naur Form:

$$A ::= \mathbf{Send}^{G\rightarrow G'}_{(l,t)}(m) \mid \mathbf{Get}^{G\leftarrow G'}_{(l,t)}(m)$$

$$B ::= \alpha_{(l,t)}(l',\delta) \mid \beta_{(l,t)}(\delta)$$

$$P ::= \mathbf{Stop}^{G}_{(l,t)} \mid \mathbf{Skip}^{G}_{(l,t)} \mid A \mid B \mid P;P \mid P \,[]\, P \mid P \parallel P \mid []_{i\in I} B_i \rightarrow P_i \mid P \trianglerighteq_\delta P \mid B \rightarrow P \mid P \trianglerighteq ([]_{i\in I} A_i \rightarrow P_i)$$

$\mathbf{Stop}^{G}_{(l,t)}$, $\mathbf{Skip}^{G}_{(l,t)}$, $\mathbf{Send}^{G\rightarrow G'}_{(l,t)}(m)$, $\mathbf{Get}^{G\leftarrow G'}_{(l,t)}(m)$, $\alpha^{G}_{(l,t)}(l',\delta)$ and $\beta^{G}_{(l,t)}(\delta)$ are atomic processes, $P;P$ is the sequence operator and $P \parallel P$ is the parallel operator. $P[]P$ is the choice operator. $P \trianglerighteq_\delta Q$ behaves as P for up to δ time units and then is interrupted by Q. $[]_{i\in I} B_i \rightarrow P_i$ is a guard choice. $P \trianglerighteq ([]_{i\in I} A_i \rightarrow P_i)$ initially proceeds like P and is interrupted on occurrence of the atomic command A_i and then process like P_i. Here A_i is an interrupt command.

For more details about the syntax of STeC, refer to [13] and [14].

2) The Operational Semantics of STeC: A storage σ is introduced in STeC as the storage of messages between agents. Let $\sigma \subseteq 2^{\mathbb{M}}$, $\mathbb{M}$ is the set of messages of the form $m_{G\rightarrow G'}$. Here we use $\mathbf{Sto}$ to represent the set of storages and it is easy to see that: $\mathbf{Sto} \subseteq 2^{2^{\mathbb{M}}}$.

Let $\mathbf{Loc}$ be the set of locations. $\mathbb{T} \subseteq \mathbb{R}$ is the set of time ~~space~~. An environment is a triple (a,u,σ) where σ is a storage, a is a location and u is time. We use the notation $\mathcal{E}$ to represent the set of all the environments: $\mathcal{E} = \mathbf{Loc} \times \mathbb{T} \times \mathbf{Sto}$.

Let $\mathbb{B} = \{B | B\ is\ a\ 'B'\ type\ formula\ in\ STeC\}$(see the syntax in STeC). A function $\mathcal{T} : \mathbb{B} \rightarrow \{0,1\}$ is defined as the truth valuation function. We have $\mathcal{T}(B) = 1$ (resp. $\mathcal{T}(B) = 0$) if B is true (resp. false).

A configuration of process P is defined as a tuple $\langle P,(a,u,\sigma)\rangle$. Define $\mathbf{Stconf} = \{\langle P,(a,u,\sigma)\rangle \mid P\ is\ a\ process\ and\ for\ some\ a,\ u,\ \sigma\}$ as the set of all configurations in STeC.

The transition relation of STeC $\hookrightarrow_{STeC}$ is a function:

$$\hookrightarrow_{STeC}: \mathbf{Stconf} \rightarrow 2^{\mathbf{Stconf}}$$

The transition relation ~~can be~~ denoted as $\langle P,(a,u,\sigma)\rangle \rightarrow \langle P',(a',u',\sigma')\rangle$ iff $\hookrightarrow_{STeC} (\langle P,(a,u,\sigma)\rangle) = \langle P',(a',u',\sigma')\rangle$ holds.

τ is a function ~~which~~ maps each process in STeC to the real set $\mathbb{R}$. It computes the duration time for each process. For example, $\tau(\alpha_{(l,t)}(l',\delta)) = \delta$. For more detail, see [13].

The operational semantics of STeC is as follows. For more detail, refer to [13].

- $\frac{a=l,u=t}{\langle \mathbf{Send}^{G\rightarrow G'}_{(l,t)}(m),(a,u,\sigma)\rangle \rightarrow \langle E,(l,t,\sigma\cup\{m_{G\rightarrow G'}\})\rangle}$
- $\frac{a=l,u=t}{\langle \mathbf{Get}^{G\leftarrow G'}_{(l,t)}(m),(a,u,\sigma)\rangle \rightarrow \langle E,(l,t,\sigma\setminus\{m_{G\rightarrow G'}\})\rangle}$
- $\frac{a=l,u=t}{\langle \alpha_{(l,t)}(l',\delta),(a,u,\sigma)\rangle \rightarrow \langle E,(l',t+\delta,\sigma)\rangle}$
- $\frac{a=l,u=t}{\langle \beta_{(l,t)}(\delta),(a,u,\sigma)\rangle \rightarrow \langle E,(l,t+\delta,\sigma)\rangle}$
- $\frac{a=l,u=t}{\langle \mathbf{Stop}^{G}_{(l,t)},(a,u,\sigma)\rangle \rightarrow \langle \mathbf{Stop}^{G}_{(l,t+1)},(l,t+1,\sigma)\rangle}$
- $\frac{a=l,u=t}{\langle \mathbf{Skip}^{G}_{(l,t)},(a,u,\sigma)\rangle \rightarrow \langle E,(l,t,\sigma)\rangle}$
- $\frac{\langle a,(a,u,\sigma)\rangle \rightarrow \langle E,(a',u',\sigma')\rangle}{\langle a\rightarrow P,(a,u,\sigma)\rangle \rightarrow \langle P,(a',u',\sigma')\rangle}$

- $\frac{\langle P,(a,u,\sigma)\rangle \to \langle P',(a',u',\sigma')\rangle}{\langle P;Q,(a,u,\sigma)\rangle \to \langle P';Q,(a',u',\sigma')\rangle}$
- $\frac{\langle P,(a,u,\sigma)\rangle \to \langle P',(a',u',\sigma')\rangle}{\langle P;Q,(a,u,\sigma)\rangle \to \langle P';Q,(a',u',\sigma')\rangle}$
- $\frac{\langle P,(a,u,\sigma)\rangle \to \langle P',(a_1,u_1,\sigma_1)\rangle,\ \langle Q,(a,u,\sigma)\rangle \to \langle Q',(a_2,u_2,\sigma_2)\rangle}{\langle P[]Q,(a,u,\sigma)\rangle \to \{\langle P',(a_1,u_1,\sigma_1)\rangle,\langle Q',(a_2,u_2,\sigma_2)\rangle\}}$
- $\frac{\langle P,(a,u,\sigma)\rangle \to \langle P',(a',u',\sigma_1)\rangle,\ \langle Q,(a,u,\sigma)\rangle \to \langle Q',(a',u',\sigma_2)\rangle}{\langle P\|Q,(a,u,\sigma)\rangle \to \langle P'\|Q',(a',u',\sigma_1 \uplus \sigma_2)\rangle}$
- $\frac{\mathcal{T}(B_i)=}{\langle []_{i\in I} B_i \to P_i,(a,u,\sigma)\rangle \to \langle P_i,(a,u,\sigma)\rangle}$
- $\frac{\langle P,(a,u,\sigma)\rangle \to \langle P',(a',u',\sigma')\rangle \ \wedge\ u'\leq\delta}{\langle P \trianglerighteq_\delta Q,(a,u,\sigma)\rangle \to \langle P \trianglerighteq_{\delta-u'} Q,(a',u',\sigma')\rangle}$
  $\frac{\langle P,(a,u,\sigma)\rangle \to \langle P',(a',u',\sigma')\rangle \ \wedge\ u'>\delta}{\langle P \trianglerighteq_\delta Q,(a,u,\sigma)\rangle \to \langle Q,(a_1,\delta,\sigma_1)\rangle}$
- $\frac{\mathcal{T}(B)=1}{\langle B \to P,(a,u,\sigma)\rangle \to \langle P,(a,u,\sigma)\rangle}$
  $\frac{\mathcal{T}(B)=0}{\langle B \to P,(a,u,\sigma)\rangle \to \langle \mathbf{Stop}^G_{(l,\ldots)},(a,u,\sigma)\rangle}$
- $\frac{\langle A_i,(a,u,\sigma)\rangle \to \langle E,(a',}{\langle P \trianglerighteq ([]_{i\in I} A_i \to P_i)\rangle \to \langle P_i,(a',u',\sigma')\rangle}$

B. An Introduction of CCSL

~~CCSL~~ was ~~firstly~~ proposed by ~~professor~~ Charles Andre and Frederic Mallet in 2008[7]~~. It is firstly designed to be used in the specification verification for the UML profile of MARTE~~(see [17]). ~~Now it has been developed as an independent domain specific language on its own with formal syntax and semantics.~~ The rest of this section focus on ~~the~~ basic concepts ~~on it~~. For more details, see the relative references.

1) Clock: A Clock in CCSL is a tuple $\langle \mathcal{I}, \prec, \mathcal{D}, \lambda, \mu \rangle$, where $\mathcal{I}$ is a set of instants(possibly infinite), $\prec$ is a quasi-order relation on $\mathcal{I}$, $\mathcal{D}$ is a set of labels, $\lambda : \mathcal{I} \to \mathcal{D}$ is a labelling function, μ is a symbol, standing for a 'unit' of time.

A discrete-time clock in CCSL is a clock with a descrete set of instants $\mathcal{I}$. Since $\mathcal{I}$ is discrete, it can be indexed by natural numbers in a way that respects the ordering on $\mathcal{I}$: set $\mathbb{N}^* = \mathbb{N}\backslash\{0\}$, $idx : \mathcal{I} \to \mathbb{N}^*$, $\forall i \in \mathcal{I}, idx(i) = k$ if and only if i is the k^{th} instant in $\mathcal{I}$.

Set ~~$\mathcal{C}$~~ $= \langle \mathcal{I}, \prec, \mathcal{D}, \lambda, \mu \rangle$ denoting a CCSL clock. ~~$\mathcal{C}$~~$[k]$ denotes the k^{th} element in $\mathcal{I}$. Then we have $k = idx($~~$\mathcal{C}$~~$[k])$.

2) Clock Constraint: In CCSL, we use 'clock' to denote the occurrence of events in a system, binary operations between clocks describe the logical relationship between clocks. They are divided into two basic type of operations, one called 'Sub Clock' and the other is 'Precedence'. All the binary operations in CCSL can be deducted by these two type of basic operations. For example, binary operations like 'Equality', 'Restriction', 'Discretization', 'Filtering' can be deducted by 'Sub Clock'. And operations such as 'Speed', 'Alternation', 'Synchronization' can be deducted by 'Precedence'. For more details, see [7].

'**Sub Clock**' is a binary relation between two clocks which can be defined as this: let ~~$\mathcal{C}_1$~~ $= \langle \mathcal{I}_1, \prec_1, \mathcal{D}_1, \lambda_1, \mu_1 \rangle$, ~~$\mathcal{C}_2$~~ $= \langle \mathcal{I}_2, \prec_2, \mathcal{D}_2, \lambda_2, \mu_2 \rangle$ be two clocks, we say ~~$\mathcal{C}_1$~~ $\subseteq$ ~~$\mathcal{C}_2$~~ hold if and only if $\exists h : \mathcal{I}_1 \to \mathcal{I}_2$ such that:

- h is injective.
- h is order preserving: $(\forall i, j \in \mathcal{I}_1)(i \prec_1 j) \to (h(i) \prec_2 h(j))$.
- an instant and its image are coincident: $(\forall i \in \mathcal{I}_1) i = h(i)$.

'**Precedence**' is a binary relation between two clocks which can be defined as follows: let $\mathcal{C}_1 = \langle \mathcal{I}_1, \prec_1, \mathcal{D}_1, \lambda_1, \mu_1 \rangle$, $\mathcal{C}_2 = \langle \mathcal{I}_2, \prec_2, \mathcal{D}_2, \lambda_2, \mu_2 \rangle$ be two clocks, we say '$\mathcal{C}_1$ precedence' $\mathcal{C}_2$' which is denoted by $\mathcal{C}_1 \prec \mathcal{C}_2$ holds if and only if $\exists h : \mathcal{I}_2 \to \mathcal{I}_1$ such that:

- h is injective.
- h is order preserving: $(\forall i, j \in \mathcal{I}_2)(i \prec_2 j) \to (h(i) \prec_1 h(j))$.
- an instant and its image are ordered: $(\forall i \in \mathcal{I}_2)(h(i) \prec i)$.

From these two basic binary operations, we can see that the relationship between clocks only concern the ordered set $\mathcal{I}$, and have nothing to do with the domain set $\mathcal{D}$ and label function λ. Next in this paper we will see that it is a good property when defining CCSL for STeC.

$\mathcal{A} = \mathcal{B}$ means that the two clocks are 'synchronous', clock $\mathcal{A}$ ticks whenever clock $\mathcal{B}$ ticks and vice versa. $\mathcal{A}$ **Alternate** $\mathcal{B}$ means that two clocks ticks 'alternately', in other words, for all $i \in \mathcal{I}_A$, set $k = idx_A(i)$, then we have $\mathcal{A}[k] \preceq \mathcal{B}[k] \prec \mathcal{A}[k+1]$ if $\mathcal{I}_A$ and $\mathcal{I}_B$ are in the same domain(which means that $\mathcal{I}_A$ and $\mathcal{I}_B$ is comparable by an operator '$\prec$'). $\mathcal{B} = \mathcal{A}\ \$\ n$ where $n \in \mathbb{N}$ means that clock $\mathcal{A}$ is delayed n units of times comparing with clock $\mathcal{B}$, that is to say, clock $\mathcal{A}$ 'mimic' clock $\mathcal{B}$'s behaviours after n units of times when clock $\mathcal{B}$ starts. For the detail of binary operations of CCSL, see [7], and for more other information about CCSL, refer to [6], [8], [9], [11], [12] and [17].

C. An Introduction of Timed Automata[5]

Properly speaking, ~~the~~ well-defined Timed Automata has the same expression power ~~with the~~ deterministic finite automata(DFA), ~~and it just takes~~ 'time' ~~as the first citizen~~. ~~The expression power of different kind of automata are stated in~~ [18].

1) The Syntax of TA: Let $\mathcal{C}$ be a finite set of non-negative real-valued variables called clocks. The set of guards $G(\mathcal{C})$ is defined by the grammar $g := x \bowtie c \mid g \wedge g$ where $g \in \mathcal{C}$, $c \in \mathbb{N}$ and $\bowtie \in \{<, \leq, >, \geq, ==\}$. A Timed Automata is a tuple $A = (Q, \Sigma, \mathcal{C}, q_0, E, I, AP, L, F)$, where:

- Q is a finite set of states,
- Σ is a finite alphabet,
- $\mathcal{C}$ is a finite set of clocks,
- $q_0 \in Q$ is an initial state,
- $E \subseteq Q \times \Sigma \times G(\mathcal{C}) \times 2^{\mathcal{C}} \times 2^Q$ is a finite transition relation,
- $I : Q \to G(\mathcal{C})$ is an invariant-assignment function,
- AP is a finite set of atomic propositions,
- $L : Q \to 2^{AP}$ is a labeling function for the states,
- $F \subseteq Q$ is a set of accepting states.

2) The Operational Semantics of TA: A clock valuation is a function $\nu : \mathcal{C} \to \mathbb{T}$. We define $\nu + r$ as: for each clock $x \in \mathcal{C}$, $(\nu + r)(x) = \nu(x) + r$, where $r \in \mathbb{T}$. If $Y \subseteq \mathcal{C}$ then a valuation $\nu[Y := 0]$ is such that for each clock $x \in \mathcal{C} \backslash Y$, $\nu[Y := 0](x) = \nu(x)$ and for each clock $x \in Y$, $\nu[Y := 0](x) = 0$. The satisfaction relation $\nu \models g$ for $g \in G(\mathcal{C})$ is defined in the natural way.

The operational semantics of a TA $A = (Q, \Sigma, \mathcal{C}, q_0, E, I, AP, L, F)$ is a labeled transition system (LTS):

$$\hookrightarrow_A (q, \nu) : (Q \times \mathbb{T}^{\mathcal{C}})_\perp \to 2^{(Q \times \mathbb{T}^{\mathcal{C}})_\perp}$$

where each mapping ~~defines~~ a transition relation $\langle q, \nu \rangle \to \langle q', \nu' \rangle$, which is defined as follows:

$$\hookrightarrow_A (q, \nu) = \begin{cases} \{\langle q, \nu + \alpha \rangle\} & \text{if } \exists \alpha \in \mathbb{T} \text{ ~~.~~ } (\nu + \alpha \models I) \\ \{\langle q', \nu' \rangle \mid q' \in Q'\} & \text{if } \exists \langle q, a, g, Y, Q' \rangle \in E \\ & \text{where } \nu \models g, \nu' = \nu[Y := 0] \\ \{\langle q', \nu' \rangle \mid q' \in Q'\} \cup \{\langle q, \nu + \alpha \rangle\} & \text{if both} \\ \perp & \text{otherwise} \end{cases}$$

3) Transition Relation Function: The transition relation function $\psi : Q \to 2^Q$ is a function over the states of TA. We declare that for any $q \in Q$, there exists a value of $\psi(q)$ if and only if there exists a tuple $\langle q, a, g, Y, Q' \rangle \in E$, denoted as $\psi(q) = Q'$. So we can say for each TA there is a co~~r~~rrespondent transition relation function.

III. The Verification Framework of Spati~~al~~-Temporal Models

Classic model checking techniques deal~~t~~ with a formal model $\mathcal{M}$ and ~~its~~ specifications ~~φ~~ (also called properties). As shown in Fig. 1, a formal model is usually described with a process algebra like CSP, CCS, Timed CSP~~, etc~~, while a formal formula is usually described as a type of modal logic such as LTL, CTL, TCTL, or PSL. ~~After that the equivalent transition systems of a formal model and a formula are considered for using specific specification algorithms. We often call the transition system of a formal formula 'observer' different from that of formal models because it usually contains 'final states'(or 'accept states') from which if an error path can be reached we say an counter-example which make our specification failed is found. Different verification techniques has been found and applied in history for different specifications. For example, a LTL formula is usually transformed into a Buchi Automata and combined with the automata of model by doing cartesian product, and an 'counter-example' path is checked by traversing every path from the 'accept states' of it. For a CTL formula, things get more efficient since CTL is a language base on branching trees. The verification algorithm for CTL formula on which many verification tools like Uppaal, PAT, etc based is carried on according to the tree structure of the syntax of CTL formulas.~~

~~The classic verification framework shown in Fig. 1 has been developed for many years and is very mature both in theory and application. So we enhance STeC model to be directly verifiable by transforming it into Timed Automata - a very powerful tool directly support for verification ([19] introduces the basic verirification technique of TA). We introduce a specification language, called CCSL and use it to specify properties of STeC models. In CCSL the concept of 'clock' of each event was introduced and the complexity logical relations between events are expressed by the composition of such 'clock'. Because of this, some properties are stated in a simpler way using CCSL than using other temporal logic specification languages (such as LTL or CTL). And CCSL in fact is not exactly a subset of PSL [6]. Encoding directly a 'subset' of CCSL into timed automata is not so straightforward, recent work [11] has shown that safe CCSL specifications could be compiled as a timed automaton as a result of the clock calculus process.~~

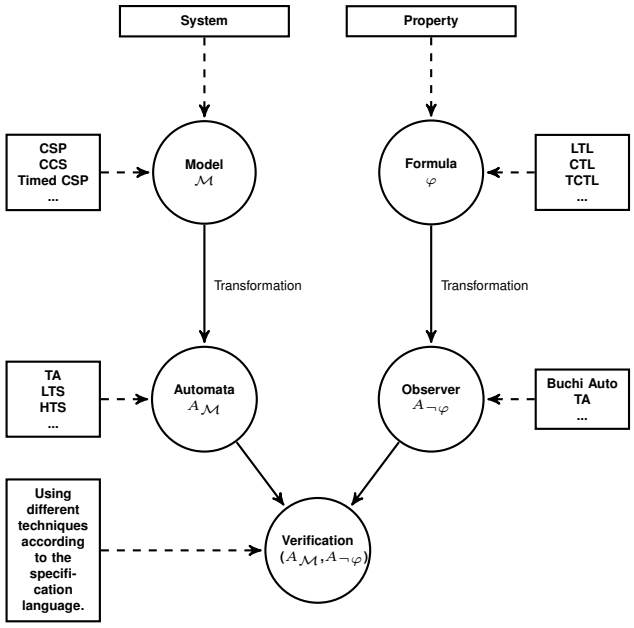

Fig. 1. Classic Verification Framework

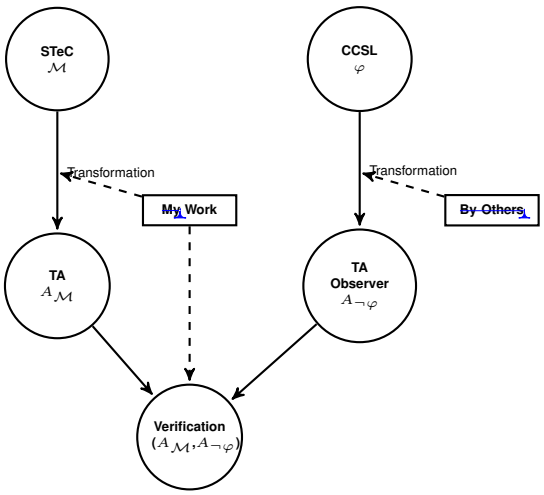

Fig. 2. P~~ur~~posed ~~New~~ Verification Framework

Fig. 2 shows the framework we ~~propose to build~~ in order to

verify spatial-temporal specifications of STeC models: i.e., by transforming both STeC and CCSL into Timed Automata(TA). In this paper, we focus on the transformation of STeC into Timed Automata and the definition of CCSL in the domain of STeC. We use a simple example to illustrate the modeling and verification techniques of the new verification framework.

IV. The Definition of CCSL in the Domain of STeC

In this section, we import CCSL into STeC as a specification language. The key idea is to define CCSL based on the domain of STeC, which is quite trivial, and explain(formally define) how a STeC formula 'satisfy' a CCSL formula. For this purpose we firstly give a definition of traces for STeC in a nature way.

A. Traces on STeC

Like what it is done for other process algebra, we define a trace in STeC as a sequence of events(or actions) which a STeC system (or say a formula) can generate. With it we can import CCSL later and further it lead the road to the simulation of the behaviours of STeC-described system using Timed Automata. It is necessary when we further consider the relationship between STeC and Timed Automata. It can be defined on the structure of syntax of STeC.

Definition IV.1. *For any atomic process of STeC a, use αa denote its alphabet. Use αP ($P \in \mathbb{P}_{STeC}$) to represent the set of actions in process P. An action in an atomic process defined in Section 2 is just an alphabet. For example, for an atomic process $Send_{(Lapp,t,G)}(Appr)$, its action is 'Send', denoted as $\alpha Send_{(Lapp,t,G)}(Appr) = Send$.*

One more example, let $P = Run(\infty);(Send_{(Lapp,t,G)}(Appr) \parallel Approach_{(Lapp,t)}(\Gamma))$. Then $\alpha P = \{Run, Send, Approach\}$.

Definition IV.2. *An element of trace in STeC, called a STeC word, is a quadruple $e = \langle act, l, t, \delta \rangle$, where act is an atomic process in STeC, l is location, t is time and δ is the duration of action.*

Denote the set of all elements in STeC as $\mathbb{E}_{STeC}$.

Define the projection of each item in element e as π_{act}, π_l, π_t, ... respectively. Generally, let $e = \langle e_1, e_2, ..., e_i, ... \rangle$, we just define π_i be the projection on the ith item of e. (e.g, $\pi_2(\langle e_1, e_2, e_3 \rangle) = e_2$, $\pi_{act}(\langle a_1, l, t, \delta \rangle) = a_1$).

Definition IV.3. *Denote the set of all words of a process P as* $\mathbf{Words}(P)$.

Different from 'actions' in STeC, a 'word' in STeC is a tuple which not only contains the information of the alphabet of an action, but also the location where an action happens, the time it occurs and the duration it would go through. For example, let $P = Run(\infty);(Send_{(Lapp,t,G)}(Appr) \parallel Approach_{(Lapp,t)}(\Gamma))$. Then $\mathbf{Words}(P) = \{\langle Run, l_0, t_0, \infty \rangle, \langle Send, Lapp, t, 0 \rangle, \langle Approach, Lapp, t, \Gamma \rangle\}$.

Definition IV.4. *A trace of STeC is a finite or infinite sequence of words which is defined as a partial function $t : \mathbb{N}^+ \to \mathbb{E}_{STeC}$ ($\mathbb{N}^+ = \{1, 2, 3, ...\}$ is the set of all natural numbers), where $dom(t) \subseteq \mathbb{N}^+$. It is a partial, injective function and satisfies: for any $n \in \mathbb{N}^+$, if $n \in dom(t)$, then $m \in dom(t)$ for all $m < n$.*

Denote the set of all traces in STeC as $\mathbb{T}_{STeC}$.

Definition IV.5. *A binary operator $\frown : \mathbb{T}_{STeC} \times \mathbb{T}_{STeC} \to \mathbb{T}_{STeC}$ concatenates two traces. Let s, t be any traces in STeC, s is finite. Set $|dom(s)| = m$, then we have a new trace $r = s \frown t$, which satisfies:*

$$r(x) = \begin{cases} s(x) & \text{if } x \leq m \\ t(x-m) & \text{otherwise} \end{cases}$$

Definition IV.6. *Let $s, t \in \mathbb{T}_{STeC}$, we denote $s \sqsubseteq t$ when s is a sub trace of t. It satisfies:*

1) $ran(s) \subseteq ran(t)$
2) *for any $t(i), t(j) \in ran(t)$ and $i \leq j$, if there exists $m, n \in \mathbb{N}$ such that $s(m) = t(i), s(n) = t(j)$, then $m \leq n$ holds.*

A restriction on trace is a sub trace of it on some specific alphabet set. We have the following definition.

Definition IV.7. *Let $s, t \in \mathbb{T}_{STeC}$, we say s is a restriction of t on an alphabet set αP if and only if s is a sub trace of t where each element belongs to αP, denoted as $s = t \upharpoonright \alpha P$. It satisfies:*

1) $s \sqsubseteq t$.
2) $ran(s) = \{\langle act, l, t, \delta \rangle \mid act \in \alpha P\}$.

After some preparations, we now give a definition of traces for STeC formulas.

Definition IV.8. *Denote the set of all processes (formulas) of STeC as:*

$$\mathbb{P}_{STeC} = \{\ P \mid \ P \text{ is a process in STeC}\}$$

Definition IV.9. *A trace of a STeC formula is a function from the set of STeC formulas to a set of STeC traces, denoted as* $\mathbf{Traces} : \mathbb{P}_{STeC} \to \mathbb{T}_{STeC}$. *It is defined according to following lows:*

1) $\mathbf{Traces}(\mathbf{Stop}^G_{(l,t)}) = \emptyset$.
2) $\mathbf{Traces}(a_{(l,t)}) = \{\langle a, l, t, \delta \rangle\}$ *(where a can be* $\mathbf{Skip}, \mathbf{Send}, \mathbf{Get}$ *or any atomic actions. The value of δ depends on the specific atomic action).*
3) $\mathbf{Traces}(P; Q) = \{s \frown t \mid s \in Traces(P) \wedge t \in Traces(Q)\}$.
4) $\mathbf{Traces}(P \ [] \ Q) = Traces(P) \cup Traces(Q)$.
5) $\mathbf{Traces}(P \parallel Q) = \{t \mid t \upharpoonright \alpha P \in Traces(P) \wedge t \upharpoonright \alpha Q \in Traces(Q) \wedge ran(t) \subseteq \alpha P \cup \alpha Q\}$. *(αP is the set of all words in process P).*
6) $\mathbf{Traces}(P \trianglerighteq_\delta Q) = \{t \mid$ *if $\pi_t(t) < \delta$, then $\pi_{act}(t) \in \alpha P$ else $\pi_{act}(t) \in \alpha Q$}$\}$
7) $\mathbf{Traces}(P \trianglerighteq ([]_{i \in I} A_i \to P_i)) = \{t \mid$ *if $\pi_{act}(t(i)) = \alpha A_i$, then for any $n > i$ we have $\pi_{act}(t(n)) \in \alpha P_i$, or $\pi_{act}(t(i)) \in \alpha P$ for all $i \in \mathbb{N}^+$.* $\}$

8) $\mathbf{Traces}(B \rightarrow P) = \begin{cases} \{t \mid \pi_{act}(t(1)) = \alpha B\} & \text{if } \mathcal{T}(B) = 1 \\ \emptyset & \text{otherwise} \end{cases}$

~~Easy to know that~~ $[]_{i\in I} B_i \rightarrow P_i$ can ~~easily~~ be considered ~~as~~ a special case of $P[]Q$.

B. The Definition of CCSL for STeC

Next let us define the CCSL in the domain of STeC formally according to the definition of STeC traces.

Definition IV.10. *Let $P \in \mathbb{P}_{STeC}$ be any process of STeC, $A \subseteq \alpha P$ be any subset of the set of all actions of P. $C_A = \langle \mathcal{I}, \prec, \lambda, \mathcal{D} \rangle$ is a clock of A in P, which satisfies:*

1) *$\mathcal{I}$ is a set of instants, $\prec$ is a quasi-order relation on $\mathcal{I}$ just as defined in the early reference [7].*
2) *$\mathcal{D} = A$ which is a set of words in process P.*
3) *$\lambda : \mathcal{I} \rightarrow \mathcal{D}$ is a labelling function. It is a partial, injective function and satisfies that for any $i \in \mathcal{I}$, if $i \in dom(\lambda)$ then for all $j \prec i$, $j \in dom(\lambda)$. This definition is the same as the definition of λ in [7].*

~~We call such a~~ clock ~~which~~ satisfies the above rules is a ~~clock~~ in the domain of STeC.

~~Since the CCSL in the domain of STeC is just the same as the original one except the unit time μ, so, the definition of binary operators of CCSL are in fact the same as the definition in [7]. So we have the following declaration.~~

Definition IV.11. *The definition of the 'Coincidence-based' clock constraint, and the 'Precedence-based' clock constraint which are defined in p[7] remain the same for the definition of operations of CCSL clock in the domain of STeC.*

At last, we give the definition of the satisfiability of CCSL formula for a STeC formula.

Definition IV.12. *$C_A = \langle \mathcal{I}, \prec, \lambda, \mathcal{D} \rangle$ is a CCSL clock and Exp_C is an expression obtained from the combination of clocks using binary operators in CCSL. Then for any traces t (of some process) in STeC,*

1) *$t \models C_A$ iff $t \upharpoonright A \sqsubseteq \lambda$*
2) *If $t \models C_A$, $t \models C_B$, $graph(t \upharpoonright A) \subseteq graph(t \upharpoonright B)$ and $\pi_t(t \upharpoonright A(i)) = \pi_t(t \upharpoonright B(i))$ for any $i \in \mathbb{N}^+$, then $t \models C_A \subseteq C_B$*
3) *If $t \models C_A$, $t \models C_B$ and $\pi_t((t \upharpoonright A)(i)) < \pi_t((s \upharpoonright B)(i))$ for each $i \in \mathbb{N}^+$, then $t \models C_A \prec C_B$*

For any $P \in \mathbb{P}_{STeC}$, $P \models C_A$ (resp. Exp_C) iff $t \models C_A$ (resp. Exp_C) for all $t \in \mathbf{Traces}(P)$.

'*graph*ans the graph of functions. Note that operator $\sqsubseteq$ being used for λ is correct if we co[illegible]er $\mathcal{I}$ as a isomorphic set of $\mathbb{N}^+$ (in fact they are, since we only consider $\mathcal{I}$ as a countable set in almost most cases).

V. The Transformation From STeC to TA

In this section, we mainly focus on the transformation from STeC to TA. We want to build a deductive mapping on the structure of ~~the syntax of~~ STeC. But firstly, we have to introduce three binary operations on TAs ~~in order to get a new TA from the old ones. Later in the~~ Definition IV.6, ~~we see how to apply these three operations on the transformation~~.

~~In this section, we do NOT consider the binary operation~~

A. New Binary Operators for Timed Automata

Firstly, we introduce some binary operations on TAs, which are essential for ~~building~~ the transformation from STeC to TA.

1) STeC-~~Concatenation~~: We introduce a~~n~~ binary operator in TA, denoted as $\diamond_{STeC}$. ~~Later in Definition IV.6, we can see that for every $R = P;Q$, we have $A_R = A_P \diamond_{STeC} A_Q$, where A_* are the corresponding Timed Automatas of process $*$.~~

Definition V.1. ~~*Set*~~ *$A = (Q, \Sigma, \mathcal{C}, q_0, E, I, AP, L, F)$, $A' = (Q', \Sigma', \mathcal{C}', q_0', E', I', AP', L', F')$ ~~are~~ two TAs. And ψ and ψ' are their transition relation functions correspondingly. We ~~define~~ an operation called "STeC-Concatenation" ~~of two TAs as a special case of the operator "Concatenation" of TA. It is defined as~~*

$$\diamond_{STeC} : \mathbf{TA} \times \mathbf{TA} \rightarrow \mathbf{TA}$$

where **TA** *represents the set of all TAs. Let $A'' = (Q'', \Sigma'', \mathcal{C}'', q_0'', E'', I'', AP'', L'', F'')$ be ~~a TA which satisfies~~ $A'' = A \diamond_{STeC} A'$. ~~We define it as follows:~~*

- ~~$A'' = (Q \cup Q', \Sigma \cup \Sigma', \{x, y\} \cup \mathcal{C} \cup \mathcal{C}', q_0, E'', I \cup I', AP \cup AP', L'', F')$. clock x records the 'global' time from the start of TA. Clock y record the 'duration' of each action and is set to zero after each transition(we will see later).~~
- *The transition relation function $\psi''(q)$ is defined as:*

$$\psi''(q) = \begin{cases} \psi(q) & \text{if } q \in Q \wedge \{q\} \cap F = \emptyset \\ \psi(q) \cup \psi'(q_0') & \text{if } q \in Q \wedge \{q\} \cap F \neq \emptyset \\ & \quad \text{OR } q \in Q' \wedge q = q_0' \\ \psi'(q) & \text{if } q \in Q' \cap q \neq q_0' \end{cases}$$

- *The labelling function L'' is defined as:*

$$L''(q) = \begin{cases} L(q) & \text{if } q \in Q \wedge q \notin F \\ L(q) \cup L'(q_0') & \text{if } q \in F \\ & \quad \text{OR } q \in Q' \wedge q = q_0' \\ L'(q) & \text{if } q \in Q' \wedge q \neq q_0' \end{cases}$$

2) STeC-~~Union~~: ~~Denoted as $\oplus_{STeC}$, like $\diamond_{STeC}$, it corresponds to the "choice" operator in STeC, for example, $P[]Q$.~~

Definition V.2. *Let $A = (Q, \Sigma, \mathcal{C}, q_0, E, I, AP, L, F)$, $A' = (Q', \Sigma', \mathcal{C}', q_0', E', I', AP', L', F')$ be two TAs. Define*

$$\oplus_{STeC} : \mathbf{TA} \times \mathbf{TA} \rightarrow \mathbf{TA}$$

as the "STeC-union" as ~~a special case of the "union" operator of TA, which satisfies:~~

- *Let $A'' = A \oplus_{STeC} A'$, then $A'' = (Q \cup Q' \cup \{q_0''\}, \Sigma \cup \Sigma', \{x, y\}, q_0'', E'', I'', AP \cup AP', L'', F \cup F')$*
- *We define the transition relation function ψ''. $\psi''(q)$ is defined as:*

$$\psi''(q) = \begin{cases} \psi(q) & \text{if } q \in Q \\ \psi'(q) & \text{if } q \in Q' \\ \{q_0, q_0'\} & \text{if } q = q_0'' \end{cases}$$

- *The labelling function L'' is defined as:*

$$L''(q) = \begin{cases} L(q) & \text{if } q \in Q \\ L'(q) & \text{if } q \in Q' \\ \emptyset & \text{if } q = q_0'' \end{cases}$$

- *The invariant-assignment function I'' is defined as:*

$$I''(q) = \begin{cases} I(q) & \text{if } q \in Q \\ I'(q) & \text{if } q \in Q' \\ \{y \leq 0\} & \text{if } q = q_0'' \end{cases}$$

3) STeC-HandShaking: Denoted as $\otimes_{STeC}$, it corresponds to the "parallel" operator in STeC, such as $P \mid\mid Q$.

Definition V.3. *Knowing $A = (Q, \Sigma, \mathcal{C}, q_0, E, I, AP, L, F)$, $A' = (Q', \Sigma', \mathcal{C}', q_0', E', I', AP', L', F')$ are two TAs. The definition of "STeC-Union" is given as follows:*

$$\otimes_{STeC} : \mathbf{TA} \times \mathbf{TA} \to \mathbf{TA}$$

set $A'' = A \otimes_{STeC} A'$, ~~it satisfies the following properties~~:

- $A'' = (Q \times Q', \Sigma \cup \Sigma', \{x, y\}, \langle q_0, q_0' \rangle, E'', I'', AP \cup AP', L'', F \times F')$
- *Define the transition relation function ψ''. $\psi''(\langle q_1, q_2 \rangle)$ is defined as:*

$$\psi''(\langle q_1, q_2 \rangle) =$$

$$\begin{cases} \langle \psi(q_1), \psi'(q_2) \rangle & \text{if } \langle q_1, q_2 \rangle \in Q \times Q' \wedge \\ & \exists \langle q_1, m!, g, Y, Q \rangle \in E \text{ and} \\ & \exists \langle q_2, m?, g', Y', Q' \rangle \in E' \\ & \textit{OR} \\ & \exists \langle q_1, m?, g, Y, Q \rangle \in E \text{ and} \\ & \exists \langle q_2, m!, g', Y', Q' \rangle \in E' \\ \{ \langle \psi(q_1), \psi'(q_2) \rangle, \langle \psi(q_1), q_2 \rangle, \langle q_1, \psi(q_2) \rangle \} & \text{if } \langle q_1, q_2 \rangle \in Q \times Q' \wedge \\ & \exists \langle q_1, a, g, Y, Q \rangle \in E \text{ and} \\ & \exists \langle q_2, a', g', Y', Q' \rangle \in E' \\ \langle \psi(q_1), q_2 \rangle & \text{if } \langle q_1, q_2 \rangle \in Q \times Q' \wedge \\ & \exists \langle q_1, \_, \_, Y, Q \rangle \in E \\ \langle q_1, \psi(q_2) \rangle & \text{if } \langle q_1, q_2 \rangle \in Q \times Q' \wedge \\ & \exists \langle q_2, a', g', Y', Q' \rangle \in E' \end{cases}$$

where m is any message in STeC.

- *The labelling function L'' is defined as:*

$$L''(\langle q_1, q_2 \rangle) = L(q_1) \cup L'(q_2) \text{ for all } q_1 \in Q \text{ and } q_2 \in Q'$$

- *The invariant-assignment function I'' is defined as:*

$$I''(\langle q_1, q_2 \rangle) = I(q_1) \cup I'(q_2) \text{ for all } \langle q_1, q_2 \rangle \in Q \times Q'$$

B. From SteC to TA: Induction on the Structure of STeC Specification

After the essential operators ~~are~~ defined, we ~~are trying to define~~ a mapping from STeC to TA. We need to define a ~~homo~~homomorphism from the set of language STeC to the set of TA inductively.

Definition V.4. *Denote the set of all TAs as:*

$$\mathbb{M}_{TA} = \{ A \mid A \text{ is a Timed Automata} \}$$

Definition V.5. *Knowing that $\mathbb{P}_{STeC}$ and $\mathbb{M}_{TA}$ are the sets of processes of STeC and TAs respectively. We define a homomorphism $\mathcal{F} : \mathbb{P}_{STeC} \to \mathbb{M}_{TA}$ which can be inductively built according to the following rules:*

i)

$$\mathcal{F}(\mathbf{Send}_{(l,t)}^{G \to G'}(m)) = A =$$
$$(\{q_0, q_1\}, \{m!\}, \{x, y\}, q_0, E, I, \{l\}, L, \{q_1\}).$$

A is a TA. $\langle q_0, m!, \{x == t\}, \{y\}, q_1 \rangle$ is the only element in E. $I = \emptyset$. $AP = \{l\}$, $x == t$ is a guard, meaning that only when the global clock x equals t~~, transition happens~~. And L is a labelling function which satisfies(Fig. 3 shows the graph of the TA):

$$L(q) = \begin{cases} l & \text{if } q \in \{q_0, q_1\} \\ \emptyset & \textit{otherwise} \end{cases}$$

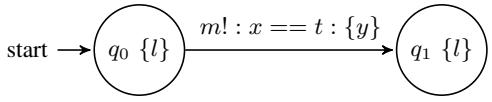

Fig. 3. The TA of $\mathbf{Send}_{(l,t)}^{G \to G'}(m)$

ii)

$$\mathcal{F}(\mathbf{Get}_{(l,t)}^{G \leftarrow G'}(m)) =$$
$$(\{q_0, q_1\}, \{m?\}, \{x, y\}, q_0, E, I, \{l\}, L, \{q_1\})$$

where $\langle q_0, m?, \{x == t\}, \{y\}, q_1 \rangle$ is the only element in E. $I = \emptyset$. $AP = \{l\}$. $x == t$ is a guard. And L is a labelling function which satisfies:

$$L(q) = \begin{cases} l & \text{if } q \in \{q_0, q_1\} \\ \emptyset & \textit{otherwise} \end{cases}$$

Fig. 4 shows the TA of $\mathbf{Get}_{(l,t)}^{G \leftarrow G'}(m)$.

iii)

$$\mathcal{F}(\alpha_{(l,t)}^{G}(l', \delta)) =$$

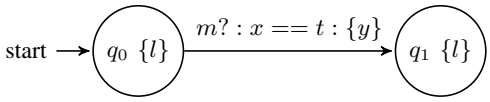

Fig. 4. The TA of $\mathbf{Get}_{(l,t)}^{G \leftarrow G'}(m)$

$$(\{q_0, q_1\}, \{\alpha\}, \{x, y\}, q_0, E, I, \{l, l'\}, L, \{q_1\})$$

.

where $\langle q_0, \alpha, x == t + \delta \wedge y == \delta, \{y\}, q_1 \rangle$ *is the only element in* E. $I(q_0) = \{y \leq \delta\}$. $AP = \{l, l'\}$, L *is a labelling function which satisfies:*

$$L(q) = \begin{cases} l & \text{if } q = q_0 \\ l' & \text{if } q = q_1 \\ \emptyset & \text{otherwise} \end{cases}$$

Fig. 5 shows the TA of atomic commmand $\alpha_{(l,t)}^{G}(l', \delta)$.

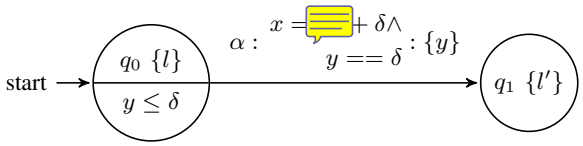

Fig. 5. The TA of $\alpha_{(l,t)}^{G}(l', \delta)$

Similarly, we have the TA for $\beta_{(l,t)}^{G}(\delta)(a, u, \sigma)$*:*

iv) $\mathcal{F}(\beta_{(l,t)}^{G}(\delta)) = (\{q_0\}, x == t + \delta, \{x, y\}, q_0, E, I, \{l\}, L, \{q_0\})$.

The TA has only 1 state—q_0. $E = \langle q_0, \alpha, x == t + \delta \wedge y == \delta, \{y\}, q_0 \rangle$ *and* $I(q_0) = \{y \leq \delta\}$. $L(q_0) = l$ *(see Fig. 6).*

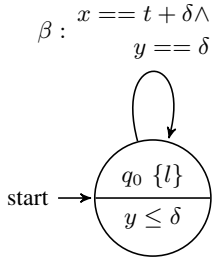

Fig. 6. The TA of $\beta_{(l,t)}^{G}(\delta)$

v)

$$\mathcal{F}(B \to P) = \begin{cases} \mathcal{F}(B) \diamond_{STeC} \mathcal{F}(P) & \text{if } \mathcal{T}(B) = 1 \\ \emptyset & \text{otherwise} \end{cases}$$

vi) For all $R \in \mathbb{P}_{STeC}$*, if* $R = P \ ; \ Q$*, then*

$$\mathcal{F}(R) = \mathcal{F}(P; Q) = \mathcal{F}(P) \diamond_{STeC} \mathcal{F}(Q) = A_P \diamond_{STeC} A_Q$$

if and only if $\mathcal{F}(P) = A_P$ *and* $\mathcal{F}(Q) = A_Q$ *hold. It is according to the of* $\mathcal{F}$*'s preservation of sequence operator in* $\mathbb{P}_{STeC}$.

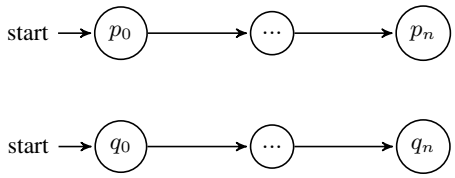

Fig. 7. The TA of P and Q

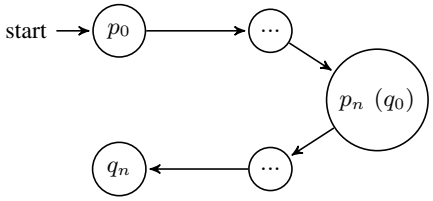

Fig. 8. The TA of $P; Q$

vii) For all $R \in \mathbb{P}_{STeC}$*, if* $R = P \ [] \ Q$*, then*

$$\mathcal{F}(R) = \mathcal{F}(P \ [] \ Q) = \mathcal{F}(Q) \oplus_{STeC} \mathcal{F}(P)$$
$$= A_Q \oplus_{STeC} A_P$$

if and only if $\mathcal{F}(P) = A_P$ *and* $\mathcal{F}(Q) = A_Q$ *hold. In other words,* $\mathcal{F}$ *preserve choise operator in* $\mathbb{P}_{STeC}$.

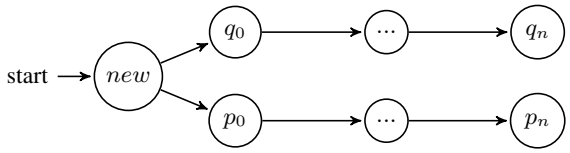

Fig. 9. The TA of $P[]Q$

viii) For all $R \in \mathbb{P}_{STeC}$*, if* $R = P \ \| \ Q$*, then*

$$\mathcal{F}(R) = \mathcal{F}(P \ \| \ Q) = \mathcal{F}(Q) \otimes_{STeC} \mathcal{F}(P)$$
$$= A_Q \otimes_{STeC} A_P$$

if and only if $\mathcal{F}(P) = A_P$ *and* $\mathcal{F}(Q) = A_Q$ *hold.* $\mathcal{F}$ *preserve parallel operator in* $\mathbb{P}_{STeC}$.

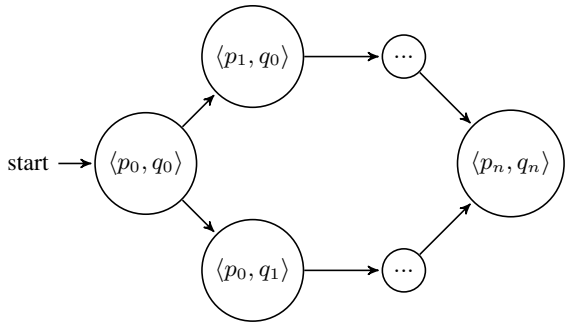

Fig. 10. The TA of $P \parallel Q$

ix) Set $\mathcal{F}(P \trianglerighteq_\delta Q) = (Q, \Sigma, \mathcal{C}, q_0, E, I, AP, L, F) = \mathcal{F}(P) \curvearrowright_\delta \mathcal{F}(Q)$ *where* $\curvearrowright_\delta$ *is a new binary operator between TAs. Set* $\mathcal{F}(P) = (Q_P, \Sigma_P, \mathcal{C}_P, s_P, E_P, I_P, AP_P, L_P, F_P)$ *and* $\mathcal{F}(Q) = (Q_Q, \Sigma_Q, \mathcal{C}_Q, s_Q, E_Q, I_Q, AP_Q, L_Q, F_Q)$. *We define* $\curvearrowright_\delta$ *as follows:*

- $Q = Q_P \cup Q_Q$, $\Sigma = \Sigma_P \cup \Sigma_Q$, $\mathcal{C} = \{x, y\}$, $q_0 = s_P$ *where* s_P *is the initial state of* $\mathcal{F}(P)$, $AP = AP_P \cup AP_Q$, $L = L_P \cup L_Q$, $F = F_Q$.
- $I = I_P \cup I_Q \cup \{\langle q, x \leq \delta\rangle \mid q \in Q_P\}$
- $E = E_P \cup E_Q \cup \{\langle q, a, g, Y, Q\rangle \mid q \in Q_P, a = \emptyset, g = x \geq \delta, Y = \{y\}, Q = \{s_Q\}$ *where* s_Q *is the initial state of* $\mathcal{F}(Q)$.

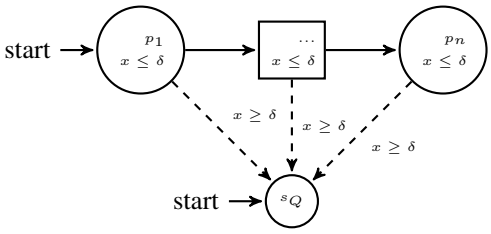

Fig. 11. The TA of $P \trianglerighteq_\delta Q$

Fig. 11 shows the TA of $P \trianglerighteq_\delta Q$, *where* s_Q *is the initial state of* $\mathcal{F}(Q)$, *and* p_1 *to* p_n *are arbitrary states of* $\mathcal{F}(P)$.

x) Set $\mathcal{F}(P \trianglerighteq ([]_{i\in I} A_i \to P_i)) = (Q, \Sigma, \mathcal{C}, q_0, E, I, AP, L, F)$. *If* $\mathcal{F}(P) = (Q_P, \Sigma_P, \mathcal{C}_P, s_P, E_P, I_P, AP_P, L_P, F_P)$ *and* $\mathcal{F}(P_i) = (Q_i, \Sigma_i, \mathcal{C}_i, s_i, E_i, I_i, AP_i, L_i, F_i)$ *,then we have the definition:*

- $Q = Q_P \cup (\cup_{i\in I} Q_i)$, $\Sigma = \Sigma_P \cup (\cup_{i\in I} \Sigma_i)$, $\mathcal{C} = \{x, y\}$, $q_0 = s_P$ *where* s_P *is the initial state of* $\mathcal{F}(P)$, $AP = AP_P \cup \cup_{i\in I}(AP_i \cup \{m_i*\})$, $L = L_P \cup (\cup_{i\in I} L_i)$, $F = \cup_{i\in I} F_i$.
- $I = I_P \cup (\cup_{i\in I} I_i)$
- $E = E_P \cup (\cup_{i\in I} E_i) \cup \{\langle q, m_i*, x == t_{A_i} + \tau(A_i), \{y\}, Q\rangle \mid q \in Q_P, l_{A_i} \in L(q), Q = \{s_i\}\}$ *where* m_i* *is either* $m_i!$ *or* $m_i?$ *depending on the type of action* A_i. l_{A_i} *and* t_{A_i} *are the start time and location of action* A_i.

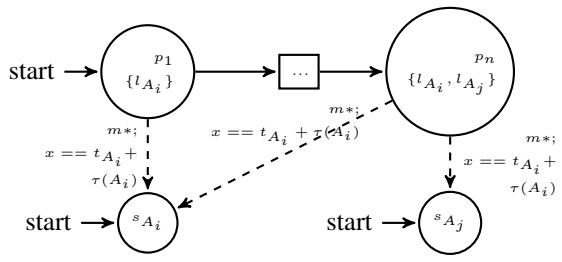

Fig. 12. The TA of $P \trianglerighteq ([]_{i\in I} A_i \to P_i)$

Fig. 12 shows the combination procedure of $P \trianglerighteq ([]_{i\in I} A_i \to P_i)$.

Theorem V.1. *The function* $\mathcal{F} : \mathbb{P}_{STeC} \to \mathbb{M}_{TA}$ *descriped in Definition IV.6 is well defined, and it satisfies*

$$dom(\mathcal{F}) = \mathbb{P}_{STeC}$$

Proof: It is obvious according to the definition of $\mathcal{F}$. ■

VI. An Example of ITS—Railroad Crossing System

Modeling and verifying Intelligent Transportation Systems using the proposed new verification framework is the main goal of our research. We want to design neater, nicer and more importantly easier-understandable modeling language for engineers to use at design-level of the development of Intelligent Transportation Systems without losing any expressive power when comparing the old modeling languages like timed CSP, and transform it into 'complex' languages such as TA in order to do verification. On the other hand, automatic tools should be implemented when people come up with a new formal language, only then it possibly becomes a powerful tool widely used in industry. In this section, we consider a classical example called 'railroad crossing system' shown in Fig. 13. In the next few subsections we use this example to show how to use STeC and CCSL as our new verification techniques to model behaviours and specify properties of intelligent transportation systems.

We describe the scenario of the system as follows: there are two agents: a train and a gate. The railroad lies in the east-west direction and the road lies in the south-north direction. When no train pass by, gate is open so the vehicles are allowed pass the cross. When a train comes, the train communicate with the gate to avoid collision on the crossing: it sends the gate a message 'Approach' to inform that it is approaching. The gate receive the message and make a judgement according to the traffic condition on the road, and then sends the message back to the train which in turn makes a judgement on what is going to do next according to this. If the gate is allowed to close, then the gate sends a message 'Close' back to the train, then the train can pass the gate, or the gate sends 'NonClose' back to the gate, then the gate have to stop and wait the gate open. After the train passes, the train sends a message 'Leav' to the gate, the latter should be opened again so that vehicles can pass again. It is a real-time system in which the time and location involves. In Fig. 13, we can see that 'Lapp','Lpass','Lstop' and 'Lleav' are locations at which train do certain action at certain time. We call such a constraint special-temporal specification and we use STeC as the 'right' language to describe such type of specifications (As a comparison, in [20] a similar 'train' system is described using CSP).

A. Modeling ITS Using STeC

We now use STeC to describe such an special-temporal system shown above. We firstly describe train agent and gate agent apart, and then combine them using parallel operator '||' in STeC shown before.

1) Train Agent: The system model descripted by STeC is as follows. Let P_{Train} and P_{Gate} be the process of train agent and gate agent, A_{Train} and A_{Gate} be the corresponding TA. Then we have:

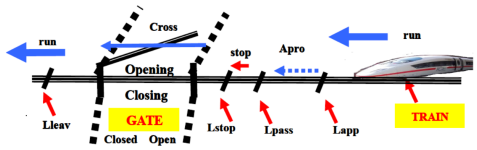

Fig. 13. Railroad Crossing System

$$P_{Train} = Run(\infty); (Send_{(Lapp,t,G)}(Appr) \parallel Approach_{(Laap,t)}(\Gamma));$$

$$\left\{ \begin{array}{l} Get_{(Lpass,t+\Gamma,G)}(Cross) \rightarrow (Pass_{(Lpass,t+\Gamma)}(\triangle); \\ \quad (Send_{(Lleav,t+\Gamma+\triangle,G)}(Lleav); P_{Train})) \\ [] \\ Get_{(Lpass,t+\Gamma,G)}(Noncross) \rightarrow (Stop_{(Lpass,t+\Gamma)}(\Upsilon); \\ Wait_{(Lstop,t+\Gamma+\Upsilon,G)}(Cross); Pass_{(Lstop,t\prime+\Omega,G)}(\Omega); \\ \quad (Send_{(Lleav,t\prime+\Omega,G)}(Lleav); P_{Train})) \end{array} \right\}$$

Easy to see that P_{Train} is a recursive process. The train is firstly approaching the crossing road, it send a message to the gate, informing that there is a train coming. After Γ time it tries to get a message from the gate. If it gets the "Cross" message, then the gate is now closed and it is safe for train to pass the crossing road. After the train passed, it send a "Lleav" message to inform the gate and continuely run again. If it gets the "NonCross" message, then the train will stop and wait for an inform by gate to pass again.

Applying Def.V.5, we can use the function $\mathcal{F}(P_{Train})$ to transform the model P_{Train} into a TA inductively:

$$\mathcal{F}(P_{Train}) = \mathcal{F}(Run(\infty); P'_{Train})$$
$$= \mathcal{F}(Run(\infty)) \diamond_{STeC} \mathcal{F}(P'_{Train})$$
$$= A_{Run(\infty)} \diamond_{STeC} \mathcal{F}((Send_{(Lapp,t,G)}(Appr) \parallel Approach_{(Laap,t)}(\Gamma));$$

$$\left\{ \begin{array}{l} Get_{(Lpass,t+\Gamma,G)}(Cross) \rightarrow (Pass_{(Lpass,t+\Gamma)}(\triangle); \\ \quad (Send_{(Lleav,t+\Gamma+\triangle,G)}(Lleav); P_{Train})) \\ [] \\ Get_{(Lpass,t+\Gamma,G)}(Noncross) \rightarrow (Stop_{(Lpass,t+\Gamma)}(\Upsilon); \\ Wait_{(Lstop,t+\Gamma+\Upsilon,G)}(Cross); Pass_{(Lstop,t\prime+\Omega,G)}(\Omega); \\ \quad (Send_{(Lleav,t\prime+\Omega,G)}(Lleav); P_{Train})) \end{array} \right\}$$

$$)$$

$$= A_{Run(\infty)} \diamond_{STeC} \mathcal{F}(P_1; P_2) = A_{Run(\infty)} \diamond_{STeC} \mathcal{F}(P_1; P_2)$$
$$= A_{Run(\infty)} \diamond_{STeC} \mathcal{F}(P_1) \diamond_{STeC} \mathcal{F}(P_2) = ... = A_{Train}.$$

where $P_1 = Send_{(Lapp,t,G)}(Appr) \parallel Approach_{(Laap,t)}(\Gamma)$ and

$$P_2 = \left\{ \begin{array}{l} Get_{(Lpass,t+\Gamma,G)}(Cross) \rightarrow (Pass_{(Lpass,t+\Gamma)}(\triangle); \\ \quad (Send_{(Lleav,t+\Gamma+\triangle,G)}(Lleav); P_{Train})) \\ [] \\ Get_{(Lpass,t+\Gamma,G)}(Noncross) \rightarrow (Stop_{(Lpass,t+\Gamma)}(\Upsilon); \\ Wait_{(Lstop,t+\Gamma+\Upsilon,G)}(Cross); Pass_{(Lstop,t\prime+\Omega,G)}(\Omega); \\ \quad (Send_{(Lleav,t\prime+\Omega,G)}(Lleav); P_{Train})) \end{array} \right.$$

Finally we can get the A_{Train} as the TA of the process P_{Train}. The result of A_{Train} is shown in the following figure(Fig. 14).

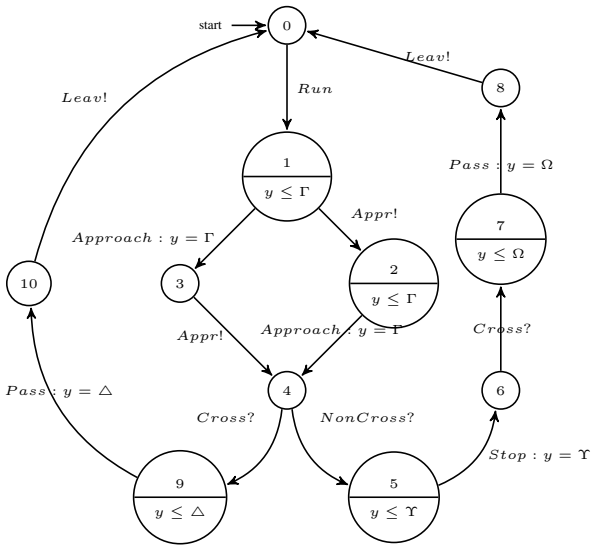

Fig. 14. The TA of Train Agent

Set $A_{Train} = (Q, \Sigma, \mathcal{C}, q_0, E, I, AP, L, F)$ be the corresponding timed-automata translated from STeC, the interpretations of how TA can mimic the behaviors of STeC is listed as follows:

- $Q = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$ is the set of locations of TA.
- $\Sigma = \{Run, Appr!, Cross?, Approach, Pass, Noncross?, Stop, Cross?\}$ corresponds to all the actions in STeC.
- $\mathcal{C} = \{x, y\}$. Two clocks, one for recording total time from the beginning and one for recording the delay of actions in STeC.
- $E \subseteq Q \times \Sigma \times G(\mathcal{C}) \times 2^{\mathcal{C}} \times 2^{Q}$ is a transition relation (as shown above). In timed-automata, the delay of time only happens in a state, action is executed instantaneously when the guard is satisfied. For example, transition which transits from state 1 to state 3 can be denoted as the tuple—$\langle 1, Approach, \{y = \Gamma\}, \{y\}, \{3\}\rangle$. Remember that $G(\mathcal{C})$ is the set of constraints under the set of clocks $\mathcal{C}$, a clock constraint can be defined as a BNF form:

  $$g := x \bowtie c \mid g \wedge g$$

  where $g \in \mathcal{C}$, $c \in \mathbb{N}$ and $\bowtie \in \{<, \leq, >, \geq\}$. $2^{\mathcal{C}}$ represents the clocks that should be set to 0 after transition is fired (here,in this example, clock y should be reset).
- $q_0 = 0$ is the initial state.
- $I : Q \rightarrow G(\mathcal{C})$ is an invariant-assignment function. It maps each state(or location) to a subset of $G(\mathcal{C})$. It specifies how long the timed-automata can stay in one state. For example, in our A_{Train}, we can only stay at most 5 units of time and after that, any transition

which satisfies the guard condition must happen, or (if no such transitions exists) no further progress is possible. In some states, one transition must happen when the state no longer satisfy its invariant, for example, state 9 and state 10. Since the invariant is $y \leq \triangle$, so the action $Pass$ is executed immediately when the clock y equals $\triangle$.

- $L : Q \rightarrow 2^{AP}$ is a labelling function which relates a set $L(s) \in 2^{AP}$ of atomic propositions to any state s. In this example, the set of atomic propositions represents the set of locations in STeC, we have $L(10) = \{Lleav\}$, $L(1) = \{Lapp\}$,$L(4) = \{Lpass\}$, $L(0) = \emptyset$, etc.

2) Gate Agent: Following the same procedure, we can get the A_{Gate} from P_{Gate}.

$$P_{Gate} = (Get_{(Open,t,G)}(Appr) \rightarrow Closing_{(Open,t+\theta_0)}(\Pi));$$

$$\left\{ \begin{array}{l} (Closed_{(Closed,t+\theta_0+\Pi)}(1); \\ Send_{(Closed,t+\theta_1,G)}(Cross); (Get_{(Closed,t+\theta_2)}(Leav) \\ \rightarrow Opening_{(Closed,t+\theta_2)}(\zeta))); P_{Gate} \\ [] \\ (Unclosed_{(Unclosed,t+\theta_0+\Pi)}(0); \\ (Send_{(Unclosed,t+\theta_0+\Pi,G)}(NonCross) \parallel \\ Clossing_{(Unclosed,t+\theta_0+\Pi)}(\pi)) \\ ; (Closed_{(Closed,t+\theta 3)}(0) \parallel (Send_{(Closed,t+\theta_3,G)}(\\ Cross)); (Get_{(Closed,t'+\Omega,G)}(Leav) \\ \rightarrow Opening_{(Closed,t'+\theta_4)}(\zeta)); P_{Gate} \end{array} \right\}$$

For the agent Gate, it begins with the location "Open" and wait for the message "Appr" sent by train. Then the gate tries to close the gate. If the gate is successfully closed, then it send "Cross" message to train. Then keep listening the train's message until message "Leav" is received. Open the gate and then again keep listenning. If the gate is not closed. Send "NonCross" message to train to stop the train and try to close again. Send the "Cross" message to train if it is successfully closed and reopen it after receiving "Leav".

We can get $\mathcal{F}(P_{Gate}) = A_{Gate}$ which is shown in Fig. 15 :

3) Railroad Crossing System: The railroad crossing system can be presented as

$$P_{Sys} = P_{Train} \parallel P_{Gate}$$

according to the STeC. So the corresponding TA is:

$$A_{Sys} = \mathcal{F}(P_{Sys}) = \mathcal{F}(P_{Train} \parallel P_{Gate}) = A_{Train} \otimes_{STeC} A_{Gate}$$

.

B. Modeling System in Practice Using STeC

We have developed a program which takes STeC text as input and convert it into Uppaal-Timed Automata. The latter can be easily read and analyzed by Uppaal—a popular verification tool developed by a team at Uppsala University(visit www.uppaal.org for more information). In this section, we just give an example showing such a tool because introducing such an automatic tool in detail goes beyond the theme of this paper. We omit the details of architecture (Notice that the algorithm used for the implementation of such a tool is slightly different

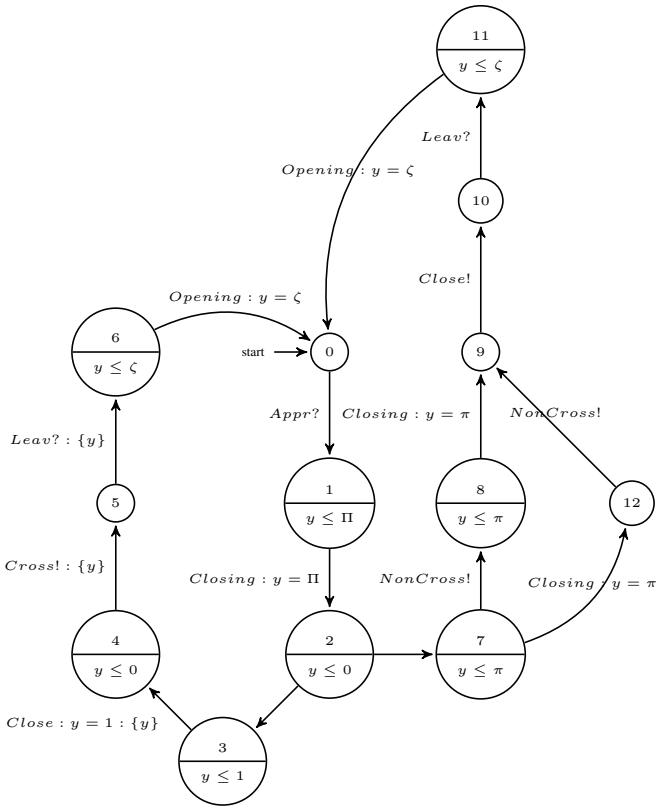

Fig. 15. the TA of Gate Agent

from the definition of transformation in this paper (Def.V.5). So the result of TA in Fig. 17 and Fig. 18 are a bit different from the result in theory in Fig. 14 and Fig. 15.

In the following we show the STeC text for the 'Railroad Crossing System'. Here is the description of the train agent using STeC .

```
//Variable Declaration
t=0.       //time for starting

//Train Agent
t_appr=5.       //time spent on approach
t_pass=10.      //time spent on pass
t_stop=8.       //time spent on stop

//two macros P1 and P2
#define P1 \
  (Get(Lpass)(Cross)->Pass(Lpass)($t_pass)-> \
  Send(Lleav)(Leave)->#T)

#define P2 \
  (Get(Lpass)(NonCross)->Stop(Lpass)($t_stop)-> \
  Get(Lstop)(Cross)->Pass(Lstop)($t_pass)-> \
  Send(Lleav)(Leave)->#T)

//Train Process
Train=@T(
  Run()();Send(Lapp,$t)(Appr)->Approach(Lapp)($t_appr)
  -> (P1 [] P2)
).
```

In the block of STeC code shown above, we firstly define a starting time $t = 0$, then define the variables 't_appr', 't_pass'

and 't_stop' which are 'the time spent on approach', ' the time spent on pass' and 'the time spent on stop' respectively. The symbol '$' is a variable-prefix. Two macro 'P1' and 'P2' are subprocesses in the 'Train process' which is defined next. In 'Train process', we define a process variable named 'Train', and describe its behaviour as the STeC formulas that we saw in the previous section. '@T' is a recursion symbol.

Here is the description of Gate agent.

```
//Gate Agent
t_close=4. //time spent on close
t_open=4. //time spent on open

//Two macros P3 and P4
#define P3 ( \
        Closed(Closed)(1)->Send(Closed)(Cross)->\
        Get(Closed)(Leave) \
        ->Opening(Closed)($t_open);#G \
)

#define P4 ( \
        Unclosed(Unclosed)(0)->( \
                Send(Unclosed)(NonCross)|| \
        Closing(Unclosed)($t_close) \
                )-> \
        Send(Closed)(Cross)-> \
        Get(Closed)(Leave)-> \
        Opening(Closed)($t_open);#G \
)

//Gate Process
Gate=@G(
        Get(Open,$t)(Appr)-> \
        Closing(Open)($t_close);
        (P3 [] P4)
).
```

As the result of STeC tool here we just give the corresponding transformed Uppaal TA shown in Fig. 17 and Fig. 18. Another version of result of TA for Gate agent which is drawn with 'graphviz' tool is shown in Fig. 16 ('graphviz' is a famous open source visualization tool, visit http://www.graphviz.org/ for more information), where the red node(that is 'Get_S106' is the initial state and the green node 'F_211' is the final state.

C. Specifying Properties Using CCSL

As introduced before, we define CCSL as the specification language for STeC. Def. IV.10 shows the details. We show using CCSL describing 'logic' specifications of ITS, using the 'Railroad Crossing System' as an example.

In the Railroad Crossing System introduced above, people usually care that the model is safe or live. For example, the next property is a simple safe property:

Property 1: When the train passes, the gate must be closed.

Intuitively it means that we never allow a train passes a crossing when the gate is still open in reality. Such a property can be described by LTL or CTL and also PSL. But I want to stress here that we choose CCSL not only because its nicer and neater way of describing logical relationship between events,

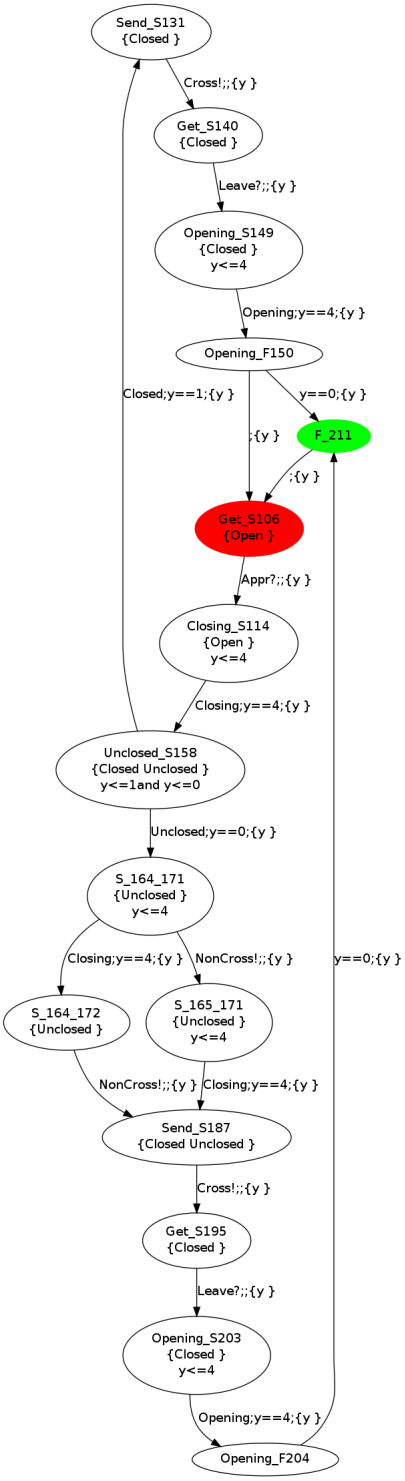

Fig. 16. The TA of Gate Drawn Using graphviz tool

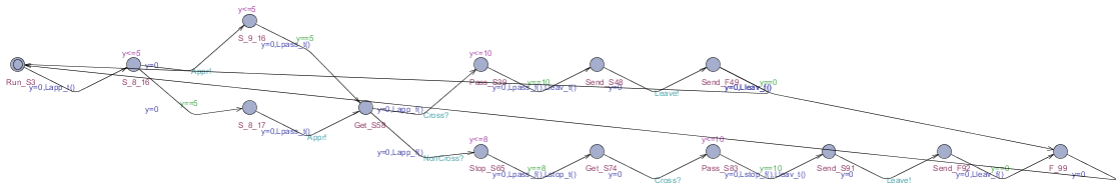

Fig. 17. The Uppaal TA of Train

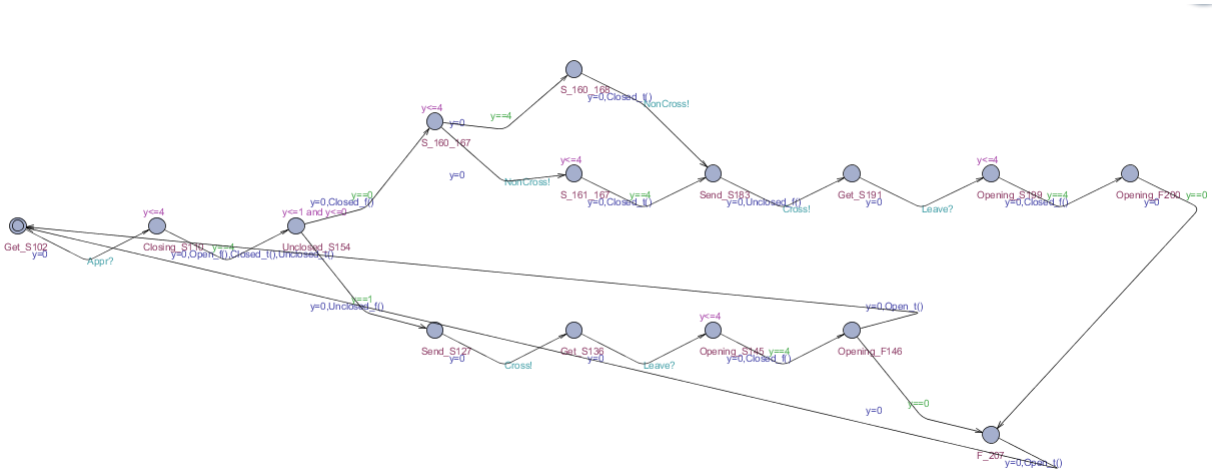

Fig. 18. The Uppaal TA of Gate

but also it is not completely equivalent to the PSL. For more about that, see [6].

To specify this property in CCSL, we have:

$$\mathcal{C}_{close} \prec \mathcal{C}_{mayPass} \prec \mathcal{C}_{open};$$
$$\mathcal{C}_{close}\ Alternate\ \mathcal{C}_{open};$$
$$\mathcal{C}_{pass} \subset \mathcal{C}_{mayPass};$$

'$\prec$' means 'precedence' in CCSL, the first formula says the clock 'close' always ticks before the clock 'mayPass' and the clock 'mayPass' always ticks before the clock 'open'. The second formula says that the clock 'close' and the clock 'open' tick alternately. The third formula says that clock 'pass' is a subclock of the clock 'mayPass', which means that whenever the clock 'pass' ticks, the clock 'mayPass' must tick. Consulting the definition of these operators in CCSL(see the previous section of the introduction of CCSL or for more details, refer to [10]), it is not hard to see that this CCSL constraint is exactly the Property 1 mentioned above.

A more intuitive way of looking at this property is shown in Fig. 19 where a tool called 'TimedSquare' which is devoloped by AOSTE team(see [**?**]) is used for simulating the behaviour of CCSL constraint. It gives a 'possible' result of events which satisfies the CCSL constraint defined on it.

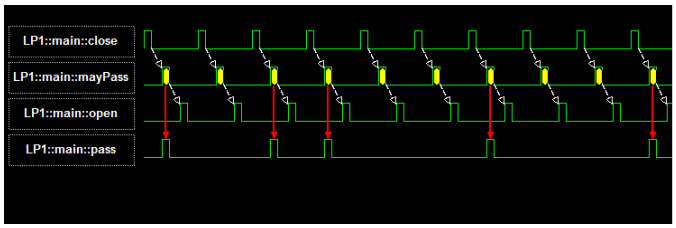

Fig. 19. Simulate result of Property 1

Considering another property:

> **Property 2**: Once the train approaches, the gate will close in at most 1 minutes.

Unlike the former one, it contains a time factor '1 minutes'. Similarly, the corresponding CCSL constraint is as follows:

$$\mathcal{C}_{appr} \prec \mathcal{C}_{close} \prec \mathcal{C}_{appr\_delay};$$
$$\mathcal{C}_{appr\_delay} = \mathcal{C}_{appr} \ \$ \ 1;$$

Similarly, we can easily understand the first formula in CCSL. For the second formula, symbol '$' means 'delay' in CCSL. It says that the clock 'appr_delay' ticks 1 units of time delay for the clock 'appr'. Fig. 20 shows the relationship between the three clocks.

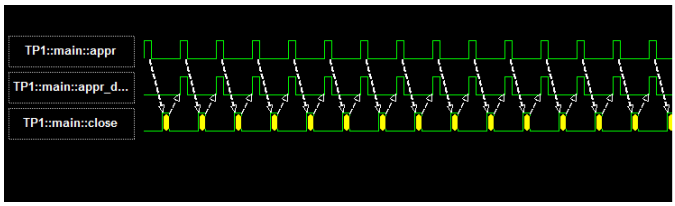

Fig. 20. Simulate result of Property 2

VII. Conclusion

In this paper, we introduce a new verification framework where we use STeC as a modeling language and CCSL as a specification language. And we mainly focus on the import of CCSL into STeC and the transformation from STeC into TA. We also analysis the rationality of such a transformation by proofing a bisimilarity between STeC and its transformed TA. At last we use a simple ITS system to illustrate how to use STeC ans CCSL in practice, relative automatic tools have been developed for both STeC and CCSL for such a purpose.

As for the future work, in this paper, we give the encoding from STeC to TA, but the equivalence between them has not yet been proofed. And we also will focus the implementation of the proposed new verification platform using STeC and CCSL. By now we have showed that we can transform both STeC and CCSL into Timed Automata and have such an automatic tool for STeC. But for CCSL, such a tool needs to be built.

Acknowledgment

This work is supported by the INRIA Associated Team DAESD between INRIA and ECNU, NSFC (No. 61021004 and No. 61370100) and Shanghai Knowledge Service Platform Project (No. ZF1213).

References

[1] C.A.R Hoare. *Communicating Sequential Processes*. Communications of ACM, Volume 21 No. 8, 1978.

[2] Joel Ouaknine,Steve Schneider. *Timed CSP:A Retrospective*. Electronic Notes in Theoretical Computer Science, 162 (2006) 273C276.

[3] Robin Milner. *A Calculus of Communicating Systems* . Computer Science, Vol.92, 1986.

[4] Steve Schneider . *An Operational Semantics for Timed CSP* . Information and Computation, 116:193-213, 1995.

[5] R. Alur and D. L. Dill . *A theory of timed automata* . Theoretical Computer Science, 126(2):183-235, 1994.

[6] Regis Gascon, Frederic Mallet, Julien DeAntoni. *Logical time and temporal logics: comparing UML MARTE/CCSL and PSL*. Temporal Representation and Reasoning (TIME), pp 141-148, 2011.

[7] Charles Andre, Frederic Mallet. *Clock Constraints in UML/MARTE CCSL*. Research Report of Research Unit of INRIA Sophia Antipolis, 2008.

[8] Charles Andre. *Syntax and Semantics of the Clock Constraint Specification Language(CCSL)*. Research Report of Research Unit of INRIA Sophia Antipolis, 2009.

[9] OMG. *UML Profile for MARTE, v1.0*. Object Management Group, 2009.

[10] Frederic Mallet, Jean-Vivien Millo, Yuliia Romenska. *State-based representation of CCSL operators*. Research Report of Project-Team Aoste of INRIA Sophia Antipolis, 2013.

[11] Jagadish Suryadevara, Cristina Seceleanu, Frederic Mallet and Paul Pettersson. *Verifying MARTE/CCSL Mode Behaviors using UPPAAL*. 11th International Conference on Software Engineering and Formal Methods, Volume 8137, pp 1-15, 2013.

[12] Iryna Zaretska, Galyna Zholtkevych, Grygoriy Zholtkevych. *Clocks Model for Specification and Analysis of Timing in Real-Time Embedded Systems*.

[13] Yixiang Chen. *STeC: A Location-Triggered Specification Language For Real-Time Systems*. Science China, Vol.53, No.1-18, 2010.

[14] Hengyang Wu, Yixiang Chen and Min Zhang *On Denotational Semantics of Spatial-Temporal Consistency Language STeC*. International Symposium on Theoretical Aspects of Software Engineering, pages 113-120, 2013.

[15] Stefano Cattani and Marta Kwiatkowska. *A Refinement-based Process Algebra for Timed Automata*. Under consideration for publication in Formal Aspects of Computing, 17(2), pages 138-159, Springer. August 2005.

[16] Parosh Aziz Abdulla, Pavek Krcal and Yi Wang. *Sampled semantics of times automata*. Logical Methods in Computer Science, Vol. 6 (3:14): 1-37, 2010.

[17] Frederic Mallet. *Logical Time in Model-Driven Engineering*. University of Sophia Antipolis, 2010.

[18] Volker Diekert, Paul Gastin. *First-order definable languages*. Logic and Automata: History and Perspectives, Texts in Logic and Games, 2008.

[19] Rajeev Alur, David Dill. *Automata-theoretic Verification of Real-time Systems*. 1995.

[20] A.W.Roscoe. *A CSP solution to the "trains" problem*. Programming Research Group, University of Oxford.