IJR International Journal of Railway
Vol. 5, No. 2 / June 2012, pp. 65-70

The Korean Society for Railway

Formal Analysis of Automatic Train Protection and Block System for Regional Line Using VDM++

Guo Xie†, Xinhong Hei**, Hiroshi Mochizuki*, Sei Takahashi* and Hideo Nakamura*

Abstract

This paper introduced a novel railway system, Automatic Train Protection and Block (ATPB) briefly, which is proposed to improve the efficiency of existing regional train lines with low cost in Japan. The biggest superiority of ATPB system is a great use of universal and mature technologies, such as GPS and regular mobile telephone networks, so that there is nearly no increment of trackside equipments in the reconstruction. Then in order to guarantee the system safety, a formal model of ATPB is established and analyzed by formal method VDM++. Firstly, the specification is specified by VDM++ formally without ambiguity. Secondly, its internal consistency is proved by discharging the proof obligations. And finally, its satisfiability is checked by systematic testing, which executes specification and checks the outputs against corresponding inputs.

Key words **:** *ATPB, Railway System, Formal Methods, VDM++*

1. Introduction

How to improve the utilization rate and efficiency, and ensure the safety and reliability with low cost at the same time, has become an increasingly serious problem for existing regional train lines in Japan. Until now many of new control and management systems with high performance, such as COSMOS by JR-EAST and COMTRAC by JR-CENTRAL have been developed for Shinkansen in Japan, and achieved great success. However, the high cost of these powerful systems cannot be afforded by the regional lines whose operation with low profit margin.

In terms of these current problems, a novel Automatic Train Protection and Block (ATPB) railway system is proposed by the author aid in the reconstruction of regional train lines in Japan. After studying the system and clarifying the requirements, the specification should have to be depicted firstly, and then the software is designed in accordance with the specification. As you know, errors resulting from the ambiguity and contradictory of specification, are very difficult to detect in the following steps. Furthermore, railway system is safety-critical system [1,2], whose failure may result in death or serious injury to people. The common approach to detect and remove errors and bugs by testing the possible problems as is usually inadequate. By contrast, the formal methods (FMs), who specify and prove system in a rigorous mathematical way, are highly recommended for Software Safety Integrity Level 4 (SWSIL4) systems by international standards, such as EN 50128, EN 50129. They can eliminate the ambiguity and verify the specification of the system before accomplishment. Until now, FMs have been adopted in several railway systems [3-5]. However, as a novel railways system, ATPB has not been analyzed formally; what's more, because of the complexity of the railway system structure and difficulties in validation, almost all of the applications of FMs focus on just subsystem of railway, such as interlocking system, automatic train stop (ATS) or level crossing controller. This paper will analyze the whole ATPB system formally by VDM++ [6] as the reasons as following: it is a super set of the ISO standardized notation VDM-SL with rigorous definition, well tool supports, and what`s more important is that it is similar to that of object-oriented programming languages: structural aspects of

† Corresponding author: College of Science and Technology, Nihon University, Chiba, Japan
E-mail : shakoku2010@yahoo.co.jp
* College of Science and Technology, Nihon University, Chiba, Japan
** School of Computer Science and Engineering, Xi'an University of Technology

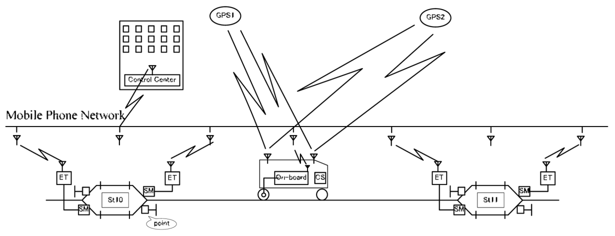

Fig. 1 Overall Structure of ATPB

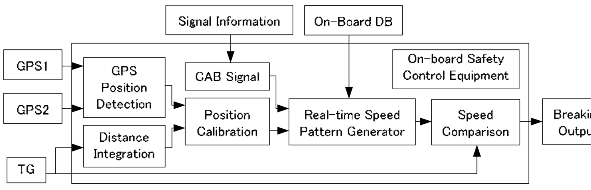

Fig. 2 Position detection based on GPS

software are specified using classes and instance variables, that makes it easy to understand by engineer and programmer.

The aim of this paper is to formalize and validate the specification of ATPB. The rest of this paper is organized as follows: the overall structure of ATPB is depicted in section 2; followed by the formal model for ATPB is established in section 3; and section 4 validates the VDM++ model; at last, section 5 is the conclusion and future work.

2. ATPB Structure

The ATPB system in this paper is a combination of ATP (Automatic Train Protection) and Block control system. The main structure of this system is shown as Fig. 1.

It mainly consists of two parts, the onboard system and control center. The former is composed of: (1) state detection subsystem, which detects the state of train, such as position, speed and train integrity, which means whether all carriages are still in the train; (2) ATP subsystem, which is used to supervise the speed and will take appropriate actions if some certain situations happened (unresponsive train operator, earthquake, disconnected rail, overrun of the authority, etc.) to prevent accidents from happening; (3) communication subsystem, which is used to send and receive messages and commands to and from control center; and (4) record system. The later is mainly composed of: (1) communication system, which is used to send and receive commands and messages to and from trains; (2) interlocking system, which is responsible for the set of points, and route set based on information received from the trains; (3) traffic operation management and so on.

As showed in Fig. 1, the exchange of messages in this system between trains, stations and control center are accomplished by radio communication system. Every train in operation is connected with control center all the time. And the communicating process is described briefly as: when a train approaches a station and arrives at the entry measuring point, which is installed in advance as regard to the next main signal in the train running direction, then the train sends an entry request to control center to apply for an entry permit. Then control center checks the real-time information from its data base, and then sends movement authority to train, if the home is ready to receive this train. When the train pulls into the station, and stops in the train stop territory, it sends an arrival message to control center, then control center releases the corresponding route, which is connecting to the last station, and then checks the departure conditions and sets the points in appropriate position for the train passing through. Meanwhile the train waits in station until received a movement authority. In contrast, once the train overruns of signal, the ATP triggers emergency brake until stop the train. After the train departed from station, and passed the departure measuring point, which is also installed in advance, it sends a departure message to control center. Then control center releases the home and so on. And once even if only one train becomes out of contact with control center, all trains on the line will stop. Furthermore, control center updates the line information real-time, and Automatic Train Protection (ATP) system supervises the movement of train all the time, preventing the accident resulting from over speed.

Contrast to the private communication network system for railway, such as AATC in America, ATACS in Japan, ETCS and CTCS in Europe and China respectively, the biggest superiority of ATPB system is a great use of universal and mature technologies, and nearly no increment of trackside equipments in the reconstruction of the existing regional lines. For example, the communication between trains and control center is accomplished through common mobile telephone network, such as *docomo*, *SoftBank* and *au*; and the position detection of the train is realized by GPS signals and necessary auxiliary equipments. Its flow chart is showed as Fig. 2, and the measurement error can be up to less than 3 meters, that satisfies the actual demand.

Therefore, the reconstruction of the existing regional lines with low cost comes true. Then in order to guarantee the safety and reliability of ATPB software system, its specification is formalized and analyzed in this paper.

3. Formal Specification for ATPB

As mentioned before, ATPB mainly consist of two parts, i.e. onboard system and control center system. They will be modeled separately by VDM++ in this section.

3.1 Onboard system

3.1.1 Running State

The running state of a train includes position, speed and so on. And the position and speed detection of train in our project is realized by analyzing messages from GPS and necessary auxiliary tracking beacons. The flow diagram of position detection is showed as Fig. 2. Considering the variety of analytical algorithms and implementations, the specific definition is not defined, and is described as:

```
EnginePosition:real*real*real==>()
EnginePosition(gps1,gps2,detector)==
is not yet specified;
```

3.1.2 Speed Comparator

The speed comparator is used to supervise the speed always to prevent its speed exceeding the safe speed profile, and ensure the train under speed restrictions, which is determined by maximum line speed, curves, points, tunnel profiles, bridges, etc. This function is accomplished by *SpeedCompare* as follows:

```
public speedStatus: real*real -> State
speedStatus (speed, maxspeed) ==
if speed < (1 - Factor)*maxspeed
then <NormalSpeed>
elseif (1 - Factor)*maxspeed <= speed and
speed < maxspeed
then <AlarmSpeed>
else <EmergencySpeed>;
end SpeedCompare
```

where Factor = 0.1 is a constant, i.e. buffer capacity for the speed. As the specification above, the speed below 90% of the maximum speed limit is permitted speed limit, which is the speed the driver is requested to follow and shall be indicated to the driver. If the speed of train under permitted speed, it gives a safe speed signal. If the speed up to 90% but less than the maximum speed limit, it gives a warning signal; and while the speed exceeds the maximum speed limit, it gives a signal emergency stop.

3.1.3 ATP

ATP system will take appropriate action according to different speed state. According to the received signal, if a safe speed signal, it gives a safe indicate; if a warning signal, it commands the service brake to be applied; and if an emergency brake command, it brakes the train immediately. The corresponding specification by VDM++ is described through function *SpeedCompare* as,

```
public CheckSpeed: real ==> ()
CheckSpeed (speed) ==
(let speedStatus = speedCompare.SpeedStatus
(speed, GetMaxSpeed())
in cases speedStatus:
 <NormalSpeed> -> if not emergencyBrake.
IsEmergencyBrakeState()
then boolDisplay.SetNormal(),
 <AlarmSpeed> -> if not emergencyBrake.
IsEmergencyBrakeState()
then (StartNormalBrake();
record.WriteLog("Alarm", enginePosition)),
<EmergencySpeed> -> (StartEmergencyBrake();
 record.WriteLog("Alarm", enginePosition))
end;);
```

Because after the service brake has been triggered, the brake command shall be revoked when the train speed is equal to or below the permitted speed limit, however, once the emergency brake command was triggered, it brakes the train until stop. Therefore It must judge whether the emergency brake is being applied firstly, even the speed state of a train is <AlarmSpeed> at some time, service brake may be not available, so as the <NormalSpeed>.

3.1.4 Station Arrival

As in the regional single line, the routes are defined relatively to stations, so we can know the determined route and take appropriate operations by querying the other data stored in control center database. While a train arrives in station, the train sends an arrival message to control center. Once the control center receives this message from train, it releases and unlocks the block area relative to the last station, and queries the schedules to get the information of next station. This is realized as,

```
StationArrival: TrainID*StationID
*StationID ==> bool
StationArrival (traid, past, current) ==
(UnBlocking (prior, current);
UnlockRoute (prior, current);
onboards(traid).GetPullin(false);
def next = schedules.GetNextObject(traid,
current, CurrentTime());
in (onboards(traid).SetRunObject (next););;
```

3.1.5 Signal Display

In this specification, the display signals are divided into two types, real and Boolean display. In the specification, it is described by a parent class, Display, and two sub classes, *OnboardRealDisplay* and *OnboardBoolDisplay*. They are described as follows,

```
class OnboardRealDisplay is subclass of Display
operations
public SetCurrentSpeed: Onboard`Speed ==> ()
SetCurrentSpeed (speed) ==
actualSpeed := speed;
public SetMaxSpeed: Onboard`Speed ==> ()
SetMaxSpeed (speed) ==
 maxSpeed := speed;
…
end OnboardRealDisplay
```

The real quantities include current speed, position, speed limit, time, etc. The Boolean signal display is

```
class OnboardBoolDisplay is subclass of Display
operations
public SetNormal: () ==> ()
SetNormal () ==
(normal:= true;
alarm := false;
emergencyBrake := false;
systemError := false;)
pre not alarm and not emergencyBrake;
…
end OnboardBoolDisplay
```

where “pre” predicates that before the execution of operation, alarm and emergencyBrake must be false.

3.1.6 Recorder

The function of record is beneficial to improve the quality and find out the cause of fault. The function starts action when there are new operations, or warning and error happened. And the record information includes name, fault code, time and the place of train where it occurred.

```
public WriteLog: seq1 of char*real ==>()
WriteLog (message, pos) ==
(messageLog := messageLog^[message^ "occurred
 at "^Num2String(pos)^" on "^GetSystemTime()];
systemTime := systemTime ^ [GetSystemTime()];
 errorPostion := errorPostion ^ [pos];);
```

where function Num2String(pos) converts the pos (position) which is real type data into a string type, and GetSystemTime() obtains the current time.

3.2 Control Center

The structure of a station block is showed as Fig. 3. Generally, the mission of control center is controlling and

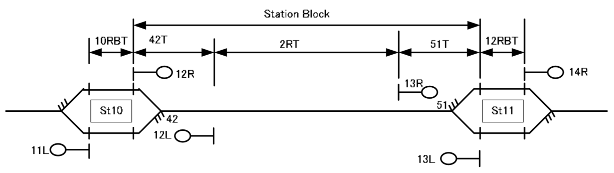

Fig. 3 Structure of station block with signals

setting the signals and points according to the current state of train and railway-network, ensuring no collision and no derail as long as the train runs stick to the signal indicator. Then the functions are formalized as following.

3.2.1 Pull In Check

When train approach the station, it sends the arrival message to control center, and the control center checks the state of relative station and track, then sends receiving message to the train. If the receiving route is prepared, then the function PullinCheck returns true, else return false. This is realized as,

```
public PullinCheck: TrainID*RunObject
*RunObject ==> bool
PullinCheck (traid, prior, next) ==
if interlock(next.station).GetInterlockState().
 platforms(next.platform) = <Free>
then (onboards(traid).GetPullout(true);
SetStrainState(traid,
mk_TrainState(<Running>, prior, next, nil));
interlocks(next.station).
SetPlatform(next.platform, <Bussy>); )
else (onboards(traid).GetPullout(false);
SetStrainState(traid, mk_TrainState(<Waiting>,
 prior, next, <In>)););
```

where TrainID and RunObject are the type of train and station respectively, and RunObject is defined as:

```
RunObject :: station : StationID
                        platform : Platform
                  direction : Schedule`Direction;
```

where StationID is the ID type for station, Direction represents the direction of train, Platform defines number of platform. All of the information, such as state, tracks and interlocking forms are all stored in control center database, which can be called conveniently.

3.2.2 Pull Out Check

When a train is stopping at a station, it keeps connection with the control center. And the control center inquiries the schedule for this train, if the train is going to leaving the station in according with railway timetable, then it checks the corresponding departure condition, including the state of relative station, signals and route. If all conditions are satisfied, it turns the departure signal green, and then sends a departure message to train; otherwise keeps the train waiting, until train receives the departure message. This process is accomplished by a true function as follows,

```
PulloutCheck: TrainID*RunObject
*RunObject ==> bool
PulloutCheck (traid, direction, next) ==
if CheckBlock(current, next) and
CheckRoute(current, next)
then (points.PointOperate(current, next);
onboards(traid).GetPullout(true);
SetStrainState(traid, mk_TrainState(<Running>,
 current, next, nil));
if next.direction = <NORMAL>
then (interlocks(current.station)
.SetRoute(<Right>, <Bussy>);
interlocks(current.station).SetBlock(<Bussy>);
interlocks(next.station)
.SetRoute(<Left>, <Bussy>);
)else
(interlocks(next.station).SetBlock(<Bussy>);
interlocks(next.station)
.SetRoute(<Right>, <Bussy>);
interlocks(current.station)
.SetRoute(<Left>, <Bussy>););)
else (onboards(traid).GetPullout(false);
SetStrainState(traid, mk_TrainState(
<Waiting>, current, next, <Out>)););
```

3.2.3 Route Control

All information of station, route and interlocking forms are all stored in database. So the control center sends appropriate route signal by querying the interlocking forms. The route signal includes route signal for departure, route signal for receiving, and the operations are route release, route locking and presetting a route.

```
SetTurnout: Choice* Turn ==> ()
SetTurnout (choice, turn) ==
if choice = <Right>
 then turnouts.right := turn
else turnouts.left := turn
pre choice in set { <Right>, <Left> } and
turn in set { <UP>, <DOWN> };
```

where the pre is the permission predicates, means that the parameter choice must be either <Right> or <Left>, and turn must be either <UP> or <DOWM>.

This section above listed some main operations relative to block control and automatic train protection described by VDM++. They are clearly defined without ambiguity and understanding deviation. In addition, the enumerated functions above, not only in *Onboard* but also in *Control-Center*, do not mean that they are exhaustive, just some main functions. For example, the emergency treatments for earthquake, etc. are important to railway system, but don't be analyzed in this paper. In the next section, the specification will be validated.

```
        Minimum: set of real -> real
        Minimum ( numbers ) ==
         let mi in set numbers be st
           forall rex in set numbers & rex >= mi
         in  mi
                 (a) Function definition
(forall numbers : set of real & numbers <> {} =>
   (exists mi in set numbers &
     (forall rex in set numbers & rex >= mi)))
                 (b) Proof obligation
```

Fig. 4 Proof obligation for Minimum

4. Model Validation

After completing the formal specification by VDM++, just as any piece of software, the formal specification may contain errors also, mainly about internal consistency and satisfiability. The former which is the correctness of the specification itself, can be found purely by analysis of the model, and will be discussed in section 4.1; and the later who increases the confidence of the specification, can only be checked when more information than just the model is given, and will be discussed in section 4.2.

4.1 Internal consistency proof

The syntax error can be checked by tool easily, however, how to check whether there is an error or contradiction hidden in the specification, is a key problem. The process is called internal consistency proof, which is an unproven theorem stating that a certain property, namely integrity property or proof obligation, must hold in order for the specification to be consistent [7]. If the integrity properties can be proved, then the part referred to in the specification is consistent, otherwise it is inconsistent. That is to say, if all of the integrity properties are demonstrated true, there will be internal consistency and no runtime errors associated with that integrity check. Take an example showed in Fig. 4, where (a) is the definition of function Minimum in class Onboard; (b) is the corresponding proof obligation in the form of VDM++ predicates, which states that under given conditions, the existence of maximum must be proved. This proof is simple, but most of them are time consuming.

The proof of internal consistence includes four steps: (1) Obtain the proof obligations. In this step, they are generated by integrity examiner of VDMTools automatically. The integrity examiner analyses the specification, looking for places where runtime errors may potentially occur and generates a series of integrity properties. Then the proof obligations are rewritten in a independent file manually; (2) Translate the concrete system specification and proof obligations to VDM Abstract Syntax Tree (VDM AST) automatically by Overture parser [8]; (3) Translate VDM AST to HOL model according to the work by S. Vermolen [9]; and (4) Prove the concrete HOL model by theorem prover HOL4 manually. In this process, inconsistencies resulted from wrong type definitions were founded and corrected. In all, 9141 proof obligations are discharged.

4.2 Satisfiability checking

By the work above, we can guarantee that all functions are defined precisely with well internal consistency. However, how to ensure the satisfiability, i.e. satisfying actual requirements, is another key point. As you know, if the result of the specification definition can be predicted before implementation, the reliability and satisfiability of the specification will increase greatly. Unfortunately, there is no formal way to achieve this until now. In our project, we check the satisfiability by executing the specification, and then compare the actual output with the expected value against the corresponding input.

As to the specific approach, the satisfiability of specification is realized by systematic testing using the facility of VDMTools. Specifically, there are three steps involved in this procedure: (1) Prepare a test coverage file manually, i.e. testing cases, which contains information about the specification`s structure but with none of the definitions covered yet; (2) Test the specification by making the interpreter execute calls to the constructs in the specification automatically; and (3) Check the test coverage files manually, including the specification and the coverage information correspondingly, which generated by pretty printer. In this process, by 2826 testing cases, 90% specifications are executed, and the remaining 10% can only be tested after connecting hardware.

5. Conclusion and Future Work

This paper introduced ATPB system briefly, and then formalized the specification by VDM++ formally; at last, the formal specification is validated from two aspects: internal consistency by discharging proof obligations, and satisfiability by systematic testing. The former is a proof approach and is complete. Though the latter is a testing way, even if some flaws or defects are detected, it is far more convenient to correct them just by modifying specification than program after implementation. Through the work in this paper, we can ensure that if the ATPB software is designed strictly in accordance with this formal specification, it will be no runtime error, and satisfy the actual requirements highly.

Lastly, there are two research projects in future, firstly, from the view of the improvement of the analysis approach, how to verify the completeness of formal specification, i.e. all functions are contained in the model, is still a challenge; and then though as mentioned above, the validation of satisfiability by testing is effective, a formal verification is necessary. Secondly, form the view of ATPB, a simulator will be implemented, and more verification is indispensable before actual implementation.

References

1. Bowen and V. Stavridou (1993). “Safety-critical systems: formal methods and standards,” Software Engineering Journal, Vol. 8, pp. 189-209.
2. Knight, J.C. (2002). “Safety critical systems: challenges and directions,” Proceedings of the 24th International Conference on Software Engineering, pp. 547-550.
3. Badeau, F. and Amelot, A. (2005). “Using B as a high level programming language in an industrial project: Roissy VAL,” ZB, H. Treharne, S. King, M. C. Henson, and S. A. Schneider, Eds. Lecture Notes in Computer Science, Vol. 3455, Springer, Berlin, Heidelberg, pp. 334-354.
4. N. Terada and M. Fukuda (2002). “Application of formal methods to the railway signalling systems,” Quarterly Report of RTRI, Vol. 43, No. 4 pp. 169-174.
5. X. Hei, S. Takahashi and H. Nakamura (2009). “Modelling and analyzing component-based distributed railway interlocking system with petri nets,” IEEJ Transactions on Industry, Sec. D, Vol. 129 , No. 5, pp. 455-46.
6. J. Fitzgerald, P. G. Larsen, P. Mukherjee, N. Plat and M. Verhoef (2005). “Validated Designs for Object-oriented Systems,” Springer, New York.
7. B. K. Aichernig and P. G. Larson (1997). “A proof obligation generator for VDM-SL,” FME’97: Industrial Applications and Strengthened Foundations of Formal Methods, volume 1313 of Lecture Notes in Computer Science.
8. P. G. Larsen, J. Fitzgerald, S. Wolff, N. Battle, K. Lausdahl, A.Ribeiro and K. Pierce (2010). “Tutorial for overture/ VDM++,” Overture – Open-source Tools for Formal Modelling TR-2010-03.
9. S. Vermolen, J. Hooman and P. G. Larsen (2010). “Automating consistency proofs of VDM++ models using HOL,” Proceedings of the 25th Symposium On Applied Computing (SAC 2010), (Sierre, Switzerland), ACM, March.

Received(April 26, 2012), Accepted(June 18, 2012)