Model-based Test Generation Techniques Verifying the On-board Module of a Satellite-based Train Control System Model

Daohua Wu
and Eckehard Schnieder
Institute for Traffic Safety and Automation Engineering
Technische Universität Braunschweig
Langer Kamp 8
38106 Braunschweig, Germany
Email: {wu, schnieder}@iva.ing.tu-bs.de

Jan Krause
ifak Magdeburg
Werner Heisenberg Str. 1
39106 Magdeburg, Germany
Email: jan.krause@ifak.eu

***Abstract*—Testing as a means of verification during system development, aims at demonstrating the compliance of actual and intended behaviours of the system. However, the degree of automatic test generation and test coverage are big challenges in testing. Concerning these problems, two approaches are proposed to test the on-board module of an advanced satellite-based train control system model.**

The first approach is a model-based test generation technique based on Coloured Petri Nets (CPNs). The test model includes a behavioural module of the on-board subsystem model and two other modules for the application environments of the on-board subsystem model. The behavioural module and modules of application environments form a closed system. The expected output of a test case is extracted from a path of the reachability graph of the test model. Beside the CPN based approach for generating test cases, another model-based test generation technique is applied. This method uses a special Petri net dialect called Safe Place Transition Net with Attributes (SPENAT) to model the intended behaviour of the test object, the on-board subsystem model in this case. No environment of the test object has to be modelled, because of the possible input/output modelling with a SPENAT. Thereby, a SPENAT is an open system and its transitions can be triggered by external events with parameters (external data). The identification of suitable test cases is based on the construction of a (complete) prefix of the SPENAT model and the specified coverage criteria.

With CPN based (closed system) and SPENAT based (open system) approaches, the verification of a railway operations control system model, such as the on-board subsystem model in this case, could be done by testing. At last, the results of both approaches are discussed and the advantages as well as disadvantages are illustrated.

I. Introduction

Nowadays system development is shifting from informal textual specifications and manual coding techniques to a model-based and tool-supported automated code generation process. With model-based system development, formal methods can be applied during the developing process. Given a system model designed for system development, executable codes of the system can be generated automatically from the model following certain transformation rules. Hence the quality of the model is vital for the developed system.

To improve and ensure the quality of the system model, several techniques could be applied simultaneously, e.g., formal analysis, simulation and testing. All these techniques should be carried out from the very early stages of the model-based system development process, e.g., the stage of functional high-level design, which produces a high-level abstract model focusing on functionalities of the system. Testing aims at verifying the conformance of the system model behaviour with the system requirements. The earlier the defects are detected, the lower the cost will be. If executable models of different development stages are used for performing tests, tests are able to be executed at very early stages in model-based system development processes.

Model-based system development centralises the development of the system model in early stages for system design. An executable system model has the advantage of testing via simulation or model execution in the phase of system design. Based on the well-known "V" model of testing and "V" representation of the lifecycle in the railway application in norm EN50126 [1], testing could be integrated into the model-based system development process for railway applications as described in Fig. 1. By integrating testing into the model-based development process, several times of testing process will be carried out during the phase of system design. However, the efforts of testing several times could be reduced by the idea of reusing the test suites for different models of different abstraction levels. Taking the view of black-box on models of different abstraction, the test suites can be reused if the interface definitions of the models of different abstraction are constant.

II. Overview of the SatZB Model

In order to derive a test model with Coloured Petri Nets (CPNs or CP-nets) [2] from the requirement specification documents of SatZB, it is necessary to get a general understanding of this satellite-based train control system.

SatZB (German "Satellitengestützer Zugleitbetrieb") is an advanced satellite-based train control system, which uses GNSS data to provide the location of the train for automatically train control. It consists of two major subsystems, an on-board

978-1-4673-5277-2/13/$31.00 ©2013 IEEE

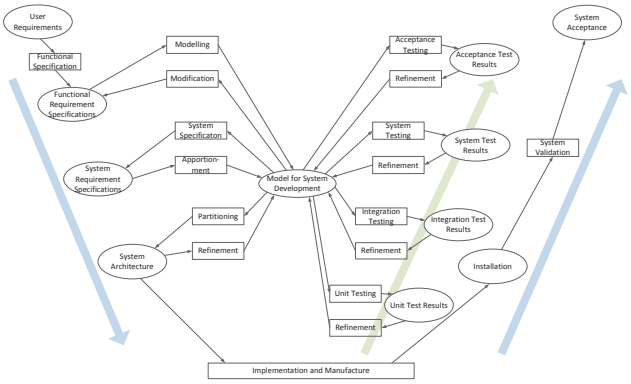

Fig. 1. Integration of testing and model-based system development

subsystem and a trackside controller (traffic control centre). The on-board subsystem communicates with the trackside controller by wireless communication with GSM.

A. CPN Model of the SatZB

A first CPN model with continuous simulation capability of SatZB without considering time constraints has been developed previously at the Institute for Traffic Safety and Automation Engineering, TU Braunschweig. There are five modules: `On-board`, `Communication`, `Traffic Control Centre`, `Localisation Unit` and `Train Movement`. Each module represents the corresponding subsystem (or component) of SatZB where `Communication` represents the communication system between the on-board subsystem and traffic control centre, and `Train Movement` stands for the movement of the train and the system environment. The `Localisation Unit` is an on-board unit in the real system. More details of the modelling approach and the whole established model can be found in [5].

Since the CPN modelling language supports the development of hierarchically structured models, the SatZB model has been established in several abstraction levels. There is a top level net for each subsystem model. It shows the connections of one subsystem model to other subsystem model and the corresponding communication channels on an abstract level. The second level is subsystem level. On this level, multiple *scenario nets* are used to describe the behaviour of the relatively independent subsystems, i.e., the on-board subsystem and the traffic control centre. A scenario net of a subsystem model describes a specific process of internal data processing with respect to the incoming data from the environment of the subsystem model. In addition, function blocks are coordinated with scenarios on this level for the purpose that different scenario nets can call the same function block, which is realised by using *fusion places* [2]. On the third level, each net represents either a detailed scenario net or a detailed function block.

B. Scenarios and Scenario Nets

In railway engineering, *scenarios* [3] are usually used to describe a number of operation processes, in which sets of functions are involved. Scenarios define operational situations

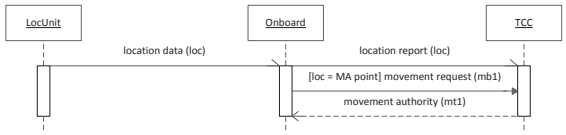

Fig. 2. Sequence diagram for the scenario RUNNING

and processes with respect to the environment of the system in focus; they show how the system and its components (subsystems) interact with each other and the environment; a scenario is a description of the behaviour of a system or a collection of components, embedded in a comprehensive operational process; and it is determined in an unambiguous way by specific starting conditions and a sequence of events in the environment of the components in focus. For instance, following text selected from the functional requirement specification of SatZB describes the scenario that releases a block section for the train's running. We call this scenario RUNNING.

"*The train that in a train station or on a free block section has received a movement authority for the next block section and has/hasn't entered this block section. A release for the following block section is/isn't required.*"

For scenario depictions, sequence diagrams of UML [14] are adopted. A sequence diagram is a kind of interaction diagram that shows the interaction between objects over time. The requirement (scenario RUNNING) presented above can be mapped into the sequence diagram shown in Fig 2. There are three objects: `LocUnit`, `Onboard` and `TCC` representing the localisation unit, the on-board subsystem and the traffic control centre, respectively. The diagram illustrates that when the on-board subsystem receives a location data (`loc`) from the localisation unit, a location report will be sent to the traffic control centre immediately. If the value of the location data is equal to the a *MA point* stored by means of the on-board map in advance, then a movement request (message `mb1`) will be sent to the traffic control centre. A MA point is a point that when the train passes over, a movement request should be sent to the traffic control centre. Consequently, if the train wants to enter the following block section, a movement authority (message `mt1`) for the block section is required.

However, the distinction between the scenario nets in the system model and scenarios for describing the operational process of the system should be clarified. In the CPN model of the SatZB system, the scenarios such as described in Fig. 2 are realised by exchanging tokens between different scenario nets in different subsystem or component models. A scenario net of a subsystem model is a *page* [1] that describes a specific process of internal data processing with relation to the incoming data from the environment of the subsystem model. For example, the scenario net SCE_RUNNING in the on-board subsystem model, is a page that shows the process of internal data processing, which is the consequence of the reception of a

[1] A page is a part of a CP-net that links to other pages through shared places and thereby enables to exchange tokens between pages in different layers.

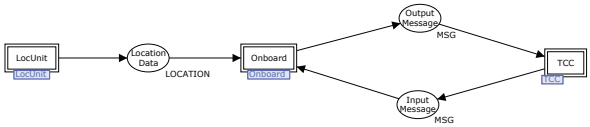

Fig. 3. Top level of the test model

movement authority (message `mt1` in Fig. 2). From this point of view, the scenario net in the system model and the scenario of the system have a one-to-one relation.

III. Testing with CPNs

Petri nets are well-known as adequate means of descriptions for distributed systems and the system behaviour of concurrency. The environment of the on-board subsystem of SatZB, i.e., the traffic control centre, is characterised by the concurrency of delivering different commands to the on-board subsystem while taking the on-board subsystem as the test object. Thus (Coloured) Petri nets are feasible to be used for developing the test model. Due to the length limitation of the paper, the introduction of CPNs will not be presented here and we recommend the book [2] to a starter.

A. Test Model

The CPN modelling language supports the specification of hierarchically structured models, which makes it possible to work with different levels of detail and abstraction.

1) Top Level: Considering the system architecture as well as the sequence diagrams (e.g. Fig. 2) that describe the scenarios of the train control system, the top level of the test model could be specified as in Fig. 3. The object elements of the sequence diagram are mapped to substitution transitions on the top level of the test model, representing the respective subsystems; the messages transmitted between objects are represented by the tokens on the places connecting to the objects. Place `Location Data`, `Output Message` and `Input Message` represent the communication channels between different modules. These communication channels are simplified as one-way channels under the condition that SatZB is a real-time system, and only one message would be transmitted on a communication channel at the same time. Moreover, we suppose that the communication system of SatZB is fully reliable.

2) Second Level: In order to cover all the scenarios and possible scenario sequences, all the scenario nets and possible combinations of scenario nets need to be covered. To exemplify our testing approach, five scenarios of the on-board subsystem are considered. These scenarios are STANDBY, RUNNING, CONDITIONAL RUNNING, BAN OF ENTRY and EMERGENCY STOP, which lead to five scenario nets SEC_STANDBY, SEC_RUNNING, SEC_CONDITIONAL RUNNING, SEC_BAN OF ENTRY and SEC_EMERGENCY STOP accordingly. The switchovers between different scenario nets in the system model are depicted in Fig. 4, where the `mb1` and `mb5` are messages (requests) transmitted from the on-board subsystem model to the model of traffic control centre, and messages `mt1`, `mt2`, `mt3`, `mt4`, and `mt5` (commands) from the

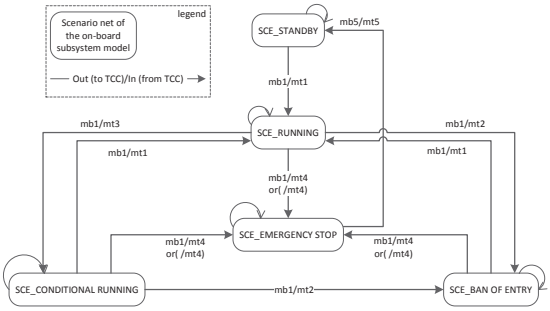

Fig. 4. Switchovers of the scenario nets of the on-board subsystem model

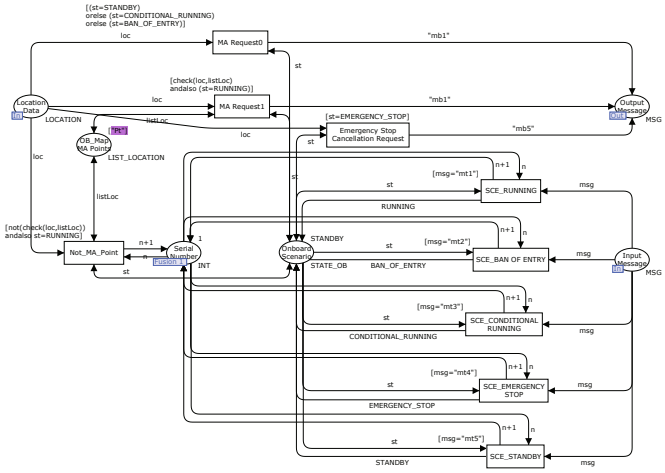

Fig. 5. Page of the submodule `Onboard`

model of traffic control centre to the on-board subsystem model.

- Submodule `Onboard`

The main idea of this page is to reflect the interactions between the on-board subsystem model and its environment illustrated in the sequence diagrams. From the given interpretation of the sequence diagram in Fig. 2, we conclude that sending movement requests and switchovers of scenario of the on-board subsystem model are determined by the received location data from the model of localisation unit and the messages from the model of traffic control centre, respectively. This is depicted in Fig. 5. When a location data is received from the place `Location Data`, it is compared to the on-board map (tokens on the place `OB_Map MA Points`), and a specific message (either `mb1` or `mb5`) might be sent out via the place `Output Message` under the consideration of the marking on the place `Onboard Scenario`. The current scenario of the on-board subsystem model, indicated by the marking on the place `Onboard Scenario`, is varied depending on the messages received from the place `Input Message`. Place `Serial Number` is used to synchronise the submodule `Onboard` and `LocUnit`, which will be introduced in the following subsection.

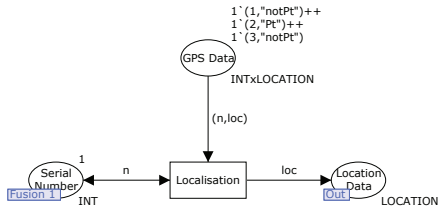

Fig. 6. Page of the submodule `LocUnit`

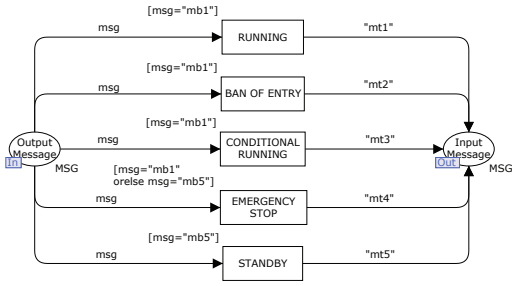

Fig. 7. Page of the submodule `TCC`

- Submodule `LocUnit`

The location data are the stimuli that trigger the operation of SatZB model, so the submodule `LocUnit` of the test model focus on putting out location data from the place `Location Data` in a desired order and the time interval between two consecutive data. The order of the output data is controlled by adding an additional indicator (integer number) on each token. The time interval between two consecutive data is controlled by using the fusion place `Serial Number`. According to Fig. 5 and Fig. 6, we can see that only the received message from the module `TCC` has been processed, the integer number on the place `Serial Number` can be increased by one. Then the next location data can be put out.

- Submodule `TCC`

The task of the submodule `TCC` of the test model is to send commands (e.g., movement authorities) to the submodule `Onboard` as the model of traffic control centre does in the SatZB model, but only considering all possible commands in a simple way. In Fig. 7, when a movement request `mb1` is received by the place `Output Message`, one of the following four commands `mt1, mt2, mt3`, or `mt4` could be sent out as a response via the place `Input Message`, and which one to be sent is random; if the cancellation of emergency stop request `mb5` is received, then either `mt4` or textttmt5 will be sent out. It describes a situation that if more than one command could be sent out (to the on-board subsystem), then all the possible commands have the same possibilities to be delivered.

B. Test Case Generation

A test case is a specification, which consists of input and expected output. The input part of a test case is called test data [4]. In general, test cases will also include additional information such as descriptions of execution conditions or applicable configurations.

1) Test Data: From the architecture of the test model and the specification of the SatZB model as well, the input data of a test case is the location data assuming that the model of the traffic control centre is fully reliable (in fact, this is easy to be ensured in the test model, see Fig. 7). The location data represent the location of the train on the rail track, which is divided into block sections. Therefore, the border of the block sections and the MA points are of interest. As a first model for the SatZB, we assume that the MA point of each block section is identical with the border of each block section. Then two location data `Pt` and `notPt`, representing the train is at the position of a MA point and not a MA point respectively, are sufficient for the system model.

2) Expected Output: To satisfy the test coverage of all the scenarios and scenario sequences, which indicates the coverage of the whole requirement specification of the system, the reachability graph that generated by state space analysis with the CPN Tools [6] is used to generate the expected output. All possible paths of reachable states could be identified if the state space is finite. Each of such paths corresponds to a possible sequence diagrams or a possible combination of sequence diagrams. Accordingly, the expected output of a test case (i.e., tokens on specific places) can be extracted from a possible path. For the given example, Nevertheless, considering SCE_STANDBY as the initial scenario net of the on-board subsystem model, at least three location data are needed (e.g., in a sequence of `notPt, Pt, notPt`) to be able to activate all transitions in the test model. Obtaining the sequence of test data from the module `LocUnit` by initializing the place `GPS Data` with a *multiset* [2] 1'(1,"notPt") ++1'(2,"Pt")++1'(3,"notPt"), 24 paths are identified form the reachability graph, which has 66 nodes and 84 arcs. In other words, 24 expected outputs (e.g. tokens on place `Output Message` and `Input Message`) are derived for testing the on-board subsystem model.

IV. Testing with SPENAT

A. Introduction of SPENAT

The SPENAT ([12], [13]) notation is built upon safe place transition nets (p/t net) [10] and concepts of high level Petri nets [2], [7], [8]. Using SPENAT it is possible to use external and parameterised signal/events as transition triggers (in contrast to STG [9], SIPN [10], IOPT [11]). Thanks to this feature it is much easier to model the required behaviour of an open and reactive system with a Petri net. Also, the mapping of existing models onto a Petri net should be possible in an easy and intuitive way. For a SPENAT it is possible to use attributes with arbitrary data type for handling internal data states. Also ports and signals with parameters as external event triggers for the transitions can be declared. With these features one can model a system and/or component behaviour with a SPENAT like a well established input/output box. An example of a declaration of a SPENAT as a Petri net reacting on externally parameterised signals is presented in Fig. 8. This SPENAT has two transitions where transition t_2 can only fire after transition t_1 and the guard of t_2 depends implicitly on the value of the parameter x of the trigger event of t_1. If transition t_2 of the SPENAT of Fig. 8 fires, it is clear that the parameter x of the external event $ev1(int\ x)$ must be 1. This value is a result of the guard of t_1 $(msg.x < 2)$, the

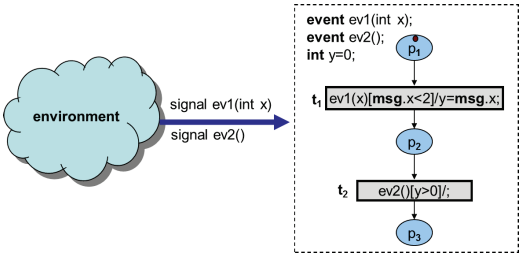

Fig. 8. SPENAT with externally parameterized signals/events

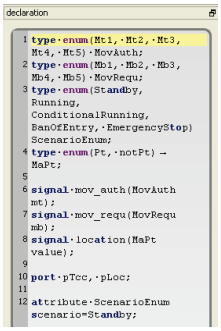

Fig. 9. The declarations of the SPENAT model

effect of t_1 $(y = msg.x)$, and the guard of t_2 $(y > 0)$. The keyword msg is a reference to the respective trigger event of the transition. In this case the value 1 is the only valid value for parameter x of the trigger event $ev1(int\ x)$ so t_2 can fire. For any other value of x, transition t_1 cannot fire (see guard $msg.x < 2$) otherwise the SPENAT of Fig. 8 would be in a deadlock after t_1 has been fired.

There are methods for verification and test generation based on SPENAT models. These methods use well known Petri net methods, especially the construction of the (complete) prefix of the Petri net unfolding ([9]). The use of UML state machines as a specification model in order to generate test cases is also possible with the use of SPENAT based methods. Therefore a mapping of a UML state machine on a SPENAT is necessary. These methods are described in more detail in [12] (see also [13]).

B. Test Model for the On-board Subsystem Model

To develop a SPENAT model for testing the on-board subsystem model of SatZB, a simple SPENAT modeller was used (see Fig. 10). With this tool it is possible to declare SPENAT properties such as signals with parameters, ports, attributes, data types as text with an easy and simple textual declaration notation (see Fig. 9). For example, parameter mt of the event $mov\_auth(MovAuth, mt)$ is a variable with the type of *enumerable*. Ports $pTCC$ and $pLoc$, representing the interfaces of the on-board subsystem model that interact with the model of the traffic control centre and the localisation unit respectively, are used to trigger the transitions in Fig. 10. Attribute $scenario$ indicates the current scenario net of the on-board subsystem model. For the behaviour modelling, a simple graphical notation can be used. With this graphical notation, a normal Petri net can be modelled. Besides this there are some simple extensions with respect to normal Petri net representations for convenience reasons. Transitions between two places with one pre- and one post- place are represented by a simple arc. *Junctions* for a better representations of possible branches can also be used. They have the same syntax and semantics like junctions of a UML state machine [14]. The SPENAT model (see Fig. 10) for testing the on-board subsystem model has been converted from Fig. 4. It has 3 places, 3 junctions and 13 transitions. For the modelling of the necessary properties 4 $enum$ types, 3 input signals with parameters, 2 ports and 1 attribute for handling the scenario nets of the on-board subsystem model are textually declared (see Fig. 9).

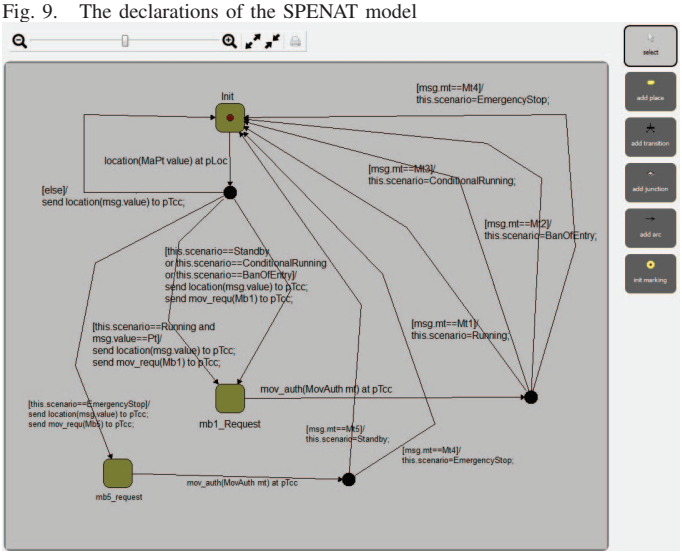

Fig. 10. Screenshot of the SPENAT model for testing on-board subsystem model

C. Test Case Generation

The generation of test cases can be controlled by the use of test coverage parameters. With the highest test coverage criteria "All Paths" for each executable path of the SPENAT one test case is created. For the identifying of all executable paths, the complete prefix of the SPENAT unfolding is created. The complete prefix of the unfolding of a Petri net includes all possible markings (states) [9] and therefore it also includes all possible alternating processes or executable paths of the relevant Petri net. The complete prefix of the SPENAT specification model of the on-board module has 64 conditions, 32 prefix events and 19 alternating processes. Therefore for the on-board module 19 test cases were created. The calculation time for the complete prefix construction took 0.094s or 94 ms. In Fig. 11 one generated test case is illustrated as a sequence diagram, whereby for each port one lifeline was inserted. The generated test cases have the same abstract level as the specification model (CPN model and SPENAT model). For the using in real world test sessions, a transformation from the abstract, logical level to a practical usable level must be realised.

V. Conclusion

Aiming at automatic test generation and test coverage control, two model-based approaches using CPNs and SPENAT

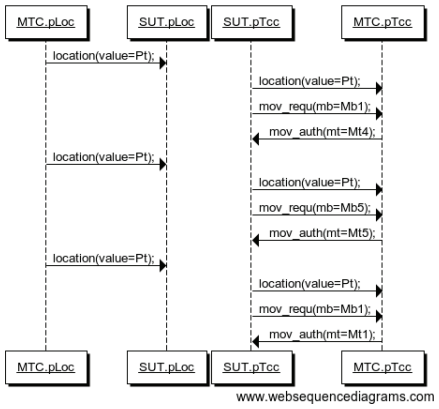

Fig. 11. A generated test case with the on-board subsystem model as SUT

TABLE I. STATISTIC DATA OF CPN AND SPENAT MODELS

Test model		CPN model		SPENAT model	
Scenarios		STANDBY, RUNNING, EMERGENCY STOP	STANDBY, BAN OF ENTRY, CONDITIONAL RUNNING, RUNNING, EMERGENCY STOP	STANDBY, RUNNING, EMERGENCY STOP	STANDBY, BAN OF ENTRY, CONDITIONAL RUNNING, RUNNING, EMERGENCY STOP
Places		16	16	3	3
Transitions		14	18	11	13
Test cases		8	24	7	19
Reachability graph	Nodes	39	66	-	-
	Paths	8	24	-	-

respectively are proposed to test the `On-board` subsystem model of an advanced satellite-based train control system. The test model developed with CPNs is a closed system including the environments that interact with the on-board subsystem model, while the test model developed with SPENAT is an open system whose transitions are triggered by external events with attributes.

For the CPN based approach, test cases are derived from the reachability graph that generated by the CPN Tools, and the test cases cover all functional requirements. As to the SPENAT based approach, test cases are generated in a similar manner but with the use of a (complete) prefix of the SPENAT unfolding and not the reachability graph. The possibility of using external signals with parameters as transition triggers are realised by a special marking representation of the SPENAT. Therefore open systems can be modelled and also the state space is much smaller than for Petri nets with the same properties and complexity. For the demonstration of the SPENAT based test generation method, an academic implementation was used. Table I presents some statistic data of the CPN model and SPENAT model.

We can see that testing with CPNs has the challenge of state space explosion problem. For example, when we shift the considering scenario from STANDBY, RUNNING, and EMERGENCY STOP to STANDBY, BAN OF ENTRY, CONDITIONAL CONDITIONAL RUNNING, RUNNING, and EMERGENCY STOP, the nodes of the reachability graph will grow from 39 to 66, and the paths of the reachability graph from 8 to 24. Testing with SPENAT could establish a model with less places and transitions. when the test object is simple enough, in this case few scenarios are under consideration, the number of test cases generated by both approaches are closed. However if the test object is much more sophisticated, e.g. more scenarios are defined in this case, then huge number of test cases will generated by the CPN based approach while the number of test cases generated with SPENAT approach is in a reasonable scope. Nevertheless, the CPN based approach has the advantage of saving the effort of constructing the test environment.

In conclusion, both CPNs and SPENAT are high-level Petri nets, and their state spaces are represented by reachability graph and (complete) prefix of the Petri net unfolding, respectively. The reachability graph and (complete) prefix of the Petri net unfolding are used to generated test cases in CPN based and SPENAT based generation techniques, so these two techniques are essentially similar. However, if the number of states of the system is large and many of the states are concurrent, then SPENAT based approach is preferred.

REFERENCES

[1] CENELEC EN 50126, “Railway applications - The specification and demonstration of Reliability, Availability, Maintainability and Safety (RAMS)”, 1999.

[2] K. Jensen, and L. Kristensen, *Coloured Petri Nets: Modelling and Validation of Concurrent Systems*, Springer, 2009.

[3] L. Jansen, E. Schnieder, J. Padberg, H. Ehrig, and R. Heckel, *Cooperability in train control systems: Specification of scenarios using open nets*. Journal of Integrated Design and Process Science, 5(1):3-21, 2001.

[4] A. Pretschner and M. Leucker, “Model-Based Testing - A Glossary,” *Model-Based Testing of Reactive Systems*, Lecture Notes in Computer Science, Springer, pp. 257-279, 2005.

[5] D. Wu, E. Schnieder, D. Lu, and H. Manz, “Realistic Modelling of Train Control System with Coloured Petri Nets,” in *13th IFAC Symposium on Control in Transportation Systems*, Sofia, Bulgaria, September 12-14, 2012.

[6] CPN Tools Homepage, http://cpntools.org.

[7] E. Best, H. Fleischhack, W. Fraczak, R. Hopkins, H. Klaudel, and E. Pelz, “A Class of Composable High Level Petri Nets,” in *ATPN’95*, Springer-Verlag, 1995.

[8] ISO/IEC 15909-1,“ Software and System engineering - High-level Petri nets - Part 1:Concepts, definitions and graphical notation,” 2004.

[9] V. Khomenko, *Model Checking Based on Prefixes of Petri Net Unfoldings*. PhD thesis, University of Newcastle, 2003.

[10] G. Frey, *Design and formal Analysis of Petri Net based Logic Controllers*. Dissertation, Aachen, Shaker Verlag, 2002.

[11] L. Gomes, J. Barros, A. Costa, and R. Nunes, “The Input-Output Place-Transition Petri Net Class and Associated Tools,” in *Proceedings of the 5th IEEE International Conference on Industrial Informatics (INDIN07)*, Vienna, Austria, 2007.

[12] J. Krause, *Testfallgenerierung aus modellbasierten Systemspezifikationen auf der Basis von Petrinetzentfaltungen*. Dissertation, Universität Magdeburg, Institut fr Automatisierungstechnik, Shaker Verlag, Aachen, 2012.

[13] J. Krause, E. Hintze, S. Magnus, and Ch. Diedrich, “Model Based Specification, Verification, and Test Generation for a Safety Fieldbus Profile,” in *SAFECOMP*, 2012.

[14] Object Management Group, “Unified Modeling Language, Superstructure Specification,” 2009. http://www.uml.org/.