An Research for Formal Verification of Safety-Critical Software

Weigang Ma[1,2], Xinhong Hei[2]
1. Key Laboratory of Computer Networks and Information Security (Ministry of Education), Xidian University, Xi'an, 710071, China
2. School of Computer Science and Engineering, Xi'an University of Technology, Xi'an, 710071, China

***Abstract*—A verification methodology is presented for railway interlocking system which is regarded as a safety-critical system. The methodology utilizes UML to model the function requirement and LTL to verify the safety requirements of the specification. The device specifications of railway interlocking system are modeled with UML, then translate into FSM. The safety specification is translated LTL and analyzed with NuSMV. We try to show the feasibility of improving the reliability and reducing revalidation efforts when designing and developing a decentralized railway signaling system.**

Keywords- SafetyCritical Software, FSM, LTL, UML

I. INTRODUCTION

Safety-critical systems, which are a sort of software application that its failure could result in loss of life, significant property damage, or damage to the environment ,had a boost in the last few years, But many problems are even more serious when the systems to be controlled are becoming more and more complex. So software correctness may be a very important issue for safety-critical systems.

A railway system may be large enough to make it necessary to divide it into many sections and to assign to each section a controller running its own specific software. A sort of distributed railway system consisting of signal units which indicating whether trains should run or stop, switch units which deciding what way the train should go, and track units which checking the train position and sending this information to control terminals or other interlocking devices belongs to safety-critical system.

there are some solid reasons for implementing an automatic verification process for that kind of development. A kind of object-oriented software architecture has been proposed for distributed railway system [1]. The new architecture avoids interlocking system being developed whenever a new station is to be constructed or revised, but once the prototyping devices are developed, they can be easily deployed in other different stations with slight configurations.

This paper proposes a modeling and verification methodology for the object-oriented software system of a railway interlocking system. The methodology combines UML (Unified modeling Language), a standard of OMG (Object Management Group) for analyzing and designing object-oriented systems [6], and FSM (Finite State Machine) as well as SMV (Symbolic Model Verifier) to analyze and verify the system safety requirement. SMV is a useful tool having a plenty of extensions and being adapting from system analysis, design, to verification [7, 14].

II. BACKGROUND

A. Unified Modeling Language

The Unified Modeling Language (UML) is a collection of semi-formal models for specifying, visualizing, constructing, and documenting models of technical systems and of software systems [8]. Various diagram types is provided and allowing the description of different system viewpoints. And it is very flexible and customizable, because of its extension mechanism. Among the behavioral UML diagrams especially the Sequence charts are suitable for modeling the systems behavior and it describe the way that objects interact externally.

Statecharts describe single cycles inside the system and how instances of classes behave internally [15]. In a complete design they provide a full description of how the system works. The statechart is linked for each object into its box in the Sequence diagram. At any point in the lifetime of this system, there are several same objects perhaps but each object must be in one of its internal states.

UML is a widely accepted modeling standard in industry. It is not allow modeling and evaluating of properties like timeliness, throughput or fault-tolerance without extensions UML[16].

B. NuSMV

All the rules were efficiently verified by NuSMV[12]. NuSMV is used as the checker which is a reimplementation that extends the SMV[13], developed at the Carnegie Mellon University. The FSM of the system of a railway station should be translated into the NuSMV program. The programming language is basically the same language as defined by SMV (the "SMV language").

The SMV language has a different semantics, since it considers that every assignment is executed in parallel.

Table I shows an example of the translation from one language into NuSMV.

TABLE I. THE TRANSLATION FROM ONE LANGUAGE INTO NUSMV.

Original program:	**SMV translation:**

A = .B * D * (C + A);
B = .A * D;
	next(A) := ! B & D & (C | A);
	next(B) := !next(A) & D;

III. DESIGN AND REQUIREMENT OF SYSTEM

A. *Modeling of Interaction of OBJECT*

When a train comes, as mentioned above, all devices must send and receive many messages and make decisions according to the conditions of all devices which are related to it. With these programs of sending and receiving messages and actions of each device predefined, a sequence diagram is designed in Figure 1.

There are three objects in the sequence diagram: Signal Unit, Track Unit and Switch Unit. The sequence diagram illustrates the program of interaction of three kinds of device objects when a train try to pass the switch unit which.

First the track object obtains the message that the train will pass the track unit though the circle of electric in it. And then the track unit sends route request message to signal unit and switch unit, then it waits for return message from signal object and switch object.

Second the signal object will call some methods to determine that if it is safe to allow the signal changing to green. Then it will send the result of the processing to the track object. The switch object will call some methods to determine that if is safe to allow the signal changing the state from current to the state which the train in the track had requested, and then send the result to the track unit.

Third the track unit will call some method to find out that if it is safe to allow the train passes itself to the switch, and send the result to signal unit and switch unit then call some methods to inform the train to control the train. The signal object received the message from track object and changed his state to green allowing the train passing or red forbidding the train passing according to the message received from track unit. The switch object received the message and changed it state according the message. If the message allows the train passing the switch, then the switch changed his state from current to which the train requested and allow the train passing the switch.

Last, the track object, signal object and switch object return to initial state and continue to work. The process of above was illustrated in the sequence diagram in the Figure 1 below.

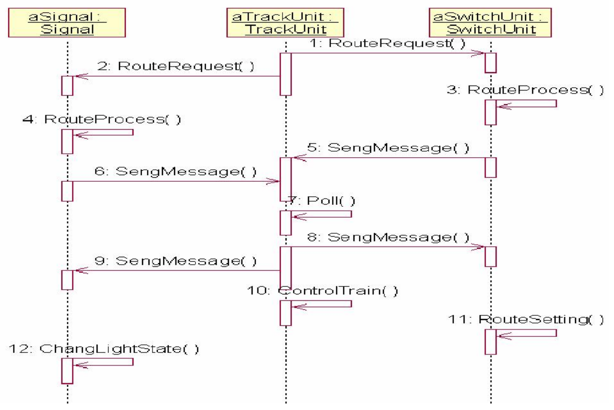

Figure 1. Design of Sequence Diagram

B. *SAFETY REQUIREMENTS*

Our goal is to develop a train control and interlocking system, satisfying the following two safety requirements:

No collision. Two trains must not reside on the same segment.

No derailing. Trains must not derail (by passing an end point of the network or by entering a point from a segment which is not connected with the next segment).The notion of safety can be formalized by defining a predicate which can be used to test whether a state is safe. Here, segments is an auxiliary function giving the segments of a position. Observe that the no-derailing safety requirement above only covers wrong point positions as the cause for derailing, but does not refer to derailing due to excessive speed of trains. However, this cause for derailing can be handled by a completely separate safety-mechanism.

To avoid derailing due to excessive speed, the maximum velocity is calculated as a function of the train type, the number of wagons attached to the train engine, and the actual train position. It is continuously checked whether the actual speed does not exceed the calculated maximum value, otherwise the train control computer issues a warning and may even automatically trigger the brakes.

Obviously, this safety mechanism can be designed and implemented completely independent from the safety mechanisms preventing collisions and derailing due to wrong point positions. Therefore, we do not consider derailing due to excessive speed in the following sections.

IV. MODELING AND ANALYSIS

A. *Transforming UML Statecharts to FSM*

A two-step translation is performed in the proposed approach in order to obtain a set of FSM from the UML statecharts constituting the DRIS specification.

In the first step, FSM is created from each statechart. All guards and actions in statechart is deleted from the transitions and replaced by event labels. This step apparently removes all data dependencies from the transitions. The synchronisation constraints described by the guards and actions on the transitions are introduced into the model by adding one state machine for each variable used in the statechart.

Given that the original UML statecharts are specified using formal notation, the translation into finite-state machines can be performed automatically.

This translation process so far has only been carried out for the DRIS model, but based on the experience obtained; it can easily be extended to handle arbitrary statecharts.

Different models have been created, representing different fault possibilities and buffering capacities.

In the second step, the FSM of a railway station is obtained though composition of each FSM of the elements of the railway station. In this step, each element of railway station has its FSM of behavior himself. The Cartesian product of all elements' FSM of railway station describes the DRIS specification of a railway station.

B. *Translate to NuSMV*

Therefore, we needed a program to automatically translate the original source code into the SMV language. The translator we implemented creates a single SMV input file declaring all the variables and the complete set of assignments defined by the original source code. There is a one to one correspondence between the assignments in the original source code and the SMV input language.

The translator is responsible for the following tasks:

•The assignment, and the operators and, or and not are represented by different symbols in both languages, so the original representation has to be converted according to the SMV language rules.

•The SMV language does not accept variables with a digit as the first character, so the translator always includes an underscore (_) at the beginning of every variable.

•The SMV operator next is attached to any variable at the left-hand side of an assignment.

In doing so, we specify that the next state value of the variable being assigned will be the result of the formula at the right-hand side.

• The translator has to decide how it translates any variable var at the righthand side of an assignment.

It can translate it just as "var" or it can translate it as "next (var)". The former case is used when, starting the translation from the first assignment, var has not already been used at the left-hand side of an assignment. The later case is used otherwise. The last task preserves the semantics of the original source code during its translation into the SMV language. In the original source code, the assignments are executed sequentially from top to bottom.

C. *Checking with NuSMV*

All the rules were efficiently verified. Tab 2 shows the part of safety specification of system. Some errors can be found through the verification by NuSMV, and then modification of the design of UML dialog is performed to debug the error.

Considering that it was a very positive result, we tried the verification of the rules to a larger section. The new experiment was a much more complicated track section, having more inputs, more state variables, and other new LTL specifications. Furthermore, we have created and analyzed the model including timing assumptions, which is needed to verify certain application-specific properties requiring timing constraints.

But it is possible to observe from experiment that the used technique circumvented the state explosion problem.

Here safety check rules associated with the observable behavior of the track is translated a series of LTL formulas in order to check and verification. The set of rules does not completely describe all the safety-related requirements of the system, but helps the feasibility analysis of the use of model checking for that kind of system.

TABLE II. LTL SPECIFICATION FOR THE SAFETY RULE

Rule	LTL Specification
1	G(T1=free & SS1=green→T1=locked)
2	G ¬(SSY=green & SS2=green)
3	G ¬(TRAIN1LOC = TRAIN2LOC)
4	G (TRAIN1LOC in trackSet).

V. CONCLUSION

A kind of methodology is presented to support design and validation of UML diagram and safety specifications in the paper,. Our research work covers to generate FSM model from UML diagram to allow use of existing analysis techniques such as NuSMV. Two key activities are discussed: 1) Generation of individual FSM of the objects and whole system's FSM through product of each object of system which composed the whole system from system and designed the interaction of each object, and 2) Translateing the FSM of the system requirement and translate them into NuSMV, and then verify the safety requirement through NuSMV. It is verified that the methodology can find some errors which may contravene the law of the safety specifications in the process of system analysis and design.
The experience showed that we are able to verify all the rules in relatively short time, with no need for further abstractions, which would demand more manual interference on the process as well as the possibility of inserting errors into the verification. The main problem that we could have faced is the state explosion problem.

REFERENCES

[1] Hei X. , Mochizuki H., Takahashi S. and Nakamura H. ,"Modeling a distributed railway interlocking system with object-oriented Petri-net," 10th International Conference on computer system design and operation in the railway and other transit system, Prague, Czech Republic, 309-318 (2006).

[2] Doron A, Peled. Software Reliability Methods. Springer (2001).

[3] John D. Musa, Software Reliability Engineering: More Reliable Software Faster and Cheaper, Autorhouse, second edition (2004).

[4] W.T. Tsai, R. Mojdehbakhsh, F. Zhu, "Ensuring System and Software Reliability in Safety-Critical Systems," 1998 IEEE Workshop on Application - Specific Software Engineering and Technology, p.48(1998).

[5] John C, Knight, "Safety Critical Systems: Challenges and Directions," Proceedings of the 24th International Conference on Software Engineering

[6] Mark Priestley, Practical Object-Oriented Design with UML, McGraw Hill Higher Education (2003).

[7] R. Malik., R. Muhlfeld, "A Case Study in Verification of UML Statecharts: the PROFIsafe Protocol," Journal of Universal Computer Science, vol. 9, no. 2 (2003), 138-151.

[8] Object Management Group. Unified Modeling Language Specification v.2.0. www.uml.org, September 2003.

[9] Anne E. Haxthausen and Jan Peleska, "Formal Development and Verification of a Distributed Railway Control System," IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 26, NO. 8, AUGUST 2000, 687-701.

[10] X. Hei, S. Takahashi, H. Nakamura, M. Fukuda, K. Iwata and K. Sato, Improving reliability of railway interlocking system with component-based technology, Journal of Reliability Engineering Association of Japan, 28(8), 2006, 557-568.

[11] Alessandro Giua, Carla Seatzu, “Modeling and Supervisory Control of Railway Networks Using Petri Nets,” IEEE TRANSACTIONS ON AUTOMATION SCIENCE AND ENGINEERING, VOL. 5, NO. 3, JULY 2008, 431-445.

[12] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri, “NuSMV: a new symbolic model checker,” International Journal on Software Tools for Technology Transfer, 2(4):410–425, 2000.

[13] K. McMillan, “Symbolic Model Checking: An Approach to the State Explosian Problem,” Kluwer Academic Publishers, Norwell, MA, USA, 1993.

[14] Nelson Guimar˜aes Ferreira and Paulo S´ergio Muniz Silva, “Automatic Verification of Safety Rules for a Subway Control Software,” Electronic Notes in Theoretical Computer Science 130 (2005) 323–343.

[15] D. Harel and M. Politi, Modeling Reactive Systems with Statecharts: The StateMate Approach, Wiley, New York, 1998.

[16] Object Management Group, UML profile for schedulability, performance, and time, www.uml.org, March 2002.