Automated Test Approach Based on All Paths Covered Optimal Algorithm and Sequence Priority Selected Algorithm

Wei Zheng, Ci Liang, Rui Wang, and Weijie Kong

***Abstract*—A timely and complete test is an important factor to assure the functionality and safety of a railway signal system before it is put into service. With the development of rail transportation in China, the traditional semiautomatic test methods cannot satisfy the timely and complete test requirements any longer. This paper proposes an automated model-based test method. First, colored Petri nets are used as a formal language to describe the system specification. Second, the all paths covered optimal algorithm and the sequence priority selected algorithm are proposed to generate the test cases and sequences automatically. Third, taking a typical radio block center (RBC) handover scenario as an example, the generated test cases and sequences are applied into the RBC functionality test platform. The testing result validated the feasibility and efficiency of the proposed automated test method. Compared with the random-walk-based test sequence generation algorithm, the repeatability rate of the generated test sequences is reduced by 46%. The test sequences can cover all the generated test cases, and the cases can cover all the related criteria in the function requirements specification of the Chinese Train Control System Level 3.**

***Index Terms*—All paths covered optimal (APCO) algorithm, automated testing, colored Petri net (CPN), rail transportation, sequence priority selected (SPS) algorithm.**

I. Introduction

As the safety-critical system, the Chinese Train Control System Level 3 (CTCS-3) is capable of ensuring the safe and efficient operation for a train with high speed and density. Before the system comes into service, it is essential to execute a series of tests, including laboratory testing, field testing, integrated testing, and interoperability testing to verify the consistence of the system to the requirement specification [1]. A timely and complete test can significantly contribute to finding the drawbacks in the system design and to assuring the appropriate functional behaviors of the system. Taking the "7.23" accident in China as an example, there existed flaws in the fault-tolerant design of the acquisition drive unit, with the result that the Train Control Center (TCC) misunderstood the train occupancy information. One of the reasons was that the software functionalities of the TCC were not fully tested because there were not enough test time and efficient test methods for the signal system before it was put into operation.

Most current test methods in China are semiautomatic, and the test cases and test sequences that were manually generated have the weakness of a heavy workload, low efficiency, and high demand for professional expertise. As an agile and systematic method, model-based testing, which aims to automate the testing process, has attracted much more attention in recent years [2]. These methods need to model the system using a formal description technique, such as the timed I/O automata [3], the UPPAAL timed automata (UTA) [4], [5], the unified modeling language (UML) [6]–[8], the finite-state automata [8], [9], and the colored Petri net (CPN) [10], [11].

In 2004, Hessel and Pettersson proposed a test case generation algorithm based on the UTA, and the amount of time and space saved by the algorithm in the experiments was 50% to 58% and 30% to 35%, respectively [5]. However, this method could not describe the concurrent behaviors in CTCS-3, and the generated test cases are a relatively high level of abstraction, which means the test process could not be truly automated. Samuel and Joseph proposed to generate the test sequences with UML 2.0 sequence diagrams [6]. This method is fully automated, and the test sequences can be used to check the correctness of the system under test. However, the drawback of the UML is that there is no definition of any executable metadata model and no hardware support. Lee *et al.* [12] put forward the automated conformance test method for the communication security protocol of a train control system based on an I/O finite-state machine (I/O FSM). In addition, they achieved the conformance test sequences with the automatic generation of tools for the communication security protocol of the train control system with the C++ language [13], which enables the automatic test to be practical. However, the method based on the FSM requires rigorous mathematical assumptions and costs a large amount of computation time. Watanabe and Kudoh presented the automatic generation of test suites based on CPNs [10], which can test the interaction parameters of concurrent systems and reduce the computation costs compared with the FSM-based test. Their CP tree method requires the generation of a reachability tree from the CPN model, and then, test sequences can be generated by traversing through arcs and nodes from the root to the leaf nodes of the CP tree.

Manuscript received June 16, 2013; revised December 28, 2013; accepted April 14, 2014. Date of publication May 21, 2014; date of current version December 1, 2014. This work was supported in part by the International Science and Technology Cooperation Program of China Project under Grant 2012DFG81600. The Associate Editor for this paper was H. Dong.

W. Zheng, R. Wang, and W. Kong are with the National Engineering Research Center of Rail Transportation Operation and Control System, Beijing Jiaotong University, Beijing 100044, China (e-mail: wzheng1@bjtu.du.cn; 11120306@bjtu.edu.cn; 11120273@bjtu.edu.cn).

C. Liang is with CASCO Signal Ltd., Beijing 100045, China (e-mail: liangci321@126.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TITS.2014.2320552

1524-9050 © 2014 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

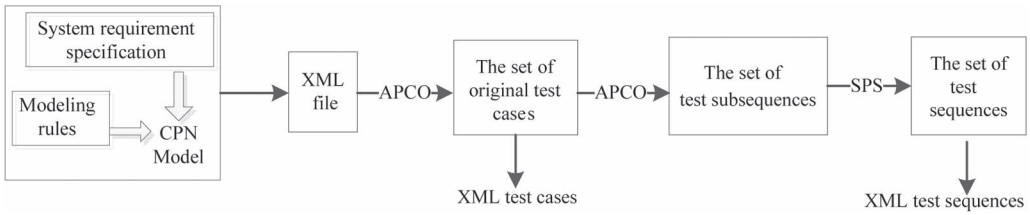

Fig. 1. Frame diagram of the test generation method.

Chen *et al.* [14] defined a simple path as one path having no loops or concurrence, and then, a modified depth-first searching (DFS) algorithm was proposed to generate a set of simple paths. However, the proposed technique will bring about a lot of repeated and invalid test cases. Farooq *et al.* [2] proposed an efficient random-walk-based test sequence generation (RW-TSG), which adapted the random selection process to the predefined semantic for the CPN to avoid any invalid paths. However, this algorithm needs much iteration to reach the complete coverage, and the results are not robust because of its randomness.

Taking all these aspects into consideration, two new algorithms are proposed in this paper to deal with the problems of inefficiency and repetition. The path optimization method contains two algorithms, i.e., the all paths covered optimal (APCO) algorithm and the sequence priority selected (SPS) algorithm. They are applied to generate Extensible Markup Language (XML) test cases and XML test sequences, respectively. These test sequences are totally feasible for practical test executions. In addition, the repeatability rate of the generated test sequences is reduced much compared with the available RW-TSG and DFS, and the test sequences can cover all the related criteria in the CTCS-3 function requirements specification (FRS). Furthermore, a scenario of the radio block center (RBC) handover in CTCS-3 is taken as an instance to be modeled, and the corresponding tests are carried out on the RBC test platform, which verified the efficiency of the proposed method.

The rest of this paper is organized as follows. In Section II, the definitions related to the system model, test cases, and test sequences are described. The APCO algorithm, the SPS algorithm, and the automated generation method based on the aforementioned algorithms are elaborated. In Section III, the proposed method is applied into an RBC handover scenario and validated on the RBC test platform. Some concluding remarks are presented in Section IV, with directions for future research suggested as well.

II. Automated Test Generation

The automated test method using the CPN model can be depicted in Fig. 1, and it is mainly divided into three stages, including modeling, test case generation, and test sequence generation. In the first stage, according to the system requirement specification and the modeling rules, the CPN model is built for the object under test, and the XML file and the reachable graph of the model are obtained. Then, on the basis of the CPN model with the XML format, the proposed APCO algorithm is applied to generate the set of original test cases in the second stage. In the third stage, a test subsequence set can be generated through concatenating the test cases by using the information in the test cases and the APCO algorithm. Next, the test subsequences are strung into test sequences by using the proposed SPS algorithm. Finally, the set of original test cases and the test sequences can be transformed into files with the XML format, and the XML-format test sequences can be directly used in a practical test process.

A. Definitions in CPN Modeling

Combined with an advanced programming language and the hierarchical structure of modeling, a CPN is able to accomplish the model for a large-scale system and is particularly suitable for a concurrent, asynchronous, distributed, parallel, or indeterminate system [11]. To facilitate the elaboration of the following sections, the necessary concepts are defined in detail on the basis of the CPN theory [10], [15].

Definition 1: The test case is a 6-tuple on the basis of the CPN models, i.e., $\mathrm{TC} = \{S_{SC}, S_{EC}, I_{IC}, O_{IC}, I_{MC}, O_{MC}\}$, where S_{SC} is the subset of the initial state of the test case containing the initial state nodes and the conditions; S_{EC} is the subset of the end state containing the end state nodes and the conditions; I_{IC} is the subset of the input interface of the test case; O_{IC} is the subset of the output interface; I_{MC} is the subset of the output data of the test case, $I_{MC} \subseteq A$; and O_{MC} is the subset of the output data, $O_{MC} \subseteq A$.

Definition 2: The test subsequence is a 3-tuple on the basis of the CPN models, i.e., $\mathrm{SS} = \{\mathrm{TC}, S_{\mathrm{SS}}, S_{\mathrm{ES}}\}$, where TC is a test case set, and tc_i is a test case, $\mathrm{tc}_i \in \mathrm{TC}$; S_{SS} contains the initial conditions and is the initial state of the test subsequence; and S_{ES} contains the end conditions and is the end state of the test subsequence. TC, S_{SS}, and S_{ES} are nonempty sets.

Definition 3: The test sequence is a 4-tuple on the basis of the CPN models, i.e., $\mathrm{TS} = \{\mathrm{SS}, R, S_{\mathrm{STS}}, S_{\mathrm{ETS}}\}$, where SS is a test subsequence set; ss_i is a test subsequence, $\mathrm{ss}_i \in \mathrm{SS}$; R is a subsequence weight set, where element r is the weight of the subsequence corresponding to each element ss_i in SS; S_{STS} is the initial state of the test sequence containing the initial conditions; S_{ETS} is the end state of the test subsequence containing the end conditions.

Definition 4: Coverage criteria: Regarding a variety of coverage criteria described in [16], proper coverage criteria are proposed.

- *All-Node Coverage Criteria.* If all the generated test cases are executed, they must traverse all the nodes in the CPN model, which means traversing all the places or transitions and all the reachable states. When the test sequences are generated, a test case is seen as a node. If all the generated test sequences are executed for one time, they must ensure that all the test cases are executed at least once.

```
Function Depth Search(Node, Passed Nodes)
    Extend Node, gain Sub Nodes array;
        For i=1 to the length of Sub Nodes
            Passed Nodes of Sub Nodes[i] =Passed Nodes+Sub Nodes [i];
        End for;
        Initialized Output=False and Deep Output=False;
        For i=1 to the length of Sub Nodes
            If Sub Nodes[i] =G
                Output Passed Nodes of Sub Nodes[i];
                And Output=True;
            Else if Sub Nodes [i] is not in Disabled Nodes or Passed Nodes
                Deep Output=Depth Search (Sub Nodes [i], Passed Nodes
                of Sub Nodes [i]);
        End for;
        Result=Output or Deep Output;
        If Result=False
            Put Node into Disabled Nodes;
        Return Result;
End function;
```

Fig. 2. Software description of the APCO algorithm.

- *All-Branch Coverage Criteria*. If the generated test cases are executed, they must traverse all the arcs in the CPN model, as well as the transfer conditions between all adjacent reachable states. When the generated test sequences are executed for one time, they must ensure the coverage of all the transfer conditions of the test cases.

B. Algorithm Description

APCO Algorithm: The traditional DFS algorithm has the following disadvantages [17]. First, it does not consider the situation that there may be nodes of the previous layers in the next layer, which will easily lead to an infinite loop. Second, there exists a blindness problem that the algorithm cannot skip the invalid node (no path exists between this node and the target node) when it appears.

In terms of the aforementioned reasons, the APCO algorithm has improved the DFS, and the adjacency list is used in the APCO algorithm. The APCO algorithm searches all the possible paths between any node in the set of the initial state nodes and the corresponding node in the set of the end state nodes, and then, it removes the redundant paths, according to the test optimization strategy, to get the most simplified path set that can cover all paths. The directed graph can be described with the adjacent table or the adjacent matrix. For the adjacent matrix of a directed graph with n nodes, every node needs to be scanned in the adjacent matrix, and its time complexity is $O(n^2)$. However, for the adjacent table of the same directed graph, scanning each edge can reach the same goal, and its time complexity is only $O(n)$. Therefore, the computational complexity is greatly reduced [17]. The core concept of the APCO algorithm is shown in Fig. 2, and the improvements of the APCO algorithm are as follows: A public dynamic array named Disabled Nodes is created to record all the state nodes that do not conform to the rules, including the invalid nodes. Next, the dynamic array named Passed Nodes is created for every state node to record all the state nodes between the initial state node and this node itself.

SPS Algorithm: The SPS algorithm is described in Fig. 3. Because each subsequence has its own weight, the SPS algorithm first finds out the subsequence with maximum weight r_{maxi} in each scenario, then strings them to generate a sequence

```
Function MaxUnit (data1, data2)
    For i=0 to the total number of scenarios
        data1[i] =subsequence number of scenario i;
        data2[i] =subsequence weight of scenario i;
        For j=0 to data1[i].length
            Find max{data2 [i]};id=j;
            Return ret=data1[i].p[j];
            Delete data1 [i].p[j];
            Output scenario i to max{data2 [i]}and data1 [i].p [id];
        End for
        Call Fun (Sequence, Scenario, Weight)
        Rearrange Scenario, Weight from big to small;
    End for
    Return Scenario, Weight;
    Output Sequence;
End function
```

Fig. 3. Software description of the SPS algorithm.

from the beginning scenario to the end scenario according to the order of the scenarios. Next, it removes maximum weight r_{maxi} in R_i to get R_i^1, finds out the corresponding subsequences with maximum weight r^1_{maxi} in R_i^1 in each scenario, strings them to get a sequence according to the order of scenarios, and removes maximum weight r^1_{maxi} in R_i^1 to get R_2^i. By the parity of the reasoning, all the subsequences are strung to the sequences. Thus, the advantages are that all the cases are contained, fewer sequences contain as much cases as possible, and the repetition rate is reduced effectively.

C. Test Generation Method

Generation of Test Cases: According to Definition 1, the test case generation needs the following steps: 1) the generation of S_{SC} and S_{EC}; 2) the generation of I_{IC} and O_{IC}; and 3) the generation of I_{MC} and O_{MC}.

1) The generation of S_{SC} and S_{EC}. The information of the input and output ports is reflected in the places with the properties of the input and the output. The input and output ports of the same scenario correspond with each other. In the CPN model, there are "In" and "Out" marks in the input and output places of a scenario, and these marks also exist in the corresponding XML file. The places with the "In" property are searched in the XML file, and they correspond to the specified initial nodes and conditions. Analogously, the places with the "Out" property are searched in the XML file, and they correspond to the specified end nodes and conditions. Then, the repetitive nodes and conditions are removed to get the most concise S_{SC} and S_{EC}.
2) The generation of I_{IC} and O_{IC}. In order to get the information of I_{IC} and O_{IC}, all the paths from every initial state node in S_{SC} to the corresponding end state node in S_{EC} are found out by using the APCO algorithm. The generated paths are those with the input and output interfaces in pairs to avoid the redundancy of the paths and the errors.
3) The generation of I_{MC} and O_{MC}. First, the variable names of the output arcs that begin with the initial places must be found out. The variable names are the input message IDs. Next, the relevant variables and their values are added into I_{MC} according to the message ID. Then,

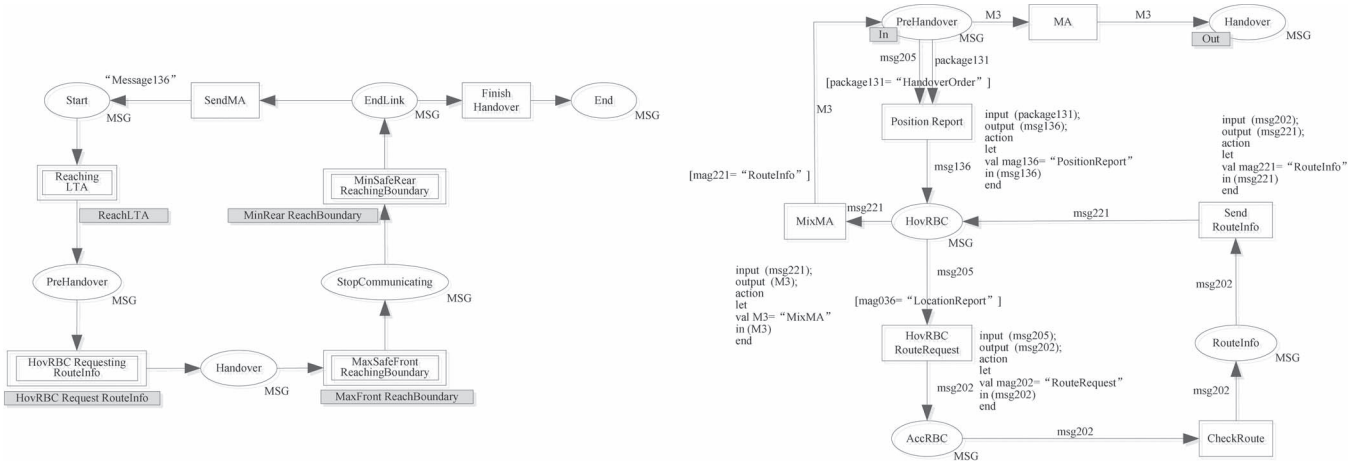

Fig. 4. Top-layer model and sublayer model of "HovRBC Requesting Route Info."

I_{MC} is obtained. Similarly, the variable names of the input arcs with their terminal pointing to the end places must be found out. The variable names are the output message IDs. Next, the relevant variables and their values are added into O_{MC} based on the message ID. Then, O_{MC} is obtained. S_{SC}, S_{EC}, I_{IC}, O_{IC}, I_{MC}, and O_{MC} can be obtained via the aforementioned three steps, and the test case set with the executable data is generated.

Generation of Test Subsequences: According to the description of Definition 2 and the generated test case set that contains the detailed information of each test case, the method of test subsequence generation is elaborated as follows. $S_{\rm SS}$ is the union set of the test case's initial state set S_{SC}, i.e., $S_{\rm SS} = S_{SC1} \bigcup S_{SC2} \bigcup \cdots \bigcup S_{SCn}$, corresponding to the initial cases in TC. S_{EC} is the union set of the test case's end state set S_{SC}, i.e., $S_{\rm ES} = S_{EC1} \bigcup S_{EC2} \bigcup \cdots \bigcup S_{ECn}$, corresponding to the end cases in TC. Moreover, S_{SCi} corresponds to S_{ECi}. In order to get the subsequences, all the paths from every initial state in the $S_{\rm SS}$ to the corresponding end state in the $S_{\rm ES}$ are found out by the APCO algorithm. Because there is no unreachable state in the state space, it does not miss any case.

Generation of Test Sequences: According to Definition 3 and the detailed information of test subsequences in the subsequence set, the detailed steps of generating the test sequences are as follows.

- The initial and ending scenarios are appointed in the second layer of the CPN model to be described in the next section, and the generated subsequences are divided into different subsequence sets $\rm SS_i$ according to the different scenarios.
- In Definition 3, R is the subsequence weight set, and element r in it is the subsequence weight. The value of r is the number of cases contained in the corresponding subsequence. According to the division of $\rm SS_i$, R is divided into R_i, and the subsequences are strung to sequences by the SPS algorithm.

The test cases and sequences generated by the aforementioned approach have to meet the coverage criteria in Definition 4, and XML is the final form of the generated cases and sequences. The automated generation software is developed in C++. The software can implement the aforementioned algorithms to generate the test cases and sequences.

III. Application and Analysis of Test Method

RBC, as key trackside equipment in CTCS-3, can generate the movement authority (MA) and other control commands according to the information coming from other subsystems [18]; hence, it should satisfy the functional requirements in the CTCS-3 specifications. In this section, a typical RBC handover scenario will be taken as an example and modeled using the CPN, and then, the corresponding test cases and sequences will be generated by using the proposed path optimization algorithm. Compared with the RW-TSG and traditional DFS algorithms, the numbers of test cases and sequences, as well as the coverage, will be seen as the indicators to show the superiority of the proposed algorithms.

A. Functionality of RBC and Interfaces

The main functionalities of the RBC include: 1) receiving the position and train data from the onboard equipment through the global system for mobile communications for railways (GSM-R) wireless communication system; 2) generating the MA according to the information from the balise, the track circuit, the temporary speed restriction server, and the interlocking system; 3) transmitting the MA to the onboard equipment, which will calculate the location–speed curve to protect the train safety operation [18].

In CTCS-3, the RBC is connected with the centralized traffic control (CTC) system, the computer-based interlocking (CBI) system, the temporary speed restriction server (TSRS), and the neighbor RBC (NRBC) by different specific safety data communication "Ethernet," and it is connected with the onboard equipment by GSM-R.

The dispatcher may send the normal operation command, the emergency stop command, and some necessary texts from the CTC to the RBC through the Ethernet. At the same time, the RBC sends the train data (the train number, the train identification, the location, the speed and length of the train, etc.), the interface warning and diagnosis data, and the unexecuted command report to the CTC. The packages from the CBI system to the RBC are periodic and include the route type, the route status, the

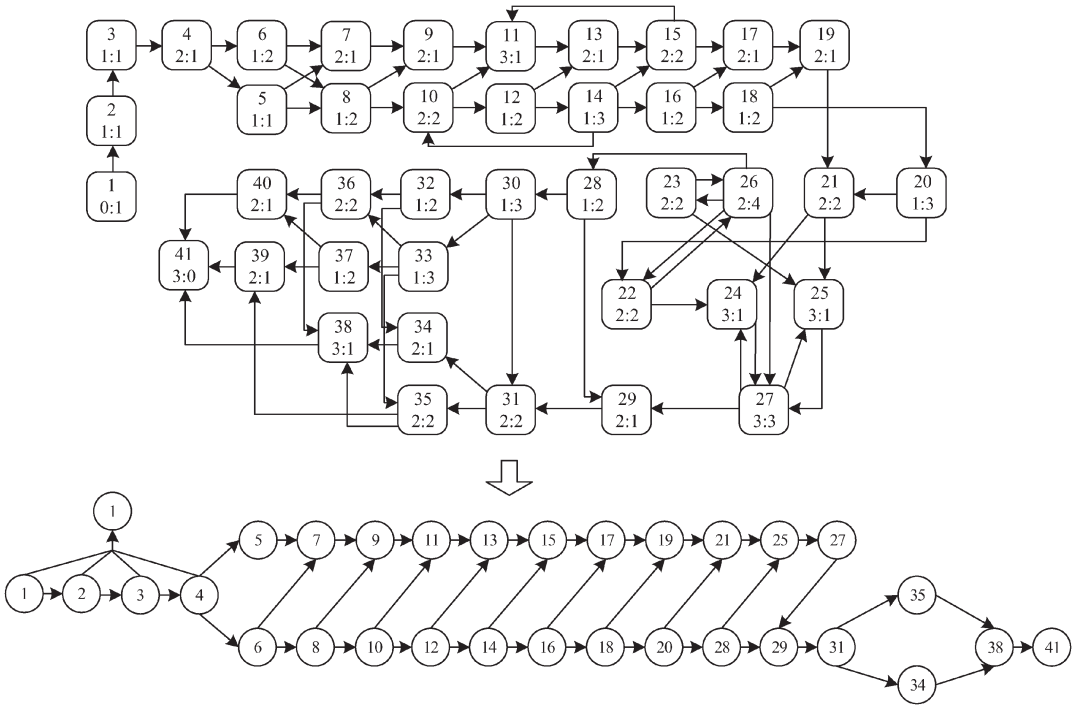

Fig. 5. Reachable states and simplified states.

route identification, the protection section, etc. The information from the RBC to the CBI system includes the train status, the MA status, the location, the speed and length of the train, etc. The dispatcher may set or cancel the TSR command from the TSRS to the RBC, and the RBC will send all the TSR status to the TSRS.

In general, in order to control more trains, several RBCs are configured to cover the operation track. For the two RBCs at the ends of the track, they only have one NRBC, whereas for the RBC in between the tracks, it will have two NRBCs. The control zones of two NRBCs share the overlap zone so that, when a train travels from one RBC directorial area to the adjacent RBC directorial area, the train has a chance to connect with the adjacent RBC and exchange the necessary information before it has to be disconnected with the first one. This process is called the RBC handover action, which has been selected as the case scenario and will be specified in the next section.

B. Application to RBC Handover Scenario

First, based on the modeling rules in Section II and the requirements in the specifications [18], the CPN model of the RBC handover scenario is established hierarchically, and the scenario is divided into four subscenarios: 1) Reaching LTA (Level Transition Announcement); 2) HovRBC Requesting Route Info; 3) Max Safe Front Reaching Boundary; and 4) Min Safe Rear Reaching Boundary. The top-layer model and the sublayer model of "HovRBC Requesting Route Info" are illustrated in Fig. 4. The handing-over RBC requests the route information from the accepting RBC and, then, sends the MA, which extends to the area of the accepting RBC, to the onboard equipment. The sublayer model of the rest of the three subscenarios is not detailed here.

```
<?xml version="1.0" encoding="UTF-8"?>
  <TestCasesFile>
    <version>1.0</version>
    <testcase_001_description>
        <testcase_number>001</testcase_number>
        <testcase_property>normal</testcase_property>
        <IO_Type>I/O</IO_Type>
        <Interface>JRU</Interface>
        <StartingConditions>LocationReport</
StartingConditions>
        <TestDescription>
            <Receive>Message136</Receive>
            <Message136>
                NID_MESSAGE=136
                L_MESSAGE=24
                T_TRAIN=137827
                NID_ENGINE=536586
                NID_PACKET=0
                L_PACKET=114
                Q_SCALE=1
                NID_LRBG=16777215
                D_LRBG=32767
                Q_DIRLRBG=2
                Q_DLRBG=2
                L_DOUBTOVER=32767
                L_DOUBTUNDER=32767
                Q_LENGTH=0
                V_TRAIN=0
                Q_DIRTRAIN=2
                M_MODE=0
                M_LEVEL=3
            </Message136>
            <Send>package131</Send>
        </TestDescription>
        <EndingConditions>HandoverOrder</EndingConditions>
    </testcase_001_description>
    <testcase_002_description>
        ……
    </testcase_002_description>
    ……
</TestCasesFile>
```

Fig. 6. XML instance of test case 001.

TABLE I
RESULTS OF THE GENERATED TEST CASES AND SEQUENCES

Scenario	#Case Generated	#Place Covered (#Total Place)	#Arc Covered (#Total Arc)	#State Node Covered (#Total State Node)	Using the SPS		Using the RW-TSG		Using the Traditional DFS	
					#Test Sequences Generated	#Coverage For Test Cases (%)	#Test Squences Generated	#Coverage For Test Cases (%)	#Test Squences Generated	#Coverage For Test Cases (%)
Reaching LTA	2	5(5)	17(17)	7(7)						
HovRBC Requesting Route Info	7	5(5)	13(13)	14(14)						
Max Safe Front Reaching Boundary	3	4(4)	10(10)	12(12)	7	100%	13	100%	28	92.9%
Min Safe Rear Reaching Boundary	2	5(5)	9(9)	13(13)						

A very important reason for using the CPN as the modeling language is that we can get the trace of the reachable states of the CPN model. In addition, it is the trace of the reachable states that can help generate the test cases. Fig. 5 shows the generated trace of the reachable states of the model, which consists of 41 reachable states, including some redundant states. They are simplified into 30 states, as shown at the bottom of the figure. Nodes 1, 2, 3, and 4 should be passed through at the beginning; thus, these four nodes are combined with Node 1.

With the APCO algorithm, the test cases generated for the four scenarios can be seen as follows:

1) / 1 6 7 / 1 5 7 /;
2) / 6 8 10 12 14 16 18 19 / 6 8 10 12 14 16 17 19 / 6 8 10 12 14 15 17 19 / 6 8 10 12 13 15 17 19 /6 8 10 11 13 15 17 19 / 6 8 9 11 13 15 17 19 /6 7 9 11 13 15 17 19 /;
3) / 18 20 28 29 / 18 20 21 25 27 29 /18 19 21 25 27 29 /;
4) / 29 31 35 38 41 / 29 31 34 38 41 /.

These aforementioned numbers represent the state nodes. The nodes between a pair of "/" are the state nodes that one test case contains. It is obvious that the numbers of the test cases generated for the four scenarios are 2, 7, 3, and 2, respectively. Taking the system requirements specifications of CTCS-3 [18], CTCS-3 FT 136-v200, and CTCS-3 FT 142-v200 [19] into consideration, the 14 test cases that are generated by using our APCO optimization algorithm can cover all the functional requirements on the RBC handover scenario in the aforementioned specifications. The generated XML instance of the test case 001 is shown in Fig. 6, including the case version, the case number, the property (normal/fault), the test start and end condition, the specific values of the message variables required in the test, etc.

In order to generate the test sequences, the generated test cases are numbered. Based on the SPS algorithm aforementioned in this paper, the test sequences are generated as follows by dividing the subsequences: / 1 5 13 / 1 4 11 13 / 1 3 10 13 / 2 6 12 14 / 2 7 14 / 2 8 14 / 2 9 14/. Compared with other excellent RW-TSG [2] and traditional DFS algorithms [17], the generation results can be shown in Table I. It is clear that the number of test sequences generated with the SPS algorithm is reduced to seven, and the test sequences can cover all the generated test cases. However, the number of test sequences generated by the RW-TSG and DFS algorithms is up to 13 and 28, respectively.

```
<?xml version="1.0" encoding="UTF-8"?>
  <TestSequenceFile>
    <version>1.0</version>
    <testsequence_001_file>
      <testsequence_001_description>
          <testsequence_number>001</testsequence_number>
          <testsequence_property>normal</testsequence_property>
          <NumofTestCases>3</NumofTestCases>
      </testsequence_001_description>
      <testcase_file>
          <name>testcases.xml</name>
          <version>1.0</version>
      </testcase_file>
      <StartingConditions>
         MsgSendingLocation=70374
      </StartingConditions>
      <ExecutionStepsDescription>
         <Step1>
           <StepNum>1</StepNum>
           <testcase_number>001</testcase_number>
         </Step1>
         <Step2>
           <StepNum>2</StepNum>
           <testcase_number>005</testcase_number>
         </Step2>
         <Step3>
           <StepNum>3</StepNum>
           <testcase_number>013</testcase_number>
         </Step3>
       </ExecutionStepsDescription>
       <EndingConditions>
          ExpectingMsg=M24
       </EndingConditions>
    </testsequence_001_file>
</TestSequenceFile>
```

Fig. 7. XML instance of sequence 001.

Fig. 7 shows the generated XML instance of the test sequence 001, i.e., /1 5 13/, including the sequence version, the sequence number, the property (normal/fault), the case number, the case information, the start condition, the execution step, the end condition, etc.

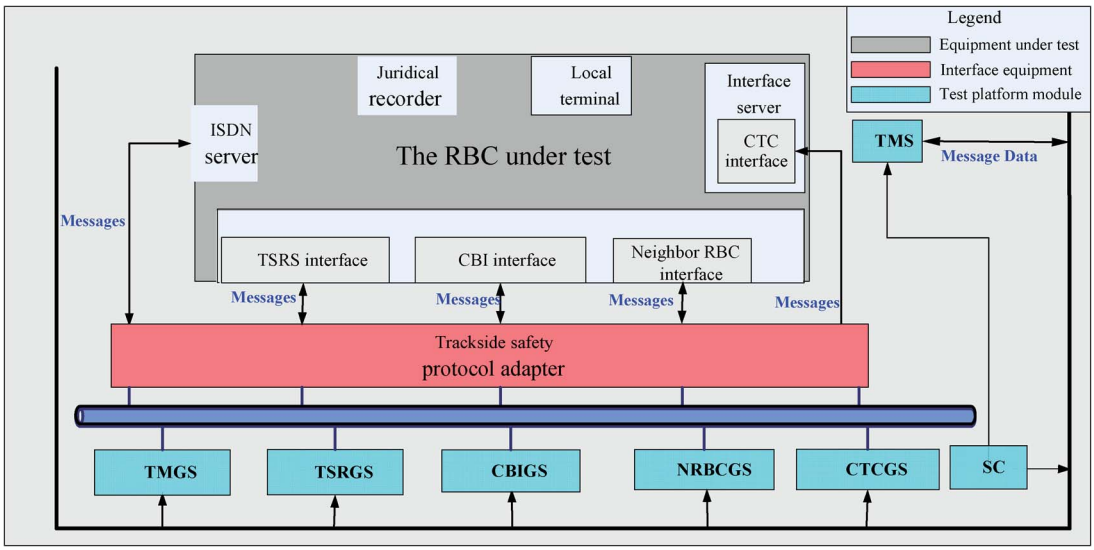

Fig. 8. Schematic of the RBC test platform.

C. Validation on RBC Test Platform

In order to verify the practicality and feasibility of the test automated generation method, an RBC functionality test platform is developed, in which the generated test cases and sequences are injected to test whether the RBC can match all the functional requirements of the CTCS-3 specification.

The schematic of the RBC test platform is shown in Fig. 8. This RBC test platform is employed to test the accuracy of the RBC systems' behavioral functionality and consists of seven subsystems that are the scenario controller (SC), the centralized traffic control generation simulator (CTCGS), the train message generation simulator (TMGS), the computer-based interlocking generation simulator (CBIGS), the NRBC generation simulator (NRBCGS), the temporary speed restriction generation simulator (TSRGS), and the train movement simulator (TMS). The RBC test platform takes the data-driven method where the key subsystems in CTCS-3 are simulated to serve for the RBC under test, i.e., five subsystems (except the SC and the TMS) only have message interaction with the RBC under test, and six subsystems (except the SC) have no communication with each other. The data necessary to these six subsystems during the testing process come from the SC, and the SC can control the whole test process. Due to a large quantity of message data that these six subsystems need, all message data are stored in a database, and the SC gets and sends the data respectively to the six subsystems. This will be prone to making mistakes and bringing troubles to the management of the test database. Thus, the XML test cases and sequences generated in this paper are necessary. With the XML test cases and sequences, the database is unnecessary, and the SC can directly read the XML files to get the test data. Moreover, it is convenient for the management and transplanting.

The subsystems that have direct relationships with the RBC handover scenario are the NRBCGS and the TMGS.

The brief introduction to the functions of these three subsystems is as follows.

- NRBCGS: It gets the message data and control commands from the SC and accomplishes the functional test of the RBC handover. When testing the handing over function of the RBC, the NRBCGS is the corresponding "accepting RBC"; when testing the accepting function of the RBC, the NRBCGS is the corresponding "handing over RBC."
- TMGS: It gets the message data and control commands from the SC, simulates the function of the onboard equipment, and communicates by GSM-R with the RBC under test.
- SC: It can control six subsystems, i.e., the NRBCGS, the CTCGS, the TSRGS, the TMS, the CBIGS, and the TMGS. As the control module in the test, the SC takes charge of controlling the start and end of a test, supervising every stage in the test, sending the configuration and message data to other subsystems, and managing and supervising the operation of the testing platform. The control interface screen of the SC is shown in Fig. 9.

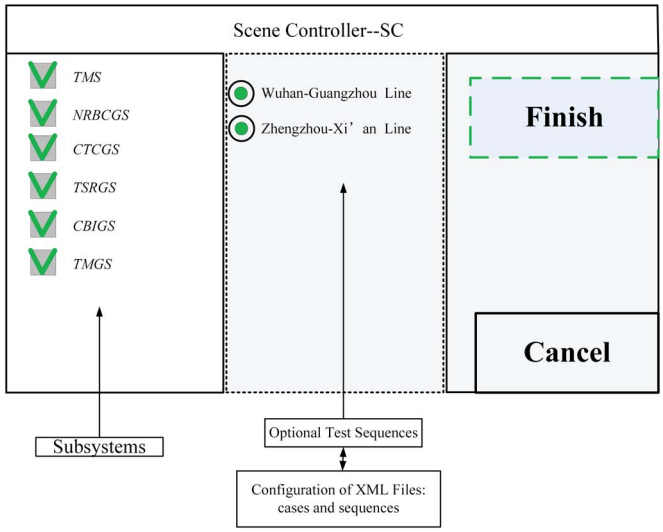

Fig. 9. Control interface screen of the SC.

During the test process, a lot of sequences are needed, and all the test data of the subsystems should be stored in the database in advance, which may be prone to cause some mistakes and inconvenience to the management. The XML files provide an

TABLE II
SCHEMATIC OF THE RBC DATA-DRIVEN TEST

Data_ID	TestSequence_ID	Tsd_Step	Train_ID	Time	Abs_Time	Distance	Event	BitFlag	Direction	Msg_ID	MsgData_Bit
140	1	65535	536586	2013/5/10 10:57:06	91302	9570	Running	1	NRBCGS<-RBC	201	C9001F023C000208300A023
141	1	999	536586	2013/5/10 10:57:07	91333	9598	Running	2	NRBCGS->RBC	205	CD0523C000420C02A3C0C40
142	1	65535	536586	2013/5/10 10:57:07	91333	9616	Running	1	NRBCGS<-RBC	202	CA0018023C000208300A023
143	1	999	536586	2013/5/10 10:57:08	91384	9692	Running	2	NRBCGS->RBC	205	CD0523C000420C02A3C0C40
144	1	7	536586	2013/5/10 10:57:08	91384	9701	Running	2	NRBCGS->RBC	221	DD4763C000420C02A3C0C40
145	1	65535	536586	2013/5/10 10:57:09	91402	9781	Running	1	NRBCGS<-RBC	203	CB0010023C000208300A023
146	1	8	536586	2013/5/10 10:57:09	91402	9831	Running	2	NRBCGS->RBC	221	DD4763C000420C02A3C0C40
147	1	65535	536586	2013/5/10 10:57:09	91402	9831	Running	1	NRBCGS<-RBC	205	CD0014023C000208300A023
307	1	65535	536586	2013/5/10 10:58:31	98802	15898	Running	1	NRBCGS<-RBC	202	CA0018023C000208300A023
308	1	999	536586	2013/5/10 10:58:31	98802	15907	Running	2	NRBCGS->RBC	205	CD0523C000420C02A3C0C40
309	1	67	536586	2013/5/10 10:58:32	98852	15907	Running	2	NRBCGS->RBC	221	DD4763C000420C02A3C0C40
310	1	65535	536586	2013/5/10 10:58:32	98852	15907	Running	1	NRBCGS<-RBC	205	CD0014023C000208300A023
311	1	68	536586	2013/5/10 10:58:33	98903	15916	Running	2	NRBCGS->RBC	221	DD4763C000420C02A3C0C40
312	1	65535	536586	2013/5/10 10:58:33	98903	15925	Running	1	NRBCGS<-RBC	205	CD0014023C000208300A023
313	1	69	536586	2013/5/10 10:58:34	99002	15925	Running	2	NRBCGS->RBC	222	DE0423C000420C02A3C0C40
Data_ID	TestSequence_I	Tsd_Step	Train_ID	Time	Abs_Time	Distance	Event	BitFlag	Direction	Msg_I	MsgData_Bit
150	1	999	536586	2013/5/10 10:46:20	26720	10083	Running	2	TMGS->RBC	136	8800180000579F083006000
151	1	65535	536586	2013/5/10 10:46:21	26750	10113	Running	1	TMGS<-RBC	24	180010000140CA018F071D4
152	1	999	536586	2013/5/10 10:46:23	27039	10402	Running	2	TMGS->RBC	132	840018000058DE083006000
153	1	999	536586	2013/5/10 10:46:24	27151	10514	Running	2	TMGS->RBC	146	92000E0000594E083006000
154	1	65535	536586	2013/5/10 10:46:24	27151	10514	Running	1	TMGS<-RBC	3	0300FF000058DE018F03150
155	1	999	536586	2013/5/10 10:46:33	27989	11352	Running	2	TMGS->RBC	146	92000E00008B74083006000
177	1	23	536586	2013/5/10 10:53:42	73883	60482	Running	2	TMGS->RBC	136	88001800013302083006000
178	1	65535	536586	2013/5/10 10:53:46	74000	60653	Running	1	TMGS<-RBC	24	180010000140CA018F071D4
179	1	999	536586	2013/5/10 10:54:08	74539	61686	Running	2	TMGS->RBC	146	92000E0001414A083006000
180	1	999	536586	2013/5/10 10:54:24	75039	56298	Running	2	TMGS->RBC	132	8400180001145E083006000
181	1	999	536586	2013/5/10 10:54:53	78039	58273	Running	2	TMGS->RBC	132	8400180001145E083006000
182	1	65535	536586	2013/5/10 10:54:59	78595	58590	Running	1	TMGS<-RBC	3	0300C3000121B3018F07130
183	1	999	536586	2013/5/10 10:54:59	78595	58590	Running	2	TMGS->RBC	146	92000E00012242083006000
189	1	24	536586	2013/5/10 10:56:12	85914	61480	Running	2	TMGS->RBC	136	88001800013ED9083006000
190	1	65535	536586	2013/5/10 10:56:18	86539	61686	Running	1	TMGS<-RBC	24	180010000140CA018F071D4
191	1	999	536586	2013/5/10 10:56:24	87039	61851	Running	2	TMGS->RBC	132	8400180001433E083006000
192	1	999	536586	2013/5/10 10:56:47	89395	62623	Running	2	TMGS->RBC	146	92000E00014C72083006000
193	1	65535	536586	2013/5/10 10:56:47	89395	62623	Running	1	TMGS<-RBC	24	180010000140CA018F071D4
194	1	65535	536586	2013/5/10 10:56:47	89395	62623	Running	1	TMGS<-RBC	3	03009300014BD7018F071F0
195	1	999	536586	2013/5/10 10:56:50	89425	62899	Running	2	TMGS->RBC	146	92000E0001AD45083006000
203	1	999	536586	2013/5/10 11:00:53	114039	70384	Running	2	TMGS->RBC	132	8400180001ACB6083006000
204	1	999	536586	2013/5/10 11:00:55	114182	70399	Running	2	TMGS->RBC	146	92000E0001AD45083006000
205	1	65535	536586	2013/5/10 11:00:55	114182	70399	Running	1	TMGS<-RBC	24	180010000140CA018F071D4
206	1	65535	536586	2013/5/10 11:00:55	114182	70399	Running	1	TMGS<-RBC	3	0300430001ACB6018F08470
207	1	25	536586	2013/5/10 11:01:02	114895	70454	Running	2	TMGS->RBC	136	8800180001B00E083006000
208	1	26	536586	2013/5/10 11:01:42	114898	70474	Running	2	TMGS->RBC	156	9C0018000197B008300A
209	1	65535	536586	2013/5/10 11:01:43	114900	70474	Running	1	TMGS<-RBC	39	27000B000197B0008F0310

alternative way to cancel the database due to its distinguished characteristics, such as understandability, platform independence, the formatted structure, and the ability of hierarchical data description [20], [21]. It contains all the test data and can be directly read by the SC, which makes the test process supermatic. As in Fig. 9, when a test begins, the involved subsystems and the optional test sequence should be selected first. After the button "Finish" is clicked, the SC will load the XML configuration document and distribute the data to the corresponding subsystems according to the framing principle. Then, each subsystem begins to communicate with the RBC under test until the test process is an error or until the test is normally over.

Table II shows a part of the key local recorded messages of the NRBCGS and the TMGS in chronological order, from which the procedure of the RBC handover is presented through the messages between the subsystems.

"Msg_ID" records the message ID, and "MsgData_Bit" records the content of messages. Messages 201, 202, 203, 205, 221, and 222 that the NRBCGS exchanges with the RBC under test are the handover preannouncement command, the request for the authorized relevant information, the notification, the response, the authorized relevant information, and the taking over responsibility (TOR), respectively. Because Messages 202 and 221 are periodic messages, they are not listed here. Message 222 (TOR) is the last message that the accepting RBC sends to the handing-over RBC. Messages 132, 136, 146, and 156 that the TMGS sends to the RBC under test are the MA request, the position report, the acknowledgement, and the communication session end, respectively. Messages 132 and 136 are periodic messages, and Message 146 does not belong to the key messages; hence, they are not listed here. Messages 3, 24, and 39, i.e., the messages that the RBC under test sends to the TMGS,

are the MA, the general message, and the acknowledgement of communication session end, respectively. Message 39 is the last message that the handing-over RBC sends to the TMGS.

When the RBC under test sends the command of "Handing-over Preannouncement" (Msg_ID = 201) to the accepting RBC (the NRBCGS) [22], it indicates that there will be a train entering into its control area; after some necessary safety-related message interactions, the NRBCGS sends the command of "Taking over Responsibility" (Msg_ID = 222) to the RBC under test, which means that it has successfully controlled the train. When the RBC under test acknowledges ending the communication session (Msg_ID = 39) with the train, it finishes controlling the handover of the train. Thus, the practicality and feasibility of the test automated generation method are verified.

IV. Conclusion and Future Research

The test automated generation approach in this paper has provided a practical implementation method for automated testing, has put forward two practical and more efficient generation algorithms, and has accomplished the automated generation of test cases and sequences in accordance with the coverage criterion in Definition 4 and the relevant criteria in the specification [18]. Through this approach, the repeatability of the test cases and sequences is decreased to the greatest extent, and all the generated test cases are completely covered. In addition, this approach is able to adapt to the changes of the requirement specifications, effectively improve the test efficiency, lower the test difficulty, and achieve the goal of test automation. The future work is to generate more and impeccable test cases and sequences automatically and inject the fault factors to improve the safety of the train control system.

Acknowledgment

The authors would like to thank Mr. D. Ren for his contribution to this paper and the anonymous reviewers and the associate editor for their valuable suggestions.

References

[1] Z. Y. Yu and Y. Zhao, "Study of CTCS-3 integrated testing and commissioning technology based on black-box testing," *Railway Signaling Commun.*, vol. 46, no. 5, pp. 1–5, 2010.

[2] U. Farooq, C. P. Lam, and H. Li, "Towards automated test sequence generation," in *Proc. 19th Australian Conf. Softw. Eng.*, Mar. 2008, pp. 441–450.

[3] D. K. Kaynar, N. Lynch, R. Segala, and F. Vaandrager, "Timed I/O Automata: A mathematical framework for modeling and analyzing real-time systems," in *Proc. 24th IEEE Real-Time Syst. Symp.*, Dec. 2003, pp. 166–177.

[4] G. Xu and Z. M. Wu, "Formal design and analysis of FMS controller," in *Proc. IEEE Intell. Transp. Syst.*, Oct. 2003, vol. 2, pp. 1299–1304.

[5] A. Hessel and P. Pettersson, "A test case generation algorithm for real-time systems," in *Proc. 4th Int. Conf. Quality Softw.*, Sep. 2004, pp. 268–273.

[6] P. Samuel and A. T. Joseph, "Test sequence generation from UML sequence diagrams," in *Proc. 9th ACIS Int. Conf. Softw. Eng., Artif. Intell., Netw., Parallel/Distrib. Comput.*, Aug. 2008, pp. 879–887.

[7] Q. A. Malik, D. Truscan, and J. Lilius, "Using UML models and formal verification in model-based testing," in *Proc. 17th IEEE Int. Conf. Workshops ECBS*, Mar. 2010, pp. 50–56.

[8] M. Utting and B. Legeard, *Practical Model-Based Testing: A Tools Approach*. San Mateo, CA, USA: Morgan Kaufmann, 2006.

[9] S. Fujiwara, G. V. Bochmann, F. Khendek, and M. Amalou, "Test selection based on Finite State Models," *IEEE Trans. Softw. Eng.*, vol. 17, no. 6, pp. 591–603, Jun. 1991.

[10] H. Watanabe and T. Kudoh, "Test suite generation methods for concurrent systems based on Colored Petri Nets," in *Proc. Softw. Eng. Conf.*, Dec. 1995, pp. 242–251.

[11] D. Vernez, D. Buchs, and G. Pierrehumbert, "Perspectives in the use of Colored Petri Nets for risk analysis and accident modeling," *Safety Sci.*, vol. 41, no. 5, pp. 445–463, Jun. 2003.

[12] J. D. Lee, J. I. Jung, J. H. Lee, J. G. Hwang, and J. H. Hwang, "Verification and conformance test generation of communication protocol for railway signaling systems," *Comput. Std. Interfaces*, vol. 29, no. 2, pp. 143–151, Feb. 2007.

[13] J. H. Lee, J. G. Hwang, D. Shin, K. M. Lee, and S. U. Kim, "Development of verification and conformance testing tools for a railway signaling communication protocol," *Comput. Std. Interfaces*, vol. 31, no. 2, pp. 362–371, Feb. 2009.

[14] M. S. Chen, X. K. Qiu, and X. D. Li, "Automatic test case generation for UML activity diagrams," in *Proc. AST'06 Int. Workshop Autom. Softw. Test*, 2006, pp. 2–8.

[15] K. Jensen, "An introduction to the theoretical aspects of Colored Petri Nets," in *Proc. Decade Concurrency, Reflections Perspectives, REX School/Symp.*, 1993, vol. 803, pp. 230–272.

[16] B. Badban, M. Franzle, and T. Teige, "Test automation for hybrid systems," in *Proc. 3rd Int. Workshop Softw. Quality Assurance*, Nov. 2006, pp. 14–21.

[17] C. A. Shaffer, *Data Structures & Algorithm Analysis in C++*, 3rd ed. New York, NY, USA: Dover, 2011, pp. 390–394.

[18] "Functional requirements specification of the CTCS-3 Train Control System (v1.0)," The Ministry of Railways of The People's Republic of China, China Railway Publishing House, Beijing, China, No. 113, 2008.

[19] "Test cases of the CTCS-3 Train Control System," The Ministry of Railways of The People's Republic of China, China Railway Publishing House, Beijing, China, No. 59, 2009.

[20] S. S. Chawathe, "Real-time traffic-data analysis," in *Proc. 7th IEEE Conf. Intell. Transp. Syst.*, Oct. 2004, pp. 112–117.

[21] R. M. Li, H. P. Lu, Z. Qian, and Q. X. Shi, "Research of in the integrated transportation information platform based on XML," in *Proc. IEEE, Intell. Transp. Syst.*, Sep. 2005, pp. 498–503.

[22] "System requirements specification of the CTCS-3 Train Control System (v1.0)," The Ministry of Railways of The People's Republic of China, China Railway Publishing House, Beijing, China, No. 127, 2008.

Wei Zheng received the B.E., M.E., and Ph.D. degrees in control science and engineering from Harbin Institute of Technology, Harbin, China, in 1997, 1999, and 2002, respectively.

From 2007 to 2008, his postdoctoral research was done in the Institute of Traffic Safety and Automation Engineering, Technical University of Braunschweig, Braunschweig, Germany, where he is currently an Alexander von Humboldt Fellow as an experienced Researcher from 2013 to 2014. He is currently an Associate Professor with the National Engineering Research Center of Rail Transportation Operation and Control System, Beijing Jiaotong University, Beijing, China. His research interests include the development of testing platforms, safety system analysis based on formal methods, and the safety assessment of safety-critical systems.

Ci Liang received the Bachelor of Engineering and Master of Engineering degrees from Beijing Jiaotong University, Beijing, China, in 2010 and 2013, respectively.

She is currently working with CASCO Signal Ltd., Beijing, China, which is the joint venture of ALSTOM and China Railway Signal and Communication Company. Her research interests include rail transportation, Chinese train operation control systems, automated tests, and system safety.

Rui Wang received the Bachelor of Engineering degree in electronic and information engineering from Beijing Jiaotong University, Beijing, China, in 2011. She is currently working toward the Master of Engineering degree in electronic and information engineering in the National Engineering Research Center of Rail Transportation Operation and Control System, Beijing Jiaotong University.

Her research interests include system safety theory, traffic safety analysis methodologies, Petri net modeling, rail transportation, and the Chinese train control system.

Weijie Kong received the B.S. degree from Lanzhou Jiaotong University, Lanzhou, China, in 2011. She is currently working toward the M.S. degree in the National Engineering Research Center of Rail Transportation Operation and Control System, Beijing Jiaotong University, Beijing, China.

Her current research interests include the high-speed railway control system and the safety analysis method.