Modelling and design of the formal approach for generating test sequences of ETCS level 2 based on the CPN

X. Zhao[1], Y. Zhang[1], W. Zheng[2], T. Tang[1] & R. Mu[2]
[1]*State Key Laboratory of Rail Traffic Control and Safety, Beijing Jiaotong University, China*
[2]*School of Electronic and Information Engineering, Beijing Jiaotong University, China*

Abstract

ETCS Level 2 (European Train Control System Level 2, ETCS-2) has drawn particularly attention from researchers and industries. A new CPN model-based formal approach for test cases and sequences generation is proposed in this paper to increase the test automation degree of the ETCS-2 system and subsystems.

In this paper, a set of modelling rules is presented firstly to make the Coloured Petri Net (CPN) model more suitable for test generation. Then, an automated test approach is described in detail, which includes an automatic test case generating algorithm and a type of automatic test sequence searching algorithm. The generated set of test cases satisfies specified coverage. The test sequence searching algorithm guarantees the results satisfying the minimum number of test sequences covering all test cases. The output of this approach is a set of well-formed XML (Extensible Markup Language) file to increase the automation degree of the test executing process. Finally, a partial model of ETCS-2 On-Board subsystem is built and analysed using the CPN Tools as a case study. The model-based formal approach is implemented on this model and the test cases and test sequences are all generated in a form of XML. The conclusion show that the CPN-model based testing approach can be used to improve the automation of the testing procedure and the generated test cases can meet the relative requirement.

Keywords: ETCS-2, CPN, test generation, formal method.

WIT Transactions on The Built Environment, Vol 114, © 2010 WIT Press
www.witpress.com, ISSN 1743-3509 (on-line)
doi:10.2495/CR100661

1 Introduction

In recent years, the safety-critical system has been come closer to peoples` life. Safety critical system (SCS) is a computer, electronic or electromechanical system whose failure may cause injury or death to human beings as in Chen [1]. ETCS-2 is a typical SCS. In order to ensure the correctness of its behaviour function, there are two commonly used techniques: validation and testing. Testing is the only method which can be used to verify the dynamic behaviours of SCS in running time as in Wegener et al. [2].

With more and more attention has been paid to the testing automation of the Safety-Critical System. How to improve test automation and testing efficiency, and reduce testing costs and risk factors of the testing process has increasingly become the focus and hot spots of the research in the testing field. Model-based testing (MBT), which is to compare the I/O behaviours of a valid behaviour model with that of a system to be tested (the system under test, SUT), has been closely watched in recent years.

Model based test generating, which is a method to generate the test cases and test sequences according to the formal model of SUT, is the most important content of MBT. Since 1970s, there had been many test generating methods based on variety of models, such as U-method in Chan and Vuong [3], D-method in Sidhu and Leung [4] and Wp-method in Fujiwara and Bochmarm [5]. But these methods cannot describe the time constraints. Since the 1990s, with the gradual maturity of many formal modelling theory, such as the Temporal logic in Lamport [6], Time Input/Output Automata (TIOA) in Alur and Dill [7] and Timed transition system in Henzinger et al. [8], many Model based test generating methods based on these models has been presented, including Test time Automa in Badban et al. [10] and TIOA based testing method in Hessel et al. [9] etc. However, most of these methods can not describe the Concurrent behaviours of the SUT, also the test cases and sequences generated through these methods are too abstract to be executed, and the generating process is not automatic. Kim et al. generate the test cases separately according to the control flow and data flow on the basis of UML state charts model in Kim et al. [11]. However, its limitation on describing the communication between the numbers of objects causes the low test generating coverage. Nogueira et al. [12] and Helke et al. [13] did the test sequence generation based on the Communication Sequence Process (CSP) model and Z model, but these models are too abstract which makes the generating result more unexecutable.

Table 1 is simply comparing among the formal modelling languages which have been used in test generation.

According to these advantages of CPN described in Table 1, this paper presents a test cases and sequences generating approach on the basis of CPN, and applies this approach to the ETCS-2 system testing. The paper is organized as follow. In Section 2, we define the test case, test subsequence, test sequence and test coverage degree in a formal way according to the CPN definition. In Section 3, we describe the test generation method, including the test case generating method, the test subsequence generating method and the test sequence

Table 1: Compare among the modelling languages.

Formal Language	Modelling Level	Verification capability	Executable	Modelling process	Data Type
TIOA	Abstract	Strong	No	Easy	Medium
UML	Abstract/Concrete	No	Yes	Easy	Rich
CSP	Abstract	Strong	No	Hard	Simple
Z	Abstract	Strong	No	Hard	Simple
CPN	Abstract/Concrete	Medium	Yes	Easy	Rich

generating method. In section 4, a XML format for describing test cases and test sequences is proposed. In Section 5, together with the example of On-Board subsystem in ETCS-2. Finally, we evaluate the whole method and discuss possible improvements in the future.

2 CPN based modelling method for test generation

2.1 Coloured Petri Net and relative definitions

Coloured Petri Nets (CPN) is an extended Petri Nets which is a graphical and mathematical modelling tool proposed by Kurt Jensen. And it can be used to model systems with complex procedures as in Jensen [14] and applicable to describe many types of systems. The locations that can be used to carry information in the graph element of CPN are showed in Fig. 1.

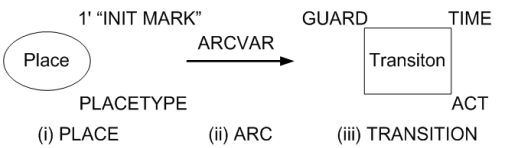


Figure 1: Information Location in CPN graph element diagram.

In Fig. 1, the "INIT MARK" location in (i) carries the initialized data; the "PLACETYPE" location in (i) carries the colour types information; the "ACRVAR" location in (ii) carries the name of the variable to be passed; In (iii), the "GUARD" location carries the data selection and comparison information; the "TIME" location carries the time restriction information; while the "ACT" location carries the data computing information. In the following part, these locations are used to be searched for information that needed by test data generation process.

On the basis of the definition of CPN in Jensen [14], some relative formal definitions will be introduced and these will be the foundation of the following work.

Definition 1. Test Case Based on the CPN

A Test Case Based on the CPN is an eight-tuple $TC_{CPN} = \{I_A, I_D, O_A, O_D, SC, SC_D, EC, EC_D\}$, where:

I_A is a finite subset of input ports, $I_A \subset PN$ and $\forall n \in I_A : [PT(n) = in]$.

O_A is a finite subset of output ports, $O_A \subset PN$ and $\forall n \in O_A : [PT(n) = out]$.

I_A and O_A must be in the same subpages: $I \cup O \subset Ps, s \in S$;

I_D is a set of the input data, and corresponding with the I_A.

O_D is a set of the output data, and corresponding with the O_A.

$SC = \{GF_{SC}, IF_{SC}, IP\}$ represents the start condition, and is a finite set of fusion places and internal input ports.

$EC = \{GF_{EC}, IF_{EC}, OP\}$ represents the end condition, and is a finite set of fusion places and internal input ports, where:

$$GF, IF \subset FS, [\forall f \in GF, FT(f) = globle] \cup [\forall f \in IF, FT(f) = page]$$

$$IP, OP \subset PN, [\forall p \in IP, PT(p) = in] \cup [\forall p \in OP, PT(p) = out]$$

Both in SC and EC, the GF set and IF set can not be empty at the same time, and the situation (GF_{SC}=GF_{EC}) ∪ (IF_{SC}=IF_{EC}) should not exist in one test case.

SC_D is the data set of the start condition corresponding to SC, and EC_D is the data set of the end condition corresponding to EC.

Definition 2. Test Subsequence

A test subsequence is a six-tuple TSsub = {SS, SC_{SS}, SC_D, EC_{SS}, EC_D, w}, where:

SS is a finite set of test cases which are in order, and the order reflects the sequence of the test cases to be executed in the subsequence. Here, tc_1 represent the first test case to be executed and tc_n represents the last test case to be executed in the subsequence.

SC_{SS} is a finite set of the start conditions, $SC_{SS} = \{GF_{SC_{SS}}, IP_{tc1}\}$, where $GF_{SC_{SS}} = \bigcup GF_{SCtci}, 1 \le i \le |SS|$ which means that the start condition set $GF_{SC_{SS}}$ is the combination of the start condition; the start condition of a test subsequence IP_{tc1} is the same with IP set in the first test case in the subsequence.

EC_{SS} is the set of end condition: $EC_{SS} = \{GF_{EC_{SS}}, OP_{tc_n}\}$, where

$$GF_{EC_{SS}} = \bigcup GF_{ECtci}, 1 \le i \le |SS|$$

SC_D and EC_D is the data set corresponding to SC_{SS} and EC_{SS} separately.

Note: For each subpage S, there is a test case set TCS corresponding to it. If there is a subset $TC_{sub} \subset TC_S$ makes the

$$\forall tc_i, tc_j \in TC_{sub} \bullet [IF_{SCi} = IF_{SCj} = IF_{ECi} = IF_{ECj}] \ (0 \le i \ne j < |TC_{sub}|)$$

coming into existence, then we can get a corresponding test subsequence, and then get the executing order of the test cases in the SS set according to the IF_D information of each test case.

w=|SS| is the number of the test cases in a test subsequence, which represents the weight of the test subsequence and contributes to the optimizing of the test sequence.

Definition 3. Test Sequence

Test sequence is a five-tuple {TS, SC_{SS}, SC_D, EC_{SS}, EC_D}, where:

TS is a set of test sub sequences and test cases, which is an ordered set. The order reflects the executing order of the test cases and the test sub sequences. Here, tc_1 represent the first test case to be executed and tc_n represents the last test case to be executed in the subsequence.

SC_{TS} is the start condition set which is a finite set of global fusion places. If the first one to be executed in the test sequence is a test case, then $SC_{TS} = GF_{SCtc_1}$; else if the first one to be executed in the test sequence is a test subsequence, then $SC_{TS} = SC_{SStc_1}$.

EC_{TS} is the end condition set which is a finite set of global fusion places. If the first one to be executed in the test sequence is a test case, then $EC_{TS} = GF_{ECtc_1}$; else if the first one to be executed in the test sequence is a test subsequence, then $EC_{TS} = EC_{SStc_n}$.

SC_D and EC_D is the data set corresponding to SC_{SS} and EC_{SS} separately.

Definition 4. Test cases Coverage Criteria – All-Edge Coverage

A test case set satisfies all-edge coverage means that if we execute all test cases on system CPN model, all arcs will be passed at least once. For the non-architecture CPN model $S = \{\Sigma, P, T, A, N, C, G, E, I\}$, the all-edge coverage $E(S)$ can be defined as: $\forall arc \in A \Rightarrow arc \in E(S)$.

Definition 5. Test sequences Coverage Criteria – All test cases Coverage

All test cases coverage means that the test sequences set should cover each test case in the test cases set at least once.

Definition 6. Test sequences Coverage Criteria – All Path Coverage

According to the start condition and end condition of every test case, we can represent the relationship using a Directed graph. All path coverage means that the test sub-sequences set should cover all edge in the test cases set at least once.

2.2 Rules of CPN based modelling for test generation

The definitions 1 to 3 can be the generting targets. And in response to these generation targets, 12 rules are presented in this section to regulate the modeling process, and cover all of the information:

Rule 1: Using hierarchical modeling approach to build a three-tier model, including System level, Senario level and Function level, to make the model clearer and easier to manage. Moreover, this will make the input/output type and the port information available.

Rule 2: In the model, the ports between different object must have INPUT or OUTPUT option; It is not allowed that the port with I/O option existing in the model.

Rule 3: The data processing-related functions, such as data decomposition, data restructuring and data searching, must be implemented with the use of Meta-Language (ML).

Rule 4: The Branch selection structure like: if/else and case/swich, should not be described by the ML languange, these should be modeled using places and transitions.

Rule 5: All of the judgment of the data content should be reflected in the "GUARD" location, and the Color name should be reserved.

Rule 6: In the internal of the third level sub-pages, use Partial Fusion Places to reflect the order information, and use nature number N to express the order.

Rule 7: In the third level, use Global Fusion Places and Internal Ports to reflect the relationship between sub-pages for test sequence generation.

Rule 8: In the whole model, the color definition, the variable naming the place naming should obey a unify rules.

Rule 9: Exept for the arcs connected with the fusion place, the variables on the other arcs should be belonged to the color set defined globally.

Rule 10: The definition of the color sets should separates the input color set, the output color set and other type of color set; So that we can recognize the input port and output port in the model.

Rule 11: The Fusion places is only allowed to be exsiting in the third level model.

Rule 12: During the color set definition, the variables should be defined with the value range according to the specification.

3 Test sequence generation method

In this section, we first describe the integrated structure of the overall test generating method. There are three phases: modelling, test case generating and test sequence generating. The overall block diagram is shown as Fig. 2.

In the modelling phase, the model must conform to the System Requirement Specification and also satisfy the modelling rules presented in section 2, and some model checking based verification methods are used here to make sure the correctness and conformance of the model.

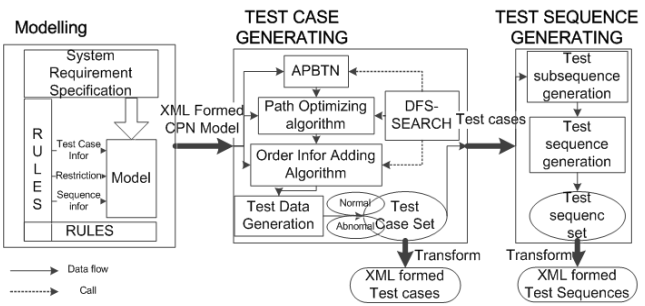


Figure 2: Overall block diagram of test generation.

3.1 Test case generation

Based on the definition of Test Case given in section 2, the generation process can be divided to 3 steps: (1) Generation of input ports set I_A and output ports set O_A; (2) Generation of start condition SC and end condition EC; (2) Generation of I_D, O_D, SC_D and EC_D (Test Data Generation)

The main idea of the test case generating method is described as follows: In each non-hierarchical CPN model, we first specify all input ports in the model as a starting set and all output ports in the model as an ending set, then make use of the APBTN algorithm to get all possible paths between these two sets to form the path set PATH1. Second, a Path optimizing algorithm is proposed to select valid paths and combine invalid paths into valid paths, and obtain the valid path set PATH2. Third, an order information adding algorithm is used to add start condition information and end condition information to PATH2. For each path in PATH2, the start condition is the fusion places or internal input ports which can reach to the path through a sub path, and the end condition is the fusion places or internal output ports that can be reached from any node in the path. Then, we can get the complete valid path set PATH3. Forth, a input and start condition data searching algorithm is used to get the input data and the start condition data of each path in PATH3. At last, making use of the dynamic executing property of CPN model and the interfaces with programming languages supplied by CPN Tools, we can get the expected output data and the end condition data.

Until now, through the generation of ID、SCD、ECD and OD, we can get all the test cases of the CPN model.

3.2 Test sub-sequence generation

The test cases generated last section includes the start condition and end condition. On the basis of this information, this section and the next section will introduce the test sub-sequence generating method and the test sequence generating method.

First, we explain the reason for generating sub-sequence.

As shown in Fig. 3, we assume that the Scenario I includes eight test cases. In the SC of TC1, the GF or IP is not empty which means that the start condition of this case is a global one that coming from another scenario. In the EC of TC6 and TC8, the GF or IP is not empty which means that these test cases can lead to other scenario. In the SC of the other five test cases in Scenario I, the GF and IP is empty, so that these test cases have no information helping to organize them directly into test sequence. But the IF of SC in these test cases is not empty and can help to organize them into sub sequence in the internal of Scenario I, and the sub-sequences have the information that can contribute to generating test sequences, such as sub-sequence TC1→TC2→TC7→TC8. Simply to say, the target of test sub-sequence generation is to connect the test cases in each scenario and make them having the information that can help to organizing them into test sequence.

Sometimes, we only need that every test case is executed just once; but sometimes, we need that each possible path in the scenario must be executed. For

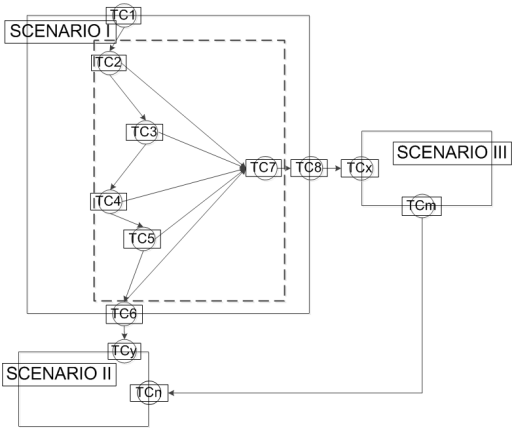


Figure 3: Relation between test cases and sub-sequences.

the two situations, two different algorithms are proposed to get different sub-sequence sets.

3.3 Test sequence generation

On the basis of the start condition, end condition and weight information supplied by test sub-sequences. We proposed a test sequence generation method. The generating target is:

a. The least number of test sequences;
b. The least repeat of test cases;
c. All test sub-sequences covered = all test cases covered;

Through this approach, we can obtain a complete test sequence set. But this set is not the unique one. According to different requirements, we can get different test sequence sets. For example, when first test the system, there may be many errors existing in the system, if the test sequence is too long, it may cause the test sequence can not be executed fully. So under this situation, the requirement may be to generate test sequences with suitable length. For different requirement, the algorithm can be adjusted.

4 XML based test description method

According to the generating result of section 3, the information required in the definition of test case and test sequence has already been included. But they can not been executed directly yet, we need an efficient description method to make the test cases and test sequences more executable.

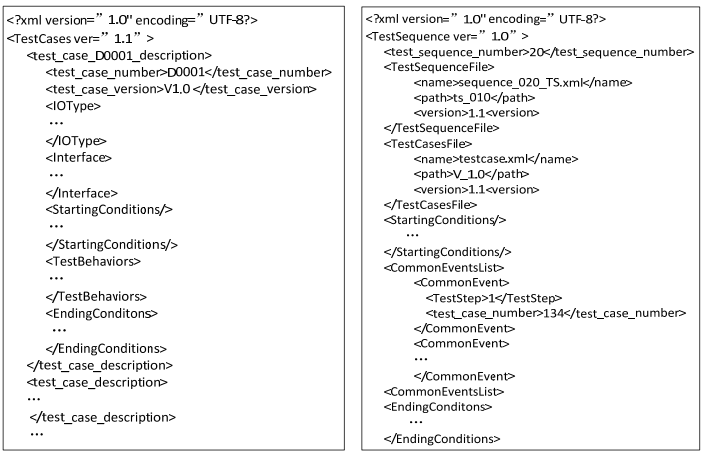
XML (Extensible Markup Language) is a very suitable language for test description which has many advantages:

- Structured, expandable, platform-independent and standardized language for the description of data;

- Good and suitable for formal data format for the representation of tests and their reference data;
- Equally readable and interpretable for humans and software applications.

Based on these advantages, this paper proposes a XML format for test case and test sequence, shown as Fig. 4.

This XML format has been used as the target of the test case generating and test sequence generating.



(i) Test case format (ii) Test sequence format

Figure 4: ML format for test case and test sequence.

5 Case study

In this section, we will apply the whole approach. First, according the modelling rules presented in section 2.2 and the "ETCS system requirement specification Subset 026" (SRS), we finished the model in System level and Scenario level, and the function model of the Registration and Start Scenario (R&S), On Sight mode Running Scenario (OSR) and the Logout Scenario (LO) in Functional level.

The result of test case generation for each scenario is shown below:

Table 2: Test case generating result.

Scenario	Places Number	Transition Number	Test case Number	Coverage Criteria	Test case Cover	Test case uncover
R&S	48	61	48	All-Edge Coverage	51	3
OSR	18	15	11		21	0
LO	25	16	14		6	0

The "Test case cover" column and the "Test case uncover" column in the table represent the comparing result between the generating result and the ERTMS/ETCS SUBSET 076-5-2:Test cases related to features (SUBSET076-5-2). From the table, we can see that the generating result of On Sight Scenario and Logout Scenario has covered all corresponding test cases in SUBSET076-5-2, and there are only three test cases in SUBSET076-5-2 have not been covered in Registration and Start Scenario. They are:

(1) The On-board sub system should show driver the SB mode when it is in SB mode;

(2) The On-board sub system should show driver the ETCS level 2 when it is in ETCS level 2;

(3) When the received message has conformance error, the On-Board subsystem should send error report to Radio Block Centre subsystem.

Through analyzing, we found that because the functions corresponding to these three uncovered test cases are not only belonging to one scenario, and we modelling these functions into a separate scenario model. That is to say, these test cases can be covered by another scenario test generating result.

According to the test case generating result, we generate the sub-sequences and test sequences. The result is shown in table 3 and table 4.

Table 3: Test sub-sequence generating result.

Scenario	Test case Number	Coverage Criteria	Subsequence Number
R&S	48	All Test cases Coverage	11
OSR	11		6
LO	14		6

Table 4: Test sequence generating result.

Scenario Sequence	Subsequence Number	Coverage Criteria	Subsequence Number
1. R&S	11	All Test cases Coverage	11
2. OSR	6		
3. LO	6		

The generating result shows that it satisfies all executing requirements, because it has all test information and test data which is needed by the executing process.

6 Conclusion

This paper has proposed a new type of model-based formal approach for test cases and sequences generation and applied it to ETCS-2 system and subsystems. This approach ensures the availability from three aspects. First, the modelling rules make the system model containing all the information that testing process needs. Second, the test cases generating method and test

sequences generating method make the generating process more automatic. Third, the XML test description form helps to increase the executable degree of the test cases and test sequences. The case study shows that this method has a high coverage and test generating automation degree.

Acknowledgements

We wish to acknowledge the support of the National High-Technology Research and Development Program ("863"Program) of China No. 2009AA11Z221, National Science & Technology Pillar Program of China No. 2009BAG12A08, and the Fundamental Research Funds for the Central Universities No. 2009YJS013.

References

[1] Chen W., Xue Y., & Zhao C., (eds). A Method for Testing Real-Time System Based on Timed Automata. *Journal of Software*, **18(1)**, pp. 62-73, 2007.

[2] Wegener J., Sthamer H., Jones B.F., & Eyres D.E., Testing real-time systems using genetic algorithms. *Software Quality Journal*, **(6)**, pp. 127-l35, 1997.

[3] Chan W., & Vuong C., (eds). An improved protocol test generation procedure based on UIOS. *Proc. on Communications Architectures& Protocols*, eds. Landweber LH, Symp. ACM Press: New York, pp. 283-294, 1989.

[4] Sidhu D., & Leung T., Formal methods for protocol testing: A detailed study. *IEEE Trans. on Software Engineering*, **15(4)**, pp. 413-426, 1989.

[5] Fujiwara S., & Bochmarm G.V., Test selection based on finite state models. *IEEE Trans. on Software Engineering*, **17(6)**, pp. 591-603, 1991.

[6] Lamport L., The temporal logic of actions. *ACM Trans. on Programming Language and Systems*, **16(3)**, pp. 872-923, 1994.

[7] Alur R., & Dill D., A theory of timed automata. *Theoretical Computer Science*, **l26(2)**, pp. 183-235, 1994.

[8] Henzinger T., Manna Z., & Pnueli A., Timed transition system. *Proc. of the Real-Time : Theory in Practice, REX Workshop*, eds. J.W.D. Bakker, C.Huizing, W.P.D. Roever, G. Rozenber, LNCS 600, Springer-Verlag: Berlin, pp. 226-251, l992.

[9] Hessel A., Larsen KG., & Mikucionis M., (eds). Testing real-time systems using UPPAAL. *Lecture Notes in Computer Science, v4949 LNCS*, pp. 77-117, 2008

[10] Badban B., Franzle M., & Teige T., Test Automation for Hybrid Systems. *Proc. of the Third International Workshop on Software Quality Assurance*, pp. 14-21, 2006.

[11] Kim Y.G., Hong H.S., & Bae D.H., (eds). Test cases generation from UML state diagram. *IEEE Proceeding-Software*, **146(4)**, pp. 187-192, 1999.

[12] Nogueira S., Sampaio A., & Mota A., Guided Test Generation from CSP Models. *Theoretical Aspects of Computing - ICTAC 2008, Lecture Notes in Computer Science,* Springer-Verlag, pp. 258-273, 2008.
[13] Helke S., Neustupny T., & Santen T., Automating Test Case Generation from Z Specifications with Isabelle. *ZUM '97: The Z Formal Specification Notation, LNCS 1212*, eds. J.P. Bowen, M.G. Hinchey and D. Till, Springer-Verlag: pp. 52-71, 1997.
[14] Jensen K., Coloured Petri Nets. *Basic Concepts, Analysis Method and Practical Use (Vol.1-3)*. Monographs in Theoretical Computer Science, Second Edition, Springer-Verlag, 1997.