1. Introduction

Railway signalling is a safety critical system. Special hardware for safety has been used in signalling systems. Recently computer has been widely introduced and software is much more responsible for safety. As far as hardware aspect of safety computer is concerned, there has been a lot of progress. Special devices for safety have been developed and used widely. But the safety technology of software is not matured yet. To increase safety of software, formal methods is expected to be a good solution. Specifications are written in formal specification languages. This enables the specification to be examined by computer. It is even possible that the specification is examined by automatic proofs. In this article, we have designed the specification of digital ATC (Automatic Train Control system) track database with a formal specification language, and analyzed with the mechanical proof. Finally our specification has proved its integrity. We report on the procedure and mention the possibility of the using formal methods and proofs.

2. Formal Methods

Formal methods has been researched to increase quality and productivity of software. The development of software with formal methods looks like Figure 1. The specifications of the systems are written in formal specification languages, which are mathematically and logically defined. It is possible to describe the specification without any ambiguousness. This is important in order to share the specification among many people, but it is often difficult to do in natural languages.

Once the specification is written in formal languages, it is possible to analyze the specification with computer. The analysis includes syntax checking, type checking, systematic testing, test coverage, model checking, etc. And it is even possible that the specification is analyzed with proofs. Some developing environments support automatic code generation directly from the specifications, which reduces the possibility of miscoding.

There is also discussion about the productivity of formal methods. With formal methods, in general, it takes time to define the specification than without them. But it is easier to find mistakes in the early stage of developing. And this reduces the errors and mistakes found in the later stage that usually costs expensive to correct. As a result it is possible to reduce the total cost and time.

There are several formal methods such as Z[1], VDM[2], B[3], CSP[4], etc. For each method, there are some advantages and disadvantages. Various methods can be selected according to the application.

Formal methods are considered to be effective where the quality of software is important. We regard this point as most important. There are already some applications to the railway domain. For example, on developing the automatic train operation system for Paris Metro, B is used[7]. And there were attempt to specify railway systems in formal specification languages, and analyzed with proofs.

In this article we used VDM. This is one of the formal methods widely used. The de-fact standard of VDM is VDM-Toolbox by a Danish software company IFAD. The VDM-Toolbox’s feature includes syntax and type checking, executing specification directly by interpreter or external programs, which can be user interface, test coverage, and code generation. The proofs are not supported so that IFAD emphasize on the modelling of the specification. But IFAD participates in the ESPRIT project PROSPER. This project intended to incorporate academic study on proof into industrial CAD and CASE tools. The proof support of VDM-Toolbox was developed as an application. We had a chance to use the tool and apply this tool for the analysis of digital ATC track database.
3. Digital ATC Track Database
3.1 Digital ATC System

In the existing ATC system, which is used in Shinkansen and some heavy commuting trains, the train is directed the maximum speed for each block. This system has proved its safety, but there are some disadvantages. For example, the maximum speed is directed according to the pre-calculated data, which is based on the worst braking system. If the trains have better braking system, the braking curve becomes multi-step and results in increasing the running time and the interval of the trains.

On the other hand, digital ATC [5] is a new train controlling system we have researched (see Figure 2). In this system, a train receives the ID of the track circuit on which the train exists and the number of the clear block. The train has a track database onboard and looks for the database to find the profile of the preceding track, and calculates the braking curve not to collide into preceding train or derail. In this system, the trains themselves determine maximum speed, while in the conventional ATC system the ground equipment does that. This means the onboard system is much more responsibility for the safety.
Digital ATC System in comparison with Conventional ATC System

3.2 The Structure of the Database

We focus on the database. This is the basis of the calculating the correct braking curve so that high integrity is required. We analyze the structure of the information that expresses the track layout, signalling equipment and some speed restriction. The image of the structure is drawn in Figure 3.

The basic component of the database is track circuits. The information of a track circuit includes signal carrier information, and location and the insulation of the boundaries to other track circuits. Each track circuit is given a unique ID. The connection among track circuits must have consistency.

For each track circuits, the path from which boundary to which boundary trains can move is also registered. On the path, speed restriction and gradients are registered. On top of that, the routes, whose IDs are sent to the train and direct the train what paths the train shall proceed, are defined as the series of the paths.

The track circuits, paths, and routes constitute an area, which corresponds to a station or an intermediate part between stations. A line consists of a series of areas. Several layers of the components can be found in the specification: track circuits -- paths -- routes -- areas -- lines. There are many conditions among components. For example, the paths specified by a route must be connected according to the order. These are complicated requirements. Therefore this is the main target to prove its integrity.
4. Analysis of Database Specification

4.1 Writing Formal Specification from Informal Specification

First we must describe the specification in a formal specification language. In VDM, ISO-standardized language VDM-SL is used as the formal specification language. Here all the components such as track circuits, routes are modelled as types. For example, the track circuit type is modelled as follows:

\[
\text{TrackC:: joints : map Joint_id to Joint} \\
\text{atc : map Direction to ATC} \\
\text{td : TD} \\
\text{atbt : ATBT}
\]

A value with the type TrackC expresses one of the track circuits.

For each type, invariants can be specified as the property that should be always satisfied. For example the invariant of the track circuit is defined as follows:

\[
\text{inv tc ==} \\
\text{card dom tc.joints > 1 and} \\
\text{dom tc.atc = \{<ADIR>, <BDIR>\} and} \\
\text{TD_Used_for_NonInsulated_TrackC(tc.td, tc.atbt, rng tc.joints) and} \\
\text{(tc.atc(<ADIR>).used and tc.atc(<BDIR>.used) =>} \\
\text{tc.atc(<ADIR>).carrier <> tc.atc(<BDIR>).carrier)}
\]

The second line of the invariant, for example, describes that the boundaries of the track circuit should be at least two. The other parts describe the condition about the signals. The invariants are added to other types as well. If the safety requirements are incorporated as invariants, the system is imposed the safety requirements at any times.

After modelling the structure of the database, we specified a small set of the database manipulations. These are specified as functions. Each of the functions takes some parameters and returns a certain value. For example, when a new track circuit is to be registered, the manipulation to register a track circuit is called with the new ID and the property of the track circuit. In this case the current state of database is included in the parameters. In many cases, the manipulating function can be added preconditions that restrict the relationship among the parameters. This precondition is written so that the manipulation keeps the invariants. Then manipulation can keep the safety requirements.

The final specification reaches 985 lines of VDM-SL.

4.2 Proof Obligations

When the formal specification passes the syntax and type checking, proof obligations can be generated automatically. Proof obligations are requirement to ensure the consistency of the specification. All of the proof obligations need to be proved. Even if only one of them are found false, the specification needs to be modified because this means inconsistency.

The proof obligations are classified into several categories. The most important and difficult is invariants proof obligations, that means whenever a value with a certain type must satisfy the invariant of the type. These obligations are generated mainly for the return value of functions. And there are many proof obligations related with the precondition. Any function call needs to satisfy the precondition and this fact results in the proof obligations.

An example of the proof obligations is shown as follows:

\[
\text{forall ar : Area, tcid : TrackC_id, tc: TrackC &} \\
\text{pre_Add_TrackC(ar, tcid, tc) =>} \\
\text{inv_Area(Add_TrackC(ar, tcid, tc))}
\]

This means that for all possible ar, tcid and tc, if the precondition of a function
Add_TrackC is satisfied, the return value of the function Add_TrackC must satisfy invariants of Area. The proof obligations are supposed to examine all possible combination of parameters. If a proof obligation is found true, this corresponds to many test cases while one counter example is enough to deny the proof obligation. So the proof obligations are very powerful requirements.

From 985 lines of the specification, 188 proof obligations in 2104 lines are generated. It is easily seen that the proof obligations are much larger than the original specification. The proof obligations are generated automatically so that the lack of the proof obligation does not happen.

Checking of proof obligations can be done by hand. Some mistakes in the specification are found in this way. What’s more, if the automatic proof is available, it is much easier to check the proof obligations, especially for the large specifications.

4.3 Automatic Proof

Proof support on extended VDM-Toolbox is based on the PROSPER Toolkit[9], which is the main deliverable of the PROSPER project. The PROSPER Toolkit is based on HOL (High Order Logic)[8], which has been developed in the academia. The PROSPER toolkit provides the application interface, and the theory, rewriting rules and reasoning procedure of HOL are available. On top of that, user defined theories can be constructed. The proof support of VDM-Toolbox is constructed in such a way that definitions, theories and rewriting rules for generic VDM-SL specifications are added in the proof engine as well as proof procedures.

The specification and its proof obligations are loaded into proof engine. Two full automatic proving procedures are provided. One is "Sweep", which sweeps easy-to-prove proof obligations away and gives up fast. The other is "Check". This looks further. Even if the proof is not finished, it reports what subgoals remain to be proved, which often helps finding mistakes. Several mistakes in the first specification are found in such a way. Finally among 188 proof obligations, 167 are proved automatically. Then we can concentrate on the rests of obligations.

For the proof obligations not proved automatically, interactive proofs are available. The snapshot of the interactive proof is shown Figure 4. There are several buttons corresponding to proof procedures. For example we can push a button to simply the assumptions and conclusions, and another button to unfold the definition of a function. Users can select the button to proceed the proofs but this is more difficult to use than the automatic proofs. We need to get used to the tool. And it is important to understand the specification well because this helps the proofs. Sometimes without knowledge of the specification it is difficult to determine when to unfold a function. Therefore even for the experts of proofs it takes a time.

The rests of the proof obligations are proved in this way. Now all of the proof obligations are proved. This means that the specification is proved its consistency with computer.
5. Conclusion
We described the specification of digital ATC track database in a formal specification language, and analyzed with help of automatic proofs. Almost 90% (167 out of 188) of the proof obligations are proved fully automatically and the rests of them are proved semi-automatically.

The proof engine used here is under development and needs improvement. With the current proof engine, it takes a time to do the interactive proofs. Not all of VDM-SL notations are available, although this is not a serious problem. And there are many things that cannot be handled by the proof engine. In spite of such many restrictions, we derived a good result and are satisfied with that. Although the good result is not always guaranteed when different specifications are applied, we can expect better results in much more applications when the proof engine becomes more sophisticated.

The proof obligations themselves have very powerful property that ensures the quality of the software. Some mistakes are found through only looking at proof obligations. And the important point is, they are automatically and fully generated directly from the specification. These are enabled because the specification is written with formal methods. Regardless of the automatic proofs, the formal methods increases the quality. Of course testing is still important, because it needs to be examined whether what is described meets what is considered.

From the result we can expect that formal analysis including proofs will be available in the near future. When the proofs are easily available, it will be easier to increase the safety of software by incorporating safety requirements in the formal specifications and proving them.

Acknowledgements
This analysis was held as a part of the cooperative work between RTRI and IFAD. We would like to thank IFAD, especially Kim Sunesen and Peter Gorm Larsen for giving us many comments on the specification and allowing and helping us the use of the prototype integrity checking tool. The tool
was developed under the framework of PROSPER project and we would like also thank PROSPER members.

Bibliography