Eindhoven University of Technology

System Validation

2IMF30

Modelling and Validation of an Automatic Train Protection System

Author
Xuanying Cheng 1306871
Yingkai Huang 1321927
Zhengjie Huang 1308971
Chaolun Ma 1317903

March 23, 2019

1 Introduction

1.1 ATP System

Nowadays, the ERTMS (European Railway Traffic Management System) is becoming the standard train control system in Europe. However,currently each country still has its own train control mechanisms, causing numerous inter-state train traffic. In order to solve this problem, a new ATP system which may work concurrently and cooperate with the ERTMS is of vital importance.

First of all, the switch of control priority between these two systems needs to be discussed. As designed in our project, when the train is on the track protected by the ERTMS, the new ATP should be switched off. Reversely, if the train leaves the ERTMS protected tracks, the new ATP should take control of the train. Besides, on special occasions where the train enters some hardly used or remote lines, the ATP will also get switched off after detecting a special pulse. Here, we specify that when the ATP is active, it monitors the current speed and speed limit of the train and sends instruction message to ERTMS when necessary(current speed exceeds allowed speed, etc.). When the ATP is not active, it does not do anything other than waiting for either a leaveERTMS action or specific receivePulses actions to become active.

Secondly, the ATP will send the speed limits to the ERTMS. When the antenna detects electrical pulses related to speed limits, the ATP system will transmit the speed limit to the ERTMS to guide the driver to adjust the speed according to country-specific rules. Besides, the ATP will also instruct the ERTMS to ring the cabin bell if the ATP detects that the speed is higher than allowed. An exception is that when the train is a freight train, the ATP will ring the cabin bell 3 times to indicate a brake release before the train reaches its allowed speed. More specifically, the driver is allowed to release the brake when the speed of the train is decreased to its allowed speed plus 20.

When it comes to the situation that the driver does not react within 3 seconds after the bell ringing, or the driver releases the brake earlier than the train reaching the desired speed, the ATP will instruct the ERTMS to apply an emergency brake, after which it is possible to reset the ATP.

Moreover, the new ATP will prevent the train from passing the red light when receiving stop messages sent by the beacon near the track.

1.2 mCRL2 Program

In the program, we have defined the following sorts which are related to the analysis and modeling of the system as shown in Table 1. The new ATP system will receive 6 kinds of pulses as seen in Table 2. The sixty, eighty, onethirty, oneforty represent different frequencies of pulses. The special pulse indicates

Sort	Elements	Meaning
Pulses	sixty	The train receives a speed pulse of frequency 1
	eighty	The train receives a speed pulse of frequency 2
	onethirty	The train receives a speed pulse of frequency 3
	oneforty	The train receives a speed pulse of frequency 4
	special	The train receives a special pulse sending by the track
	beacon	The train receives a pulse sent by beacons
	none	The train receives no pulse
Country	NL	The train is in Netherlands
	BE	The train is in Belgium
Track	ATPArea	The train is on the track of the ATP area
	ERTMSArea	The train is on the track of the ERTMS area
BellState	bellOn	The bell is ringing
	bellOff	The bell is not ringing
BrakeStatus	braking	The train is braking normally
	emergencyBrake	The train is instructing an emergency brake
	releasing	The train is not braking
TrainType	freight	The train is a freight train
	other	The train is not a freight train
ATPStatus	active	The ATP system is active
	inactive	The ATP system is inactive
	pause	The ATP will come to a pause status after an emergency brake

Table 1: Sorts of the new ATP system

that the train has entered the remote or hardly used lines and the ATP system should be switched off. The beacon pulse means that the train should stop before the red signal. If the train receives no pulse, the ATP system will get a none indication. ATP will receive the country information from ERTMS, in this system the code will be NL and BE which represent the Netherlands and Belgium.

The ATP will get the information of Track, BrakeStatus, TrainType from ERTMS. Besides, the BellState and ATPStatus are determined by the ATP and will be sent to ERTMS to execute necessary operations.

Pulse	Belgium	Netherlands
sixty	50km/h	60km/h
eighty	70km/h	80km/h
onethirty	120km/h	130km/h
oneforty	130km/h	140km/h
none	30km/h	40km/h
special	The ATP system should be switched off	
beacon	The train should stop before the red signal	

Table 2: Meaning of Pulse

2 Global Requirements

In this section, the global requirements of the new ATP system in natural language.

1. The ATP will be active if the train is not in ERTMS protected areas and remote and hardly used lines. The ATP will be inactive if the train enters ERTMS protected areas and remote and hardly used lines.
2. The ATP will tell the speed requirement to the ERTMS when the antenna detects specific electrical pulse. When the antenna detects electrical pulses (or detects no pulse then the speed limit is 40 km/h) related to speed limit like 60km/h, 80km/h, 130km/h and 140km/h, the ATP system will transmit this speed limitation to the ERTMS to guide the driver to adjust the speed according to country-specific rules.
3. The ATP will instruct ERTMS to ring the cabin bell if the ATP detects that the speed of the train is higher than allowed and the ATP is active.
4. When the speed of a freight train decreases to the allowed speed plus 20 from higher, the ATP will ring the cabin bell 3 times to indicate that the driver is allowed to release the brake.
5. The ATP will instruct the ERTMS to apply an emergency brake when the drivers don't react within 3 seconds after the bell begins ringing, or the driver releases the brake earlier than the train reaches the desired speed if the ATP is active.
6. The ATP can be reset by drivers after an emergency brake is applied and the train is stopped.
7. The ATP will instruct the ERTMS to automatically stop the train when a stop beacon signal is read.

3 Interactions

In this section, the interactions of the system are listed as table.3.

4 Interaction Diagram

The new ATP system consists of 6 controllers, namely, the ATP, the Bell controller, the Brake controller, the General, the ATP++(ATPpp) and the Antenna. Every one of them works properly, between which process communication happens so that designed requirements put forward before are possible to get realized. Fig.1 shows the structure of the system.

ATPActive	Indication that the ATP is active.
ATPInactive	Indication that the ATP is inactive.
bell_ring	Instruction to ring the bell for three seconds
instru_emergency_brake	Instruction to apply the emergency brake
reset_ATP	The ATP receives a message to reset itself
leave_ERTMS	The train leaves ERTMS protected tracks
enter_ERTMS	The train enters ERTMS protected tracks
safe_speed	Indication that the train is at safe speed
speeding	Indication that the train is not at safe speed
receive_speed_limit	The ATP receives the speed limit from track
receive_current_speed	The ATP receives current speed from ERTMS
send_bell_state	The ATP sends the bell state to ERTMS
send_brake_status	The ATP sends the status of brake to ERTMS
send_ATP_status	The ATP sends the status of ATP to ERTMS
receive_train_type	The ATP receives the type of train from ERTMS
receive_speed_pulses	The ATP receives pulses from the track
receive_country	The ATP receives the country of track from ERTMS
receive_bell_expired	The ATP receives the signal indicating that the bell has rang for 3 seconds
bell_reminder	Instruction to ring the bell three times
send_allowed_speed_ERTMS	The ATP sends the speed limit to ERTMS

Table 3: Interactions of new ATP system

5 Requirement Translations

In this section, we translate the global requirements in terms of the interactions listed in section 3. We also translate them to a model formulas which would be used in section 7.

1. **a** The ATP will be active if the train is not in ERTMS protected areas and remote and hardly used lines. The ATP will be inactive if the train enters ERTMS protected areas and remote and hardly used lines.

   b If an *enter_ERTMS* takes place, an *ATPActive* event will follow. If a *leave_ERTMS* takes place, an *ATPInactive* event will follow.

   c $[true^{\star} \cdot enterERTMS]\mu X.([\overline{ATPInactive}]X \wedge \langle true\rangle true)$
   $[true^{\star} \cdot leaveERTMS]\mu X.([\overline{ATPActive}]X \wedge \langle true\rangle true)$

2. **a** The ATP will tell the speed requirement to the ERTMS when the antenna detects specific electrical pulse. When the antenna detects electrical pulses (or detects no pulse then the speed limit is 40 km/h) related to speed limit like 60km/h, 80km/h, 130km/h and 140km/h, the ATP system will transmit this speed limitation to the ERTMS to guide the driver to adjust the speed according to country-specific rules.

   b Before the action *send_allowed_speed_ERTMS*, the action *receive_speed_limit* and *ATPActive* should have happened without *ATPInactive* in between.

   c $\nu X(b_1 : \mathbb{B} := true, b_2 : Country = NL)$.

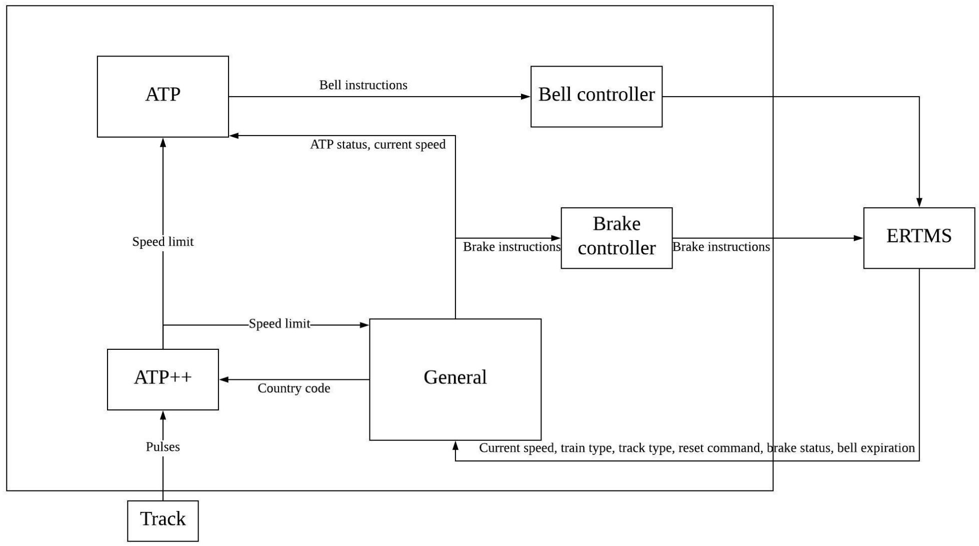

Figure 1: Interaction Diagram

$$
\begin{aligned}
&[ATPActive]X(true, b_2) \wedge \\
&[ATPInactive]X(false, b_2) \wedge \\
&[receive\_country(NL)]X(b_1, NL) \wedge \\
&[receive\_country(BE)]X(b_1, BE) \wedge \\
&[\overline{ATPActive \bigcup ATPInactive \bigcup receive\_country(NL) \bigcup receive\_country(BE)}]X(b_1, b_2) \\
&((b_1) \wedge receive\_speed\_pulses(sixty) \wedge (b_2 \approx NL) \rightarrow \\
&\mu X.[\overline{send\_allowed\_speed\_ERTMS(60)}]X \wedge \langle true \rangle true) \wedge \\
&((b_1) \wedge receive\_speed\_pulses(eighty) \wedge (b_2 \approx NL) \rightarrow \\
&\mu X.[\overline{send\_allowed\_speed\_ERTMS(80)}]X \wedge \langle true \rangle true) \wedge \\
&((b_1) \wedge receive\_speed\_pulses(onethirty) \wedge (b_2 \approx NL) \rightarrow \\
&\mu X.[\overline{send\_allowed\_speed\_ERTMS(130)}]X \wedge \langle true \rangle true) \wedge \\
&((b_1) \wedge receive\_speed\_pulses(oneforty) \wedge (b_2 \approx NL) \rightarrow \\
&\mu X.[\overline{send\_allowed\_speed\_ERTMS(140)}]X \wedge \langle true \rangle true) \wedge \\
&((b_1) \wedge receive\_speed\_pulses(none) \wedge (b_2 \approx NL) \rightarrow \\
&\mu X.[\overline{send\_allowed\_speed\_ERTMS(40)}]X \wedge \langle true \rangle true) \wedge \\
&((b_1) \wedge receive\_speed\_pulses(sixty) \wedge (b_2 \approx NL) \rightarrow \\
&\mu X.[\overline{receive\_speed\_limit(60)}]X \wedge \langle true \rangle true) \wedge \\
&((b_1) \wedge receive\_speed\_pulses(eighty) \wedge (b_2 \approx NL) \rightarrow \\
&\mu X.[\overline{receive\_speed\_limit(80)}]X \wedge \langle true \rangle true) \wedge \\
&((b_1) \wedge receive\_speed\_pulses(onethirty) \wedge (b_2 \approx NL) \rightarrow \\
&\mu X.[\overline{receive\_speed\_limit(130)}]X \wedge \langle true \rangle true) \wedge \\
&((b_1) \wedge receive\_speed\_pulses(oneforty) \wedge (b_2 \approx NL) \rightarrow \\
&\mu X.[\overline{receive\_speed\_limit(140)}]X \wedge \langle true \rangle true) \wedge \\
&((b_1) \wedge receive\_speed\_pulses(none) \wedge (b_2 \approx NL) \rightarrow
\end{aligned}
$$

$\mu X.[\overline{receive\_speed\_limit(40)}]X \wedge \langle true \rangle true) \wedge$
$((b_1) \wedge receive\_speed\_pulses(sixty) \wedge (b_2 \approx BE) \rightarrow$
$\mu X.[\overline{send\_allowed\_speed\_ERTMS(50)}]X \wedge \langle true \rangle true) \wedge$
$((b_1) \wedge receive\_speed\_pulses(eighty) \wedge (b_2 \approx BE) \rightarrow$
$\mu X.[\overline{send\_allowed\_speed\_ERTMS(70)}]X \wedge \langle true \rangle true) \wedge$
$((b_1) \wedge receive\_speed\_pulses(onethirty) \wedge (b_2 \approx BE) \rightarrow$
$\mu X.[\overline{send\_allowed\_speed\_ERTMS(120)}]X \wedge \langle true \rangle true) \wedge$
$((b_1) \wedge receive\_speed\_pulses(oneforty) \wedge (b_2 \approx BE) \rightarrow$
$\mu X.[\overline{send\_allowed\_speed\_ERTMS(130)}]X \wedge \langle true \rangle true) \wedge$
$((b_1) \wedge receive\_speed\_pulses(none) \wedge (b_2 \approx BE) \rightarrow$
$\mu X.[\overline{send\_allowed\_speed\_ERTMS(30)}]X \wedge \langle true \rangle true) \wedge$
$((b_1) \wedge receive\_speed\_pulses(sixty) \wedge (b_2 \approx BE) \rightarrow$
$\mu X.[\overline{receive\_speed\_limit(50)}]X \wedge \langle true \rangle true) \wedge$
$((b_1) \wedge receive\_speed\_pulses(eighty) \wedge (b_2 \approx BE) \rightarrow$
$\mu X.[\overline{receive\_speed\_limit(70)}]X \wedge \langle true \rangle true) \wedge$
$((b_1) \wedge receive\_speed\_pulses(onethirty) \wedge (b_2 \approx BE) \rightarrow$
$\mu X.[\overline{receive\_speed\_limit(120)}]X \wedge \langle true \rangle true) \wedge$
$((b_1) \wedge receive\_speed\_pulses(oneforty) \wedge (b_2 \approx BE) \rightarrow$
$\mu X.[\overline{receive\_speed\_limit(130)}]X \wedge \langle true \rangle true) \wedge$
$((b_1) \wedge receive\_speed\_pulses(none) \wedge (b_2 \approx BE) \rightarrow$
$\mu X.[\overline{receive\_speed\_limit(30)}]X \wedge \langle true \rangle true)$

3. **a** The ATP will instruct ERTMS to ring the cabin bell if the ATP detects that the speed of the train is higher than allowed and the ATP is active.

   b If the last *receive_current_speed(v_1)* and last *receive_speed_limit(v_2)* hold that $v_1 > v_2$, the *bell_ring* action will follow if an *ATPActive* is done without an *ATPInactive* in between.

   c $\nu X(v_m : \mathbb{N} := 0, v_c : \mathbb{N} := 0, b_1 : \mathbb{B} := true).$
   $\forall v_1 : \mathbb{N}.[receive\_current\_speed(v_1)]X(v_m, v_1, b_1) \wedge$
   $\forall v_2 : \mathbb{N}.[receive\_speed\_limit]X(v_2, v_c, b_1) \wedge$
   $[ATPActive]X(v_m, v_c, true) \wedge$
   $[ATPInactive]X(v_m, v_c, false) \wedge$
   $\overline{[(\exists v_3 : \mathbb{N}.receive\_current\_speed(v_3) \bigcup receive\_speed\_limit(v_3) \bigcup ATPActive \bigcup ATPInactive)]}$
   $X(v_m, v_c, b_1) \wedge$
   $((v_c > v_m) \wedge b_1 \rightarrow \mu Y.[\overline{bell\_ring}]Y \wedge \langle true \rangle)$

4. **a** When the speed of a freight train decreases to the allowed speed plus 20 from higher, the ATP will ring the cabin bell 3 times to indicate that the driver is allowed to release the brake.

   b The action *Bellreminder* will happen only if the *receive_train_type(freight)* is done and the last *receive_current_speed(v_1)* and last *receive_speed_limit(v_2)* hold that $v_1 \leq v_2 + 20$.

   c $\nu X(v_m : \mathbb{N} := 0, v_c : \mathbb{N} := 0, b_1 : \mathbb{B} := true, b_2 : \mathbb{B} := false).$
   $\forall v_1 : \mathbb{N}.[receive\_current\_speed(v_1)]X(v_m, v_1, b_1, b_2) \wedge$

$\forall v_2 : \mathbb{N}.[receive\_speed\_limit(v_2)]X(v_2, v_c, b_1, b_2) \wedge$
$[ATPActive]X(v_m, v_c, true, b_2) \wedge$
$[ATPInactive]X(v_m, v_c, false, b_2) \wedge$
$[receive\_train\_type(freight)]X(v_m, v_c, b_1, true) \wedge$
$[receive\_train\_type(other)]X(v_m, v_c, b_1, false) \wedge$
$[\overline{(\exists v_3 : \mathbb{N}.receive\_current\_speed(v_3) \bigcup receive\_speed\_limit(v_3) \bigcup ATPActive}$
$\overline{\bigcup ATPInactive \bigcup receive\_train\_type(freight) \bigcup receive\_train\_type(other))}]$
$X(v_m, v_c, b_1, b_2) \wedge$
$((v_c < v_m + 20) \wedge b_1 \wedge b_2 \rightarrow \mu Z.[\overline{bell\_reminder}]Z \wedge \langle true\rangle true)$

5. **a** The ATP will instruct the ERTMS to apply an emergency brake when the drivers don't react within 3 seconds after the bell begins ringing, or the driver releases the brake earlier than the train reaches the desired speed if the ATP is active.

   b Before sending an *instru_emergency_brake* action, one of the following actions must happen: *bellRing* followed by *receive_bell_expired* without *receive_brake_status(braking)* in between, or the *receive_brake_status(braking)* followed by *receive_brake_status(releasing)* without *safe_Speed* and *bell_reminder* in between. What's more the action *ATPActive* must take place.

   c
   - $[true^* \cdot ATPInactive \cdot \overline{ATPActive}^* \cdot instru\_emergency\_brake]false$
   - $[true^\star \cdot receive\_brake\_status(braking) \cdot \overline{safe\_speed \bigcup bell\_reminder}^* \cdot receive\_brake\_status(releasing)]\mu X.([\overline{instru\_emergency\_brake}]X \wedge \langle true\rangle true)$
   - $[true^* \cdot bell\_ring \cdot \overline{receive\_brake\_status(braking)}^* \cdot receive\_bell\_expired]$ $\mu X.[\overline{instru\_emergency\_brake}]X \wedge \langle true\rangle true)$

6. **a** The ATP can be reset by drivers after an emergency brake is applied and the train is stopped..

   b The action *reset_ATP* can happen after the *instru_emergency_brake* action followed by *receive_current_speed(0)*.

   c $[true^* \cdot instru\_emergency\_brake \cdot \overline{receive\_current\_speed(0)}^* \cdot reset\_ATP]false$

7. **a** ATP will instruct the ERTMS to automatically stop the train when a stop beacon signal is read.

   b The *instru_emergency_brake* action happens after *receive_speed_pulses(beacon)* is done.

   c $[true^* \cdot receive\_speed\_pulses(beacon)]\mu X.([\overline{instru\_emergency\_brake}]X \wedge \langle true\rangle true)$

6 Behavior of All Controllers realized by mCRL2

In this section, a behavior introduction and the related mCRL2 code for each controller will be presented.

The ATP++ part which is responsible for receiving the pulse signals from the outer Track system, is a customized controller used for interpreting speed codes to corresponding speeds according to different countries' rules or laws. Given that the ATP++ is required to continuously read from the Track and interpret the speed, its behavior should be recursive. ATP ++ can also detect the beacon signal and then change the ATP status.

ATPpp

```
ATP++(pls : Pulses, ctry : Country) =
    receive_pulses(sixty) .
      send_limit(map_speed_limit(sixty, ctry))
      .ATPpp(sixty, ctry)
  + receive_pulses(none) .
      send_limit(map_speed_limit(none, ctry))
      .ATPpp(none, ctry)
  + receive_pulses(eighty) .
      send_limit(map_speed_limit(eighty, ctry))
      .ATPpp(eighty, ctry)
  + receive_pulses(onethirty) .
      send_limit(map_speed_limit(onethirty, ctry))
      .ATPpp(onethirty, ctry)
  + receive_pulses(oneforty) .
      send_limit(map_speed_limit(oneforty, ctry))
      .ATPpp(oneforty, ctry)
  + receive_pulses(beacon) .
      send_ATP_status(pause) . instru_emergency_brake
      . ATPpp(pls=beacon)
  + receive_country(NL) . ATPpp(pls, NL)
  + receive_country(BE) . ATPpp(pls, BE);
```

After receiving a speed limit from the ATP++ as well as an ATP status and a current speed from the General, the ATP starts to send bell instructions to the Bell controller. In order to send the correct bell instructions, the ATP needs to implement a series of conditional statements to check and update the ATP status. Only if the ATP is on, the train is over speeding and the bell is not ringing, it is possible to change the bell state.

ATP

```
ATP(train : Train) =
      sum n : Speed . receive_limit_a(n)
```

```
      . ATP(update_aspeed(train, n))
    + sum m : Speed . receive_current_speed_a(m)
      . ATP(update_cspeed(train, m))
    + receive_ATP_status(active) .
    ((is_ATPactive(train)==false) &&
    (is_ATPpaused(train)==false)) ->
    ATP(update_ATP(train, active)) <>
    ATP(train)
    + receive_ATP_status(inactive)
       . ATP(update_ATP(train, inactive))
    + receive_ATP_status(pause)
      . ATP(update_ATP(train, pause))
    + (is_ATPactive(train))->(
        is_speeding(train) ->(
           (is_bell_ringing(train)==false) ->(
             speeding . send_bell_state(bellOn)
             . ATP(update_bell(train, bellOn))
           )
        )
      )
  ;
```

The Bell controller is designed to receive bell instructions from the ATP and then communicate with the outer ERTMS system.

Bell controller

```
BellController =
      receive_bell_state(bellOn).bell_ring.BellController
    + receive_bell_state(bellOff).bell_stop.BellController;
```

Similarly, the Brake controller directly communicate with the outer ERTMS system and General controller in case there is an emergency brake state.

Brake controller

```
BrakeController =
    receive_emergency_brake . emergency_brake
    . BrakeController;
```

The General controller deals with communication between the ATP, the ATP++ and the outer ERTMS system. By identifying the train type and comparing the current train speed with the speed limit, the General controller decides whether safe speed action or bell reminder action should be executed.Besides the General controller is also responsible for checking if the train is running in ATP area or ERTMS area and successively determines the ATP status. Moreover, it also controls the reset of the whole ATP system when it detects the train is at a full stop. The last behaviors of the General controller are that it determines the brake status and transmit it to the Brake controller. We apply the emergency brake if the driver releases the brake when the train is still speeding. If the train is a freight train, the driver is allowed to release the brake after a certain time.

General controller

```
General(train : Train) =

      sum n : Speed . receive_limit_g(n)
        . send_allowed_speed_ERTMS(n)
        .  General(update_aspeed(train, n))
    + sum m : Speed . receive_current_speed_g(m)
        . General(update_cspeed(train, m))

    + receive_train_type(freight)
       .General(update_type(train, freight))
    + receive_train_type(other)
       .General(update_type(train, other))


    + receive_track_type(ATPArea)
       .leave_ERTMS.send_ATP_status(active)
    + receive_track_type(ERTMSArea)
       .enter_ERTMS.send_ATP_status(inactive)

    + reset_command . is_stopped(train) -> (
        is_ATPpaused(train) -> (
          send_ATP_status(inactive) . General(init_train)
        ) <>General(train)
      ) <>General(train)



    + receive_brake_status(braking)
    . General(update_brake(train, braking))

    + receive_brake_status(releasing) .
        (is_speeding(train) && brake(train) == braking)
        -> (((is_freight(train)==false) ||
               (is_freight(train) &&
                 cspeed(train) >
                 (aspeed(train) + extra_freight)))
                 ->(
            instru_emergency_brake
            .send_ATP_status(pause)
```

```
            .General(update_ATP(train, pause))
          ) <>General(update_brake(train, releasing))
        ) <> General(update_brake(train, releasing))
    + receive_bell_expired
    . ((is_brake_applying(train) == false) &&
    (cspeed(train) > aspeed(train)))
    ->(
        instru_emergency_brake
        .send_ATP_status(pause)
        .General(update_ATP(train, pause))
      ) <> General(train)

  ;
```

7 Verification for the mCRL2 design

The mCRL2 verification for each requirement listed above is shown in this section.The results show that we have passed most of the requirement verification. The operating system used for verification is Windows 10 and the computer used for testing is DELL XPS 15. The mCRL2 we employed is the latest windows(64-bit) release. Three command lines are executed during the verification process where vm01 can be replaced by corresponding names of verification file.

```
mcrl22lps -lregular2 ATP_spec.mcrl2 ATP_spec.lps
```

```
lps2pbes ATP_spec.lps -f vm01.mcf vm01.pbes
```

```
pbes2bool vm01.pbes
```

Requirement 1

```
[true*.enter_ERTMS]
mu X.[!ATPInactive]X&&<true>true &&
[true*.leave_ERTMS]
mu X.[!ATPActive]X&&<true>true
```

Result: True;

Requirement 2

```
form nu X(b1:Bool=true,b2:Country=NL).
[ATPActive]X(true,b2)&&
[ATPInactive]X(false,b2)&&
[receive_country(NL)]X(b1,NL)&&
```

```
[receive_country(BE)]X(b1,BE)&&
[!(ATPActive||ATPInactive||receive_country(NL)
||receive_country(BE))]X(b1,b2)&&
([val(b1)&&receive_speed_pulses(sixty)&&val(b2==NL)]
mu X.[!send_allowed_speed_ERTMS(60)]X&&<true>true)&&
([val(b1)&&receive_speed_pulses(eighty)&&val(b2==NL)]
mu X.[!send_allowed_speed_ERTMS(80)]X&&<true>true)&&
([val(b1)&&receive_speed_pulses(onethirty)&&val(b2==NL)]
mu X.[!send_allowed_speed_ERTMS(130)]X&&<true>true)&&
([val(b1)&&receive_speed_pulses(oneforty)&&val(b2==NL)]
mu X.[!send_allowed_speed_ERTMS(140)]X&&<true>true)&&
([val(b1)&&receive_speed_pulses(none)&&val(b2==NL)]
mu X.[!send_allowed_speed_ERTMS(40)]X&&<true>true)&&
([val(b1)&&receive_speed_pulses(sixty)&&val(b2==NL)]
mu X.[!receive_speed_limit(60)]X&&<true>true)&&
([val(b1)&&receive_speed_pulses(eighty)&&val(b2==NL)]
mu X.[!receive_speed_limit(80)]X&&<true>true)&&
([val(b1)&&receive_speed_pulses(onethirty)&&val(b2==NL)
]mu X.[!receive_speed_limit(130)]X&&<true>true)&&
([val(b1)&&receive_speed_pulses(oneforty)&&val(b2==NL)]
mu X.[!receive_speed_limit(140)]X&&<true>true)&&
([val(b1)&&receive_speed_pulses(none)&&val(b2==NL)]
mu X.[!receive_speed_limit(40)]X&&<true>true)&&
([val(b1)&&receive_speed_pulses(sixty)&&val(b2==BE)]
mu X.[!send_allowed_speed_ERTMS(50)]X&&<true>true)&&
([val(b1)&&receive_speed_pulses(eighty)&&val(b2==BE)]
mu X.[!send_allowed_speed_ERTMS(70)]X&&<true>true)&&
([val(b1)&&receive_speed_pulses(onethirty)&&val(b2==BE)]
mu X.[!send_allowed_speed_ERTMS(120)]X&&<true>true)&&
([val(b1)&&receive_speed_pulses(none)&&val(b2==BE)]
mu X.[!send_allowed_speed_ERTMS(30)]X&&<true>true)&&
([val(b1)&&receive_speed_pulses(oneforty)&&val(b2==BE)]
mu X.[!send_allowed_speed_ERTMS(130)]X&&<true>true)&&
([val(b1)&&receive_speed_pulses(sixty)&&val(b2==BE)]
mu X.[!receive_speed_limit(50)]X&&<true>true)&&
([val(b1)&&receive_speed_pulses(eighty)&&val(b2==BE)]
mu X.[!receive_speed_limit(70)]X&&<true>true)&&
([val(b1)&&receive_speed_pulses(onethirty)&&val(b2==BE)]
mu X.[!receive_speed_limit(120)]X&&<true>true)&&
([val(b1)&&receive_speed_pulses(none)&&val(b2==BE)]
mu X.[!receive_speed_limit(30)]X&&<true>true)&&
([val(b1)&&receive_speed_pulses(oneforty)&&val(b2==BE)]
mu X.[!receive_speed_limit(130)]X&&<true>true);
```

Result: True

Requirement 3

```
form nu X(vm:Nat=0,vc:Nat=0,b1:Bool=true).
(forall v1:Nat. [receive_current_speed(v1)]X(vm,v1,b1))&&
(forall v2:Nat. [receive_speed_limit(v2)]X(v2,vc,b1))&&
[!(exists v3:Nat. receive_current_speed(v3)
||receive_speed_limit(v3))]X(vm,vc,b1)&&
[ATPActive]X(vm,vc,true)&&
[ATPInactive]X(vm,vc,false)&&
[!(ATPActive||ATPInactive)]X(vm,vc,b1)&&
(val(b1==true)&&val(vc>vm)=>mu Y.[!bell_ring]Y&&<true>true);
```

Result: True;

Requirement 4

```
form nu X(vm:Nat=0,vc:Nat=0,b1:Bool=true,b2:Bool=false).
(forall v1:Nat.([receive_current_speed(v1)]X(vm,v1,b1,b2)))&&
(forall v2:Nat.([receive_speed_limit(v2)]X(v2,vc,b1,b2)))&&
[ATPActive]X(vm,vc,true,b2)&&
[ATPInactive]X(vm,vc,false,b2)&&
[receive_train_type(freight)]X(vm,vc,b1,true)&&
[receive_train_type(other)]X(vm,vc,b1,false)&&
[!(exists v3:Nat. (receive_current_speed(v3)||receive_speed
_limit(v3))||ATPActive||ATPInactive
||receive_train_type(freight)||receive_train_type(other))]
X(vm,vc,b1,b2)&&
(val(b1==true)&&val(b2==true)&&val(vc<vm+20)
=>mu Y.[!bell_reminder]Y&&<true>true);
```

Result: True;

Requirement 5

```
[true*.bell_ring.!(receive_brake_status(braking))
.receive_bell_expired]
mu X.[!instru_emergency_brake]X&&<true>true]&&
[true*.ATPInactive.(!ATPActive)*.instru_emergency_brake]false&&
[true*.receive_brake_status(braking)
.(!(safe_speed||bell_reminder))*.receive_brake_status(releasing)]
mu X.[!instru_emergency_brake]X&&<true>true
```

Result: True;

Requirement 6

```
[true*.instru_emergency_brake.
```

```
!(receive_current_speed(0))*.reset_ATP]false
```

Result: True;

Requirement 7

```
[true*.receive_pulses(beacon)]
mu X.[!instru_emergency_brake]X&&<true>true
```

Result: True;

8 APPENDIX-The entire mCRL2 code

```
% two components both need the same message

sort
  Speed = Nat;

  Pulses = struct sixty | eighty | onethirty |
  oneforty | special | beacon | none;
  Track  = struct ATPArea | ERTMSArea;
  Country = struct NL | BE;
  BellState = struct bellOn | bellOff;
  BrakeStatus = struct braking | emergencyBrake |
  releasing;
  TrainType = struct freight | other;
  ATPStatus = struct active | inactive | pause;
  % The ATP will come to a pause status after an
  emergency brake

  Train =
    struct train_info(
      bell: BellState,
      brake: BrakeStatus,
      type: TrainType,
      ATP: ATPStatus,
      country: Country,
      cspeed : Speed,
      aspeed : Speed
    );

map
  init_train : Train;
```

```
eqn
  init_train = train_info(bellOff, releasing,
                     other, active, NL, 0, 0);

map
  extra_freight : Nat;

  no_signal: Speed;
  map_speed_limit: Pulses # Country -> Speed;

  is_bell_ringing: Train -> Bool;
  is_brake_applying: Train -> Bool;
  is_freight: Train -> Bool;
  is_ATPactive: Train -> Bool;
  is_ATPpaused: Train -> Bool;
  in_what_country: Train -> Country;
  is_speeding: Train -> Bool;
  is_stopped : Train -> Bool;

  update_bell: Train # BellState -> Train;
  update_brake: Train # BrakeStatus -> Train;
  update_type: Train # TrainType -> Train;
  update_ATP: Train # ATPStatus -> Train;
  update_country: Train # Country -> Train;
  update_cspeed : Train # Speed -> Train;
  update_aspeed : Train # Speed -> Train;

var
  v_train : Train;
  v_bell : BellState;
  v_brake : BrakeStatus;
  v_type : TrainType;
  v_atp : ATPStatus;
  v_country: Country;
  v_aspeed : Speed;
  v_cspeed : Speed;

eqn
  % The driver is allowed to release
  the brake when the train speed is
  below allowed speed + extra_freight

  extra_freight = 20;

  is_bell_ringing(v_train) = bell(v_train) == bellOn;
  is_brake_applying(v_train) = brake(v_train) == braking;
```

```
  is_freight(v_train) = type(v_train) == freight;
  is_ATPactive(v_train) = ATP(v_train) == active;
  is_ATPpaused(v_train) = ATP(v_train) == pause;
  is_speeding(v_train) = (cspeed(v_train)
       > aspeed(v_train));
  is_stopped(v_train) = (cspeed(v_train) == 0);


  update_bell(v_train, v_bell) =
    train_info(v_bell, brake(v_train),
       type(v_train), ATP(v_train),
       country(v_train), cspeed(v_train),
       aspeed(v_train));
  update_brake(v_train, v_brake) =
    train_info(bell(v_train), v_brake,
       type(v_train), ATP(v_train),
       country(v_train), cspeed(v_train),
       aspeed(v_train));
  update_type(v_train, v_type) =
    train_info(bell(v_train), brake(v_train),
       v_type, ATP(v_train),
       country(v_train), cspeed(v_train),
       aspeed(v_train));
  update_ATP(v_train, v_atp) =
     train_info(bell(v_train),
        brake(v_train), type(v_train),
        v_atp, country(v_train),
        cspeed(v_train),
        aspeed(v_train));
  update_country(v_train, v_country) =
     train_info(bell(v_train), brake(v_train),
        type(v_train), ATP(v_train),
        v_country, cspeed(v_train),
        aspeed(v_train));
  update_cspeed(v_train, v_cspeed) =
     train_info(bell(v_train), brake(v_train),
        type(v_train), ATP(v_train),
        country(v_train), v_cspeed,
        aspeed(v_train));
  update_aspeed(v_train, v_aspeed) =
     train_info(bell(v_train), brake(v_train),
        type(v_train), ATP(v_train),
        country(v_train), cspeed(v_train),
        v_aspeed);

%  update
```

```
  no_signal = 40;
  map_speed_limit(sixty, NL) = 60;
  map_speed_limit(eighty, NL) = 80;
  map_speed_limit(onethirty, NL) = 130;
  map_speed_limit(oneforty, NL) = 140;
  map_speed_limit(none, NL) = 40;
  % Road condition in BE may not be as good as in NL.
  map_speed_limit(sixty, BE) = 50;
  map_speed_limit(eighty, BE) = 70;
  map_speed_limit(onethirty, BE) = 120;
  map_speed_limit(oneforty, BE) = 130;
  map_speed_limit(none, BE) = 30;

act

  reset_pressed , reset_command , reset_ATP;

  % speed limit related actions
  send_limit, receive_limit_g, receive_limit_a,
  transmit_speed_limit: Speed;

  send_speed_code , receive_speed_code ,
  transmit_speed_code : Pulses;

  % current speed related actions
  receive_current_speed_g, receive_current_speed_a,
  send_current_speed, transmit_current_speed: Speed;

  % pulses
  send_pulses, receive_pulses,
  transmit_speed_pulses: Pulses;

  % bell control
  bell_ring, bell_stop;
  send_bell_state, receive_bell_state,
  transmit_bell_state: BellState;

  % running status indicator
  speeding, safe_speed;

  % ATP status
  send_ATP_status, receive_ATP_status,
  transmit_ATP_status : ATPStatus;

  % brake status message from ERTMS
  send_brake_status, receive_brake_status,
```

```
  transmit_brake_status : BrakeStatus;
  % train type message from ERTMS
  send_train_type, receive_train_type,
  transmit_train_type: TrainType;
  receive_track_type, send_track_type,
  transmit_track_type: Track;
  send_country, receive_country,
  transmit_country : Country;
  send_bell_expired, receive_bell_expired,
  transmit_bell_expired;
  leave_ERTMS, enter_ERTMS;

  % to ERTMS
  send_allowed_speed_ERTMS : Speed;
  instru_emergency_brake, receive_emergency_brake,
  transmit_emergency_brake;
  emergency_brake, bell_reminder;

  ATPActive, ATPInactive;

proc

  Track =
    send_pulses(sixty).Track+
    send_pulses(eighty).Track+
    send_pulses(onethirty).Track+
    send_pulses(oneforty).Track+
    send_pulses(special).Track+
    send_pulses(beacon).Track+
    send_pulses(none).Track
  ;

  BellController =
      receive_bell_state(bellOn).bell_ring.BellController
      +
      receive_bell_state(bellOff).bell_stop.BellController;

  BrakeController =
    receive_emergency_brake . emergency_brake
    . BrakeController;


    ATPpp(pls : Pulses, ctry : Country) =
    receive_pulses(sixty) .
      send_limit(map_speed_limit(sixty, ctry))
      .ATPpp(sixty, ctry)
```

```
+ receive_pulses(none) .
    send_limit(map_speed_limit(none, ctry))
    .ATPpp(none, ctry)
+ receive_pulses(eighty) .
    send_limit(map_speed_limit(eighty, ctry))
    .ATPpp(eighty, ctry)
+ receive_pulses(onethirty) .
    send_limit(map_speed_limit(onethirty, ctry))
    .ATPpp(onethirty, ctry)
+ receive_pulses(oneforty) .
    send_limit(map_speed_limit(oneforty, ctry))
    .ATPpp(oneforty, ctry)
+ receive_pulses(beacon) .
    send_ATP_status(pause) . instru_emergency_brake
    . ATPpp(pls=beacon)
+ receive_country(NL) . ATPpp(pls, NL)
+ receive_country(BE) . ATPpp(pls, BE);



General(train : Train) =

    sum n : Speed . receive_limit_g(n)
    . send_allowed_speed_ERTMS(n)
    . ((cspeed(train) > n) -> speeding <> safe_speed)
    .  General(update_aspeed(train, n))
    + sum m : Speed . receive_current_speed_g(m)
    . ((aspeed(train) < m) -> safe_speed <> speeding)
    . ( (is_freight(train) && (m <= (aspeed(train) +
    extra_freight)) && (m > aspeed(train))) ->
    bell_reminder )
     . General(update_cspeed(train, m))

  + receive_train_type(freight)
  .General(update_type(train, freight))
  + receive_train_type(other)
  .General(update_type(train, other))


  + receive_track_type(ATPArea)
  .leave_ERTMS.send_ATP_status(active)
  + receive_track_type(ERTMSArea)
  .enter_ERTMS.send_ATP_status(inactive)

  + reset_command . is_stopped(train) -> (
```

```
      is_ATPpaused(train) -> (
        send_ATP_status(inactive) . General(init_train)
      ) <>
      General(train)
    ) <>
    General(train)


  + receive_brake_status(braking)
  . General(update_brake(train, braking))

  + receive_brake_status(releasing) .
      % we apply the emergency brake if the driver
      release the brake when the train
      is still speeding
      (is_speeding(train) && brake(train) == braking)
      -> (
        % if the train is a freight train,
        the driver is allowed to release
        the brake after a certain time
        ((is_freight(train)==false) ||
        (is_freight(train) &&
           cspeed(train)
           > (aspeed(train) + extra_freight))) ->(
          instru_emergency_brake.send_ATP_status(pause)
          .General(update_ATP(train, pause))
        ) <>
        General(update_brake(train, releasing))
      ) <>
      General(update_brake(train, releasing))
  + receive_bell_expired .
  ((is_brake_applying(train) == false) &&
  (cspeed(train) > aspeed(train))) ->(
      instru_emergency_brake.send_ATP_status(pause)
      .General(update_ATP(train, pause))
    ) <>
    General(train)

;

ATP(train : Train) =
    sum n : Speed . receive_limit_a(n)
    . ATP(update_aspeed(train, n))
  + sum m : Speed . receive_current_speed_a(m)
  . ATP(update_cspeed(train, m))
```

```
    + receive_ATP_status(active) .
    ((is_ATPactive(train)==false) &&
    (is_ATPpaused(train)==false)) ->
    ATP(update_ATP(train, active))
    <> ATP(train)
    + receive_ATP_status(inactive)
    . ATP(update_ATP(train, inactive))
    + receive_ATP_status(pause)
    . ATP(update_ATP(train, pause))
    + (is_ATPactive(train))->(
        is_speeding(train) ->(
           (is_bell_ringing(train)==false) ->(
             speeding . send_bell_state(bellOn)
             . ATP(update_bell(train, bellOn))
           )
        )
      )
  ;

  ERTMS = send_current_speed(50).ERTMS
        + send_current_speed(0).ERTMS
        + send_train_type(freight).ERTMS
        + send_train_type(other).ERTMS
        + send_track_type(ATPArea).ERTMS
        + send_track_type(ERTMSArea).ERTMS
        + send_brake_status(braking).ERTMS
        + send_brake_status(releasing).ERTMS
        + send_country(NL).ERTMS
        + send_country(BE).ERTMS
        + reset_pressed.ERTMS
        + send_bell_expired.ERTMS
  ;

init
  allow(
    {
      transmit_speed_pulses, ATPActive, ATPInactive,
      instru_emergency_brake,
      emergency_brake,
      transmit_speed_limit,
      transmit_current_speed,
      bell_ring, bell_stop, % instructions to ERTMS
      speeding,
      safe_speed,
      transmit_bell_state,
      transmit_brake_status,
```

```
    transmit_ATP_status,
    transmit_train_type,
    transmit_speed_code,
    transmit_country,
    transmit_bell_expired,
    enter_ERTMS,
    leave_ERTMS,
    reset_ATP,
    bell_reminder,
    send_allowed_speed_ERTMS
  },
  comm(
    {
      receive_pulses |
      send_pulses -> transmit_speed_pulses,
      send_speed_code |
      receive_speed_code -> transmit_speed_code,
      send_limit  | receive_limit_a |
      receive_limit_g -> transmit_speed_limit,
      send_current_speed | receive_current_speed_a |
      receive_current_speed_g
        -> transmit_current_speed,
      instru_emergency_brake |
      receive_emergency_brake
        -> transmit_emergency_brake,
      send_ATP_status |
      receive_ATP_status -> transmit_ATP_status,
      send_bell_state |
      receive_bell_state -> transmit_bell_state,
      send_brake_status |
      receive_brake_status
         -> transmit_brake_status,
      send_train_type |
      receive_train_type -> transmit_train_type,
      send_track_type |
      receive_track_type -> transmit_track_type,
      reset_pressed |
      reset_command -> reset_ATP,
      send_country |
      receive_country -> transmit_country,
      send_bell_expired |
      receive_bell_expired -> transmit_bell_expired
    },
    General(init_train) ||  ATP(init_train)   ||
    ERTMS || Track || BellController ||
    BrakeController || ATPpp(none, NL
```

```
      )
    )
  );
```