A case study: Switch Control System for illustrating the GongZhuFen Accident

Xiaohong Chen[1], Ling Yin[1], Yijun Yu[2], and Zhi Jin[3]

[1] East China Normal University, Shanghai, 20062, CHINA
[2] The Open University, United Kingdom
[3] Peking University, Beijing, 100871, CHINA

1 Problem description

The class A accident happened to an on-board switch at Beijing railway station around the GongZhuFen (i.e., Tomb of Princess) stop on 18 May 2013 at 15:46. It caused 4 trains to delay by at least 5 minutes, 1 train to return to the original stop, 1 temporally train added into the schedule and changes of 13 scheduling tables[4]. The reason is that the switch was wrongly represented, which was not found out in time. The involved subsystem is the switch control system (SC). Here we simplify for illustration by describing basic problem domains.

SC monitors the states and actions of train and crossings to decide whether or not to let the train pass and control equipments to perform corresponding reactions. While a train is near the crossing, the on-board system of the train sends a Request to SC, asking for entry. SC then checks about the state (either Occupied or Unoccupied) of current Track, in order to respond to the Train. If the state is Unoccupied, SC shall accept; otherwise it shall reject the request. If SC does accept the request, it will notify the Light to turn into Green, and the Switch to turn into Forward position. If SC does reject the request, the train must wait. Only when the train sees the Green light and the Forward switch, it can enter the crossing. And only after the train has left, it will send a message to inform the SC. Then SC will notify the Light to turn into Red and the Switch to position Reverse.

Of all the input domains, the problem diagram of the SC system is shown in Figure 1. It captures the problem domains and their interactions. The problem domains include *On-board System (OS)*, *Track (TK)*, *Light (LT)*, and *Switch (SH)*. There are 17 interactions. The detailed interactions are asserted as follows.

$int_1 =< OS, SC, Requst >$	$int_2 =< SC, OS, Accept >$
$int_3 =< SC, OS, Reject >$	$int_4 =< OS, SC, Wait >$
$int_5 =< OS, SC, Enter >$	$int_6 =< OS, SC, Leave >$
$int_7 =< SC, TK, InqueryState >$	$int_8 =< TK, SC, Occupied >$
$int_9 =< TK, SC, Unoccupied >$	$int_{10} =< SC, LT, GreenPulse >$
$int_{11} =< SC, LT, RedPulse >$	$int_{12} =< LT, SC, Green >$
$int_{13} =< LT, SC, Red >$	$int_{14} =< SC, SH, ReversePulse >$
$int_{15} =< SC, SH, ForwardPulse >$	$int_{16} =< SH, SC, Reverse >$
$int_{17} =< SC, SH, Forward >$	

[4] www.ditiezu.com/thread-317756-1-1.html

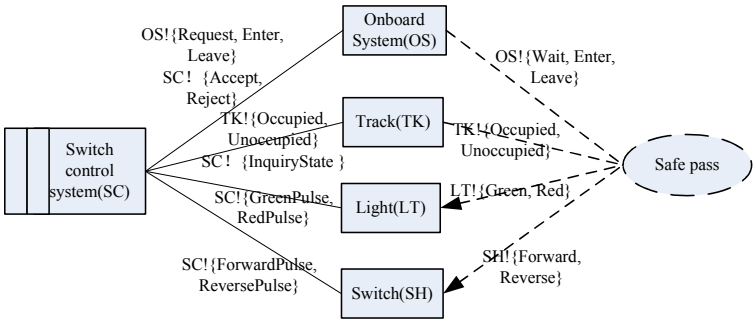


Fig. 1. Problem diagram of the SC

Figure 2 shows the state diagrams of four of the problem domains, *On-board system*, *Track*, *Switch*, and *Light*.

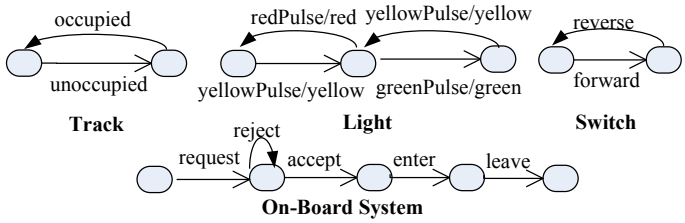


Fig. 2. State diagrams of physical entities in the SC systems

1.1 Tool support

We developed TimePF by extending a tool DPTool [**?**] which is an editor of the PF problem diagram. TimePF allows the user to graphically edit the interaction relations, the qualitative relations and quantitative relations. Moreover, the tool allows the user to follow the specification process, and aids the user with some automated timing requirements processing. Finally, it performs automated consistency checking for the resulting timing specification.

1.2 Specification

Step 1: Define entity and interaction clocks For each problem domain in the problem diagram (see Figure 1), we define an entity clock for it. For example, for the domain *Light*, we define a clock C_{LT}: $C_{LT} =< I_{LT}, \prec_{LT}, `ms' >$.

For each interaction in the problem diagram, we construct an interaction clock for it. For example, interaction *Green*. Its corresponding clock is defined

as C_{Gn}: $C_{Gn} =< I_{Gn}, \prec_{Gn}, `ms' >$, where $I_{Gn} = \{C_{Gn}[1], C_{Gn}[2], \ldots, C_{Gn}[n]\}$. In order to simply the clock name, we pick the first and last characters of the one word interaction and first two characters for two word interactions to represent the whole interaction.

Step 2: Define entity clock specifications for physical entities For each problem domain in the problem diagram, we define a clock specification for it. Take domain "Light" as an example.

(1) Find entity and interaction clocks For domain "Light", its entity clock is C_{LT}. Then $LT.CS.\ EC{=}C_{LT}$. The domain LT has four interactions: int_{10} (GreenPulse, GP), int_{11} (RedPulse, RP), int_{12} (Green) and int_{13} (Red). Their corresponding clocks are: C_{GP},C_{Rd} , C_{Gn} and C_{RP}. Then we have: $LT.CS.ICs{=}\{C_{GP}, C_{RP}, C_{Gn}, C_{Rd}\}$.

(2) Identify clock relations between the entity clocks and interaction clocks The entity clock C_{Light} is constructed by union of its interaction clocks. That means: $C_{LT} = C_{GP}$ *union* C_{RP} *union* C_{Gn} *union* C_{Rd}

(3) Reason about the clock relations among interaction clocks using physical entity property From the state diagram of Light in figure 2 and state diagram transformation patterns, we know that int_{GP} and int_{Green} has *alternate* relation. As to the quantitative constraints, suppose the response time of turning red or green is less than 3 ms.

Similarly, we get the other entity clock specifications as shown in Table 1. The constraints are listed in Table 2.

Table 1. Entity clock specification in our case

Domain	**EC**	**ICs**	**CCs**
LT	C_{LT}	$\{C_{GP}, C_{RP}, C_{Gn}, C_{Rd}\}$	$\{ct1, ct2, ct3, ct4, ct5, ct6\}$
SH	C_{SH}	$\{C_{FP}, C_{RP}, C_{Fd}, C_{Re}\}$	$\{ct7, ct8, ct9, ct10, ct11, ct12\}$
OS	C_{OS}	$\{C_{Rt}, C_{At}, C_{Rt}$, C_{Wt},C_{Er}, $C_{Le}\}$	{ct13, ct14, ct15, ct16, ct17, ct18, ct19, ct20}
TK	C_{TK}	$\{C_{IS}, C_{Od}, C_{Ud}\}$	$\{ct21, ct22, ct23, ct24\} >$
CK	C_{CK}	OS.CS.ICs $\cup$ TK.CS.ICs	OS.CS.CCs $\cup$ TK.CS.CCs $\cup$ {ct25, ct26, ct27, ct28}
$SCPD$	C_{SCPD}	CK.CS.ICs $\cup$ LT.CS.ICs $\cup$ SH.CS.ICs	CK.CS.CCs $\cup$ LT.CS.CCs $\cup$ SH.CS.CCs $\cup$ {ct29, ct30, ct31, ct32, ct33, ctry34, ct35, ct36, ct37, ct38}
cyb	C_{cyb}	SCPD.CS.ICs	$SC.CS.CCs \cup \{C_{cyb} = C_{SCPD}\}$

Step 3: Construct clock specification composition for problem domain collaboration We combine On-board Systems (OS) and Track (TK) because after the OS requests, the SC asks the state of the track. If the state is occupied, then SC rejects the request, and if the state is unoccupied, then SC accepts the request. In this way, we get constraints $ct25, ct26$, and $ct27$ listed in Table 2.

Table 2. Clock constraints list

$ct1 : C_{GP}$ *alternate* C_{Gn}	$ct2 : C_{RP}$ *exclude* C_{Rd}
$ct3 : C_{RP}$ *alternate* C_{Rd}	$ct4 : C_{RP}$ *exclude* C_{Rd}
$ct5 : C_{GP}$ *boundedDrift_0_3* C_{Gn}	$ct6 : C_{RP}$ *boundedDrift_0_3* C_{Rd}
$ct7 : C_{SH} = C_{RP}$ *union* C_{FP} *union* C_{Re} *union* C_{Fd}	$ct8 : C_{FP}$ *alternate* C_{Fd}
$ct9 : C_{RP}$ *alternate* C_{Re}	$ct10 : C_{FP}$ *exclude* C_{RP}
$ct11 : C_{FP}$ *boundedDrift_0_3* C_{Fd}	$ct12 : C_{RP}$ *boundedDrift_0_3* C_{Re}
$ct13 : C_{OS} = C_{Rt}$ *union* C_{At} *union* C_{Rt} *union* C_{Wt} *union* C_{Er} *union* C_{Le}	$ct14 : C_{Rt}$ *Alternate* C_{Re}
$ct15 : C_{Rt}$ *boundedDrift_0_50* C_{Re}	$ct16 : C_{Re} = C_{At}$ *union*C_{Rt}
$ct17 : C_{At}$ *exclude* C_{Rt}	$ct18 : C_{At}$ *alternate* C_{Er}
$ct19 : C_{Rejct}$ *alternate* C_{Wait}	$ct20 : C_{Er}$ *alternate* C_{Leave}
$ct21 : C_{TK} = C_{IS}$ *union* C_{Od} *union* C_{Ud}	$ct22 : C_{IS}$ *alternate* C_{RS}
$ct23 : C_{RS} = C_{Od}$ *union*C_{Ud}	$ct24 : C_{Od}$ *exclude* C_{Ud}
$ct25 : C_{Rt}$ *strictPre* C_{IS}	$ct26 : C_{Od}$ *strictPre* C_{Rt}
$ct27 : C_{Ud}$ *strictPre* C_{At}	$ct28 : C_{CK} = C_{OS}$ *union* C_{TK}
$ct29 : C_{At}$ *coincides* C_{GP}	$ct30 : C_{At}$ *coincides* C_{FP}
$ct31 : C_{GP}$ *union* C_{FP} *coincides* C_{Er}	$ct32 : C_{Le}$ *coincides* C_{RP}
$ct33 : C_{Le}$ *coincides* C_{RP}	$ct34 : C_{Rd}$ *boundedDift_0_5* C_{Gn}
$ct35 : C_{Re}$ *boundedDift_0_5* C_{Rt}	$ct36 : C_{Le}$ *boundedDift_0_5* C_{Re}
$ct37 : C_{FP}$ *boundedDift_0_3* C_{Fd}	$ct38 : C_{RP}$ *boundedDift_0_3* C_{Re}

Suppose the combine entity is called ComOSTK (CK), then the clock specification for the CK could be: $CK.CS = OS.CS||TK.CS$. We define a clock called C_{CK} for the entity clock: $C_{CK} =< I_{CK}, \prec_{CK}, `ms' >$ where C_{CK} is actually the union of all the interaction clocks of OS and TK. Thus we get ct28 (listed in Table 2). In the construction of CK.CS, the newly introduced constraints include ct25, ct26, ct27 and ct28.

Similarly, as to the combination of ComOSRK and Light and Switch, they collaborate together as a domain *SCPD* to fulfil their requirements. So the composed clock specification would be: $SCPD.CS = CK.CS||LT.CS||SH.CS$. The newly introduced constraints are ct29-ct38 listed in Table 2.

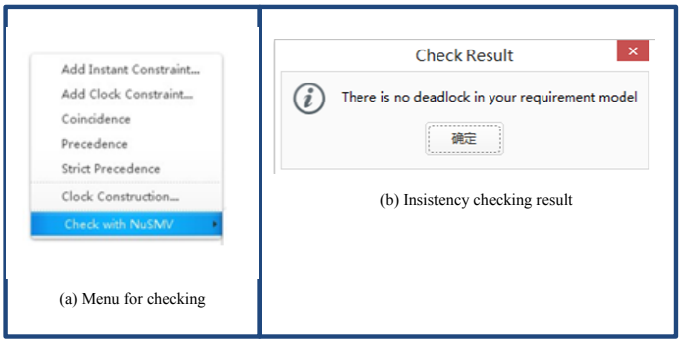
Step 4: derive timing specification for the cyber entity The clock for the cyber entity C_{cyb} is the composed problem domain clock C_{SCPD}. So we have $C_{cyb} = C_{SCPD}$ (ct39). The interaction clocks are actually the interaction clocks in $SCPD.CS$. Then the *cyb.CS* eaquals to $SCPD.CS$.

Finally, all the 39 clock constraints in *cyb.CS* are extracted to form a timing specification. We also noted that the constraint "ct36" was not given from the accident description: the device was not reset in time after the train is left.

1.3 Consistency verfication

Our tool TimePF checks the specification against the consistency properties. It can detect a counter example on the specification without constraint "ct36".

Figure 3 shows the main checking interfaces of TimePF. The left part is the menu for users to choose. The right part is the result of consistency checking for the switch control system. The verification only takes 1 second.



(a) Menu for checking

(b) Insistency checking result

Fig. 3. A checking snapshot of the Switch control system in TimePF