Hindawi Publishing Corporation
Journal of Applied Mathematics
Volume 2014, Article ID 645235, 10 pages
http://dx.doi.org/10.1155/2014/645235

Research Article

Test Purpose Oriented I/O Conformance Test Selection with Colored Petri Nets

Jing Liu, Xinming Ye, and Jiantao Zhou

College of Computer Science, Inner Mongolia University, Hohhot 010021, China

Correspondence should be addressed to Jing Liu; liujing@imu.edu.cn

Received 28 January 2014; Accepted 14 March 2014; Published 14 April 2014

Academic Editor: Guiming Luo

This paper proposes an input-output conformance (IOCO) test selection method directed by test purpose model specified with colored Petri nets (CPN). Based on conformance testing oriented CPN models for specifying software functional behaviors and specific test purposes, respectively, feasible test cases are generated, guided by the CPN based IOCO relation, using synchronized model simulation with the proof of the soundness of test generation and the coverage towards test purposes. This test selection method integrates the merits the IOCO testing theory and the CPN modeling synergistically and is applied as a novel and applicable test selection method for actual testing practice of large-scale software systems. As the synchronized model simulation with two CPN models is irrespective of their model scale, the effectiveness and practicability of our test selection method are enhanced with scalability.

1. Introduction

Software systems running based on network environment are ubiquitous. It is quite significant to validate their functional correctness. Conformance testing [1] just aims at checking whether the software implementation conforms to its functional specification. Therefore, conformance testing is indispensable in such software system validation process. Nowadays, model based testing (MBT) technology is introduced and well developed to promote the efficiency and effectiveness of conformance test generation [2–6]. It allows for generating test cases with test oracles from a formal model that specifies the software behaviors explicitly, which improves the low-level efficiency and inaccuracy of the manual test case generation process. In particular, concerning the conformance testing towards network based software systems, the well-established input-output conformance (IOCO) testing theory and technologies [6–9] are more feasible, because the IOCO relation formally defines what external output should be observed through the practical test execution and how to determine the conformance based on these observations. In our studies, network based software systems are adopted as our system under testing (SUT) and the IOCO testing theory is the most significant theoretical foundation in our testing research.

In original IOCO testing theory, labeled transition system (LTS) is utilized as its basic formal models. However, compared with LTS or other formal modeling methods such as automata or process algebra, colored Petri nets (CPN) [10] have more advantages for specifying and validating complicated functional behaviors of network software systems. First, CPN could not only specify the detailed and complicated software functionalities intuitively and hierarchically but also support visible simulation and efficient analysis to validate the correctness of software behaviors. Validated models are indispensable for successful application of MBT approaches. Second, CPN models can execute dynamically, which is directed by the data-dependent control flow of system behaviors. Generating test cases by such model simulation process, they certainly contain actual test data and test oracles, so they are quite feasible for guiding practical test execution.

We have proposed the introductory idea of CPN model based IOCO test generation approach in our conference paper [11]. However, that paper just focuses on elementary test case generation approach, and, in this paper, we focus on test case selection; that is, test purpose is considered in test case generation to improve its pertinence, and a novel test purpose model oriented IOCO test selection method is proposed. Specifically, conformance testing oriented colored

Petri nets (CT-CPN) models are used as formal models for modeling software specification, and PN-ioco relation is defined [11]. Then, we model test purposes as CT-CPN models, and test cases are generated using synchronized model simulation between a specification CT-CPN model and a test purpose CT-CPN model. Besides, we prove the soundness of test generation; that is, as long as the implementation fails one test case, it will definitely not conform to its specification. We also prove the coverage towards test purpose; that is, generated test cases should cover and only cover functional behaviors which are specified in test purpose models. We finally apply the test selection method into a file sharing software system to illustrate its feasibility and effectiveness.

CPN model based IOCO test selection has several advantages, compared with current IOCO test selection method in literatures [12–14]. First, CPN models can execute dynamically, which is directed by the data-dependent control flow of system behaviors. Generating by such model simulation process, test cases certainly contain actual test data and test oracles, so they are quite feasible for guiding practical test execution. Second, as the synchronized model simulation with two CPN models is irrespective of their model scale, the effectiveness and practicability of our test selection method are enhanced with scalability. In a word, a CPN model based IOCO test selection method tends to be a promising testing technology to validate the correctness of reactive network software systems more efficiently and more effectively.

The paper is organized as follows. The related work and preliminaries are discussed in Section 2. The framework of our CPN model based IOCO test selection method is introduced in Section 3. The formal definition of CT-CPN models and PN-ioco relation is recalled in Section 4 as basic knowledge. Then, in Section 5, a novel test case selection algorithm is proposed using synchronized simulation technology in CPN modeling context to guarantee that all test cases are feasible for practical test execution and totally cover test purposes. In Section 6, we prove the soundness of test generation and the coverage degree towards test purposes. As a representative, we apply the test selection method into a file sharing software system and perform its actual test selection and execution procedure to illustrate the feasibility and effectiveness of our test selection method.

2. Related Work and Preliminaries

As for test case generation approaches based on CPN models, Watanabe and Kudoh [15] propose a basic test generation algorithm, which could be considered as the first step in this field. First, the reachability tree of a CPN model is constructed, and all input-output sequences from the root node to leaf nodes in this tree are traversed to form the test cases, and, then, equivalent markings in that tree are combined to construct the corresponding reachability graph, and FSM model based test case generation approaches are applied directly based on this graph. Recently, Farooq et al. [16] use random walking technology to randomly traverse the model state space to generate the test cases, where several sequential coverage criteria and concurrent coverage criteria are proposed to guide the test selection. Zhu and He [17] have proposed four specific testing strategies towards the high-level Petri nets. For each strategy, they first define a set of schemes to observe and record the testing results, and they also define a set of coverage criteria to measure the test adequacy. But, no detailed test generation algorithms are explicitly presented. We have proposed the introductory idea of CPN model based IOCO testing approach [11]. It focuses on elementary test case generation approach, and, in this paper, we focus on test selection method.

In order to promote the pertinence of testing projects, certain test selection criterion is always adopted in test generation process to produce finite and indispensable actual test cases. It is quite propitious for performing feasible testing projects under constraint of testing execution time and cost. Generally, test selection for function testing is classified as test purposed oriented methods [9, 12, 13, 18], random testing methods [8, 19, 20], property coverage based methods [21, 22], and symbolic test data selection methods [14]. However, test purposed oriented methods usually specify part of functional behaviors of software as test purpose model and make generated test cases focus on testing such specific behaviors. It is quite propitious for performing feasible conformance testing towards network based software systems with black-box and reactive behavior characteristics. In this paper, a novel IOCO test selection method is proposed using synchronized model simulation technology between two CPN models. It has high scalability for dealing with the larger software models.

CPN is advantaged for modeling and validation of systems where concurrency and communication are key characteristics. Its formal definitions are referred to in [10]. Besides, other key definitions concerning the behavior simulation of CPN models which are used in following sections are listed as follows.

Definition 1. For a CPN $= (P, T, A, \Sigma, V, C, G, E, I)$, consider

(1) *preset* and *postset* of the place or the transition:

$$\begin{aligned} &\forall p \in P : \operatorname{pre}(p) = \{t \in T \mid (t, p) \in A\}; \\ &\operatorname{post}(p) = \{t \in T \mid (p, t) \in A\}, \\ &\forall t \in T : \operatorname{pre}(t) = \{p \in P \mid (p, t) \in A\}; \\ &\operatorname{post}(t) = \{p \in P \mid (t, p) \in A\}; \end{aligned} \tag{1}$$

(2) $M \overset{\sigma}{\Rightarrow} =_{\text{def}} \exists M_i : M \overset{\sigma}{\Rightarrow} M_i$, where $\sigma = (t_0, b_0), (t_1, b_1) \cdots (t_{i-1}, b_{i-1})$, $M \xrightarrow{(t_0,b_0)} M_1 \xrightarrow{(t_1,b_1)} \cdots \xrightarrow{(t_{i-1},b_{i-1})} M_i$, if $|\sigma| = 1$, $M \overset{\sigma}{\rightarrow}$ is used instead;

(3) $\operatorname{trace}(M) =_{\text{def}} \{\sigma \in \operatorname{BE}(T)^* \mid M \overset{\sigma}{\Rightarrow}\}$;

(4) M fires $\sigma =_{\text{def}} \{M_n \mid M \overset{\sigma}{\Rightarrow} M_n, \sigma \in \operatorname{BE}(T)^*\}$;

(5) CPN is *deterministic*, if $|M$ fires $\sigma| \leq 1$;

(6) CPN has *finite output*, if $|M$ fires $\sigma| \leq n$ $(n \in N)$;

(7) CPN has *finite behavior*, if $\exists n \in N, \forall \sigma \in \operatorname{trace}(M_0) : |\sigma| < n$.

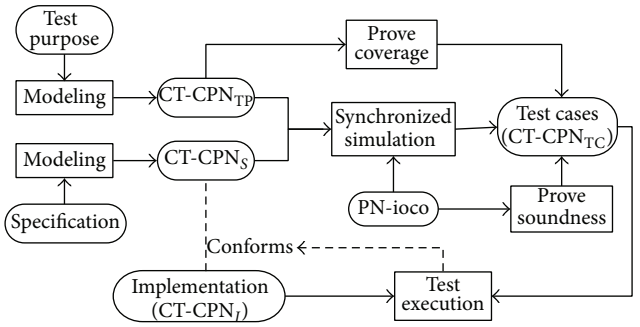

Figure 1: The framework of IOCO test selection method with CPN models.

In this definition, M stands for markings, σ stands for occurrence trace of system behavioral execution, and BE stands for binding elements with a transition and its value bindings into variables.

3. Methodology Overview

Integrating the merits of the IOCO testing theory and the CPN modeling synergistically and then constructing a novel IOCO test selection method based on test purpose CPN models are the major research goals of this paper. However, simply replacing LTS modeling with CPN modeling does not make sense. In Figure 1, we propose a framework of IOCO test selection methodology based on CPN models, which is composed of three related parts in the whole test selection process.

First, modified CPN modeling is proposed as CT-CPN models to specify key characteristics and requirements for conformance testing scenario accurately. For example, CT-CPN$_S$ models software functional behaviors according to software requirement specification, CT-CPN$_I$ models actual software implementation behaviors, CT-CPN$_{TP}$ models software functional behaviors of specific test purposes, and CT-CPN$_{TC}$ models finally generated test cases. Such CT-CPN series models explicitly specify external visible actions which are significant in practical test execution, that is, to make the most of both the place and the transition elements in CT-CPN models to distinguish visible actions from internal actions. In particular, to deal with the special output actions, such as the quiescence or deadlock [7], it introduces new kind of transitions to model them accurately. Besides, a corresponding implementation relation in the context of CT-CPN modeling is proposed as PN-ioco relation to precisely specify what it means for an implementation to conform to its functional specification. In CPN model context, software behaviors are simulated with specific system data, so we have to determine the IOCO conformance via comparing the output actions with specific data. This part of contents had been proposed in our conference paper [11], and, in order to make this paper self-contained, we will show the basic formal definition of CT-CPN model and PN-ioco relation in Section 4.

Second, based on the above CT-CPN modeling and PN-ioco relation, we need to develop a novel test selection approach with two desired goals. One is to make the test selection with high scalability for dealing with more complicated system models, and the other one is to make all test cases feasible for the practical test executions. Therefore, in Section 5, we model test purposes as CT-CPN$_{TP}$ models and then propose a novel IOCO test case selection method, where test cases are generated using synchronized model simulation between a CT-CPN$_S$ model and a CT-CPN$_{TP}$ model. When the synchronized simulation procedure terminated, a final test case is produced and specified as a CT-CPN$_{TC}$ model. As the synchronized model simulation is irrespective of model scale, its effectiveness and practicability tend to be enhanced with high scalability.

Third, in Section 6, we prove the soundness of test generation; that is, as long as the software implementation fails one test case, it will definitely not conform to its functional specifications. We also prove the coverage towards test purposes; that is, generated test cases should cover and only cover functional behaviors which are specified in test purpose models. In this way, under constraint of testing execution time and cost, we could generate finite and indispensable actual test cases to promote the pertinence of testing projects.

Developing a CPN model based IOCO test selection method is challenging but quite promising. It has solid theoretical foundation and bright application prospect, so it tends to be used as a competent and effective conformance testing technology to validate the correctness of network based software systems.

4. CT-CPN Modeling and PN-ioco Relation

4.1. Specification Modeling

Definition 2. A CT-CPN$_S$ is a triple CT-CPN$_S$ = (CPN, P_S, T_S):

(1) CPN is a basic colored Petri nets model;

(2) $P_S = P, P_S = P_S^O \cup P_S^E$: P_S^O is the set of *observable places*; P_S^E is the set of *internal places*; $P_S^O \cap P_S^E = \phi$;

(3) $T_S = T, T_S = T_S^I \cup T_S^O \cup T_S^E$: T_S^I is the set of *input transitions*; T_S^O is the set of *output transitions*; T_S^E is the set of *internal transitions*; $T_S^I \cap T_S^O = T_S^I \cap T_S^E = T_S^O \cap T_S^E = \phi$;

(4) CT-CPN$_S$ has finite output; that is, $|\text{trace}(M_0)| \leq n$ $(n \in N)$;

(5) CT-CPN$_S$ does not have infinite sequences of internal actions; that is, $\neg\exists M : M \stackrel{\sigma}{\Rightarrow} M, \sigma \in \text{BE}(T_S^E)^*$.

In the CT-CPN$_S$ modeling, token data in the observable places could present the externally observed data, so an observable place is always the postset of an input transition or an output transition to display what data should be observed after executing those external transitions. The input transition models input actions that accept input data provided by testers and the output transition models output actions that produce visible output observations. Thus, observable places and input/output transitions are used together to explicitly

specify external visible behaviors of a certain system. Besides, the internal transitions and internal places could represent internal and unobservable execution of system behaviors. Models must not have loops of internal transitions, which will make system implementations have no response to help us to distinguish this scenario from the deadlock.

4.2. Implementation Modeling. As system implementations are actual physical thing, that is, software, hardware, or a hybrid system, rather than formal objects, a test hypothesis [7] assumes that every system implementation corresponds to an a-priori formal model, but these formal models cannot be explicitly constructed. Therefore, CT-CPN$_I$ is just proposed to formally specify system implementations.

Definition 3. A CT-CPN$_I$ is a triple CT-CPN$_I$ = (CPN, P_I, T_I):

(1) CPN is a basic colored Petri nets model;

(2) $P_I^O = P_S^O$ and $T_I^I = T_S^I$, $T_I^O = T_S^O$ the implementation models and the specification model of the same system having the same observable places and input/output transitions;

(3) $\{\delta\} \subset T_I$: δ is the *suspension transition*, and $M \xrightarrow{\delta} M$.

Any possible test output, such as real data, deadlock, or quiescence should be managed in CT-CPN$_I$ modeling. In particular, the quiescence represents a scenario where software implementations have no visible outputs because they wait for input action to trigger following executions. Producing quiescence is a kind of special output action, which is modeled as the suspension transition δ. Firing a suspension action indicates that an implementation stays in the same state and needs input data as a trigger to continue executing.

4.3. PN-ioco Relation. In context of CT-CPN modeling, conformance relation should be determined according to specific data in CT-CPN models, so PN-ioco relation is defined.

Definition 4. PN-ioco is a binary relation with $ss \in$ CT-CPN$_S$ and $ii \in$ CT-CPN$_I$:

ii PN-ioco $ss =_{\text{def}} \forall\sigma \in \text{SPtrace}(M_S)$: outtoken($M_I$ fires σ) = outtoken(M_S fires σ).

(1) Consider $\text{SPtrace}(M_S) =_{\text{def}} \{\sigma \in (\text{BE}(T_S) \cup \delta)^* \mid M_S \overset{\sigma}{\Rightarrow}\}$; it enumerates all traces of the model ss, including the suspension transitions. M_S and M_I are initial markings, respectively.

(2) Consider $\text{outtoken}(M) =_{\text{def}} \{M(P) \mid P \in P_S^O \cup P_I^O\}$; it represents the observable output token data. In the model ss, it records the token data of current observable places under a specific marking, while, in the model ii, it just corresponds to the actual observable output data produced by the system implementations during the test execution.

Guided by the PN-ioco definition, the conformance is determined by comparing token data in the observable places along a specific SPtrace with the actually observed output from the implementation. If the actual observed output data are different from what are prescribed in the ss model, we could conclude with non-conformance decision. The equivalence of two outtoken sets indicates that all prescribed observations should be actually observed in the practical test executions; that is, prescribed functionalities must be completely implemented. Therefore, the implementation that has valid but partial functionalities will not be said to conform to its specification model.

5. Synchronized Simulation Based Test Selection

In Section 3, we mention that, based on CT-CPN modeling and PN-ioco relation, a novel test selection method should be developed with two desired goals. One is to make such test selection with high scalability for dealing with more complicated system models, and the other one is to make all test cases feasible for practical test executions. Accordingly, we develop a test purpose oriented IOCO test selection method to meet these two goals, where test cases are generated through synchronized model simulation between a CT-CPN$_S$ model and a CT-CPN$_{\text{TP}}$ model. When synchronized simulation procedure terminates, a final and feasible test case model is produced and specified as CT-CPN$_{\text{TC}}$.

The intuitive idea of our test selection method is essentially a synchronized traversal between a CT-CPN$_S$ model and a CT-CPN$_{\text{TP}}$ model. It is performed by model simulation execution under given initial markings in these two models. Specifically, each given initial marking in a CT-CPN$_S$ model and a CT-CPN$_{\text{TP}}$ model could conduct a synchronized simulation between these two models once. During following synchronized simulation steps, enabled transitions in both models are capable of firing sequentially in a synchronized way. However, CT-CPN$_{\text{TP}}$ model is responsible for choosing which execution sequence should be cover, and actual test sequences and the data-dependent test oracles are all generated based on CT-CPN$_S$ model. If both models have no further enabled transitions, synchronized model simulation procedure will terminate, and a corresponding test case model is finally generated and specified as a CT-CPN$_{\text{TC}}$ model, while, in context of LTS based test selection in original IOCO testing theory, synchronous product of two LTS model is performed. However, with expansion of model scale, synchronous product operation cannot be executed accurately; thus such LTS based test selection approach fails to support testing large-scale software systems.

In the following subsections, we first propose the formal definitions of test purpose and test case in CT-CPN modeling context. Then, a detailed test selection algorithm is developed with several test generation rules towards different kinds of model transitions. Finally, we adopt a file sharing software system as a representative to demonstrate practical test selection and test execution procedures.

5.1. Test Purpose Modeling. Test purpose is utilized to specify parts of software functional behaviors; for example, in the network based software systems, sending request packets, receiving data packets, or updating local key data structure could be modeled as a test purpose each. In order to promote the pertinence of testing large-scale software systems, test purpose models should participate in test generation process, since it can constrain the scope of test generation; that is, it just selects finite and indispensable test cases to only test expected partial software behaviors under constraint of testing execution time and cost. Obviously, a CT-CPN$_{TP}$ model is essentially constructed from a corresponding CT-CPN$_S$ model, because it just specifies parts of software behaviors which need to be tested. Therefore, we propose the formal definition of CT-CPN$_{TP}$ model based on the CT-CPN$_S$ definition.

Definition 5. A CT-CPN$_{TP}$ is a triple (CT-CPN$_S$, P_{TP}, Σ_{TP}):

(1) CT-CPN$_S$ is a CT-CPN$_S$ model;

(2) $P_{TP} = P_S \cup \{ph\}$, $\{ph\}$ is the *coverage verdict place*, and only *covered* token can appear in this place;

(3) $\Sigma_{TP} = \Sigma_S \cup \{covered\}$.

In CT-CPN$_{TP}$ model, *ph* is always the postset of an output transition and used as the termination place in that model. If a *covered* token appears in *ph* place, it indicates that, based on a given initial marking, a generated test sequence just corresponds to an execution path of behaviors specified in the test purpose model. Thus, we need to add some necessary guard functions towards input or output transitions in the test purpose model to guarantee such behavior path be executed as expected. It is well demonstrated in Section 5.4 through the practical test selection to a file sharing software system.

5.2. Test Case Modeling. Several modeling constraints should be fulfilled in CT-CPN$_{TC}$ modeling. CT-CPN$_{TC}$ models should have only one input transition enabled at each step. Besides, they should be deterministic and every feasible trace should have finite length; otherwise, the test execution based on this model cannot terminate in finite steps with the definite testing results.

Definition 6. A CT-CPN$_{TC}$ is a triple (CPN, P_{TC}, T_{TC}), where

(1) CPN is a basic colored Petri nets model;

(2) $P_{TC} = P$, $P_{TC} = P^I_{TC} \cup P^O_{TC} \cup P^{TO}_{TC} \cup P^V_{TC}$: P^I_{TC} is the set of *input places*; P^O_{TC} is the set of *observable places*; P^{TO}_{TC} is the set of *test oracle places*; P^V_{TC} is the set of *test verdict places*; each pair of them had no intersections;

(3) $T_{TC} = T$, $T_{TC} = T^I_{TC} \cup T^O_{TC} \cup T^\delta_{TC} \cup T^V_{TC}$: T^I_{TC} is the set of *input transitions*; T^O_{TC} is the set of *output transitions*; T^δ_{TC} is the set of *suspension transitions*; T^V_{TC} is the set of *test verdict transitions*; each pair of them had no intersections either;

(4) CT-CPN$_{TC}$ has finite behavior and is deterministic;

(5) CT-CPN$_{TC}$ has at most one input transition enabled in each step; that is, $\neg\exists M : M \xrightarrow{(t_1,b_1)} \wedge M \xrightarrow{(t_2,b_2)}$, $t_1 \in T^I_{TC} \wedge t_2 \in T^I_{TC}$.

It should be noted that, in the test verdict places, only three kinds of token data can appear, that is, *pass*/*fail*/*covered* tokens. A *pass* token indicates that current test execution step is successfully passed; a *fail* token indicates an implementation fault with current test execution step and results in the nonconformance decision; a *covered* token indicates that current test execution step covers that behaviors specified in test purpose model.

CT-CPN$_{TC}$ models could facilitate the actual test execution for their better feasibility and readability, because they not only prescribe the test sequences from the data-dependent control flow of the system behaviors but also provide necessary and definite test oracles for determining the conformance relation.

5.3. Test Selection via Synchronized Model Simulation. To develop a test purpose model oriented IOCO test selection method, we need considering simulation paths in CT-CPN$_S$ model and CT-CPN$_{TP}$ model at the same time. The reason is that the simulation in the CT-CPN$_S$ model reflects actual execution paths of a software system, including real input and output data, which is the basis of test case generation. However, the simulation in the CT-CPN$_{TP}$ model conducts to select expected execution paths among all enabled paths. Thus, test generation scope is well constrained into those software behaviors we want to test does not consider other behaviors which are not specified in test purpose models. To accomplish such goal, the CT-CPN$_S$ model and the CT-CPN$_{TP}$ model should simulate in a synchronized way. Specifically, under guidance of PN-ioco relation, given an initial marking M_S in CT-CPN$_S$ model and an initial marking M_{TP} in CT-CPN$_{TP}$ model, the IOCO test selection method selects feasible and expected test sequences from the set of SPtrace($M_S \| M_{TP}$) in order to cover behaviors specified in test purpose models; that is, corresponding enabled transitions in these two models are fired in a synchronized way according to different test generation rules towards different kinds of transitions. Meanwhile, the IOCO test selection method also presents explicit way to decide the conformance relation via test output and test oracle and indicate whether the test purpose is covered. Finally, if no further transitions could fire, that is, a termination marking is reached, the synchronized simulation procedure will terminate. If this final marking represents a valid termination of test purpose directed system behavioral execution, a final CT-CPN$_{TC}$ test case model covering specific test purpose is generated. Otherwise, if this final marking happens to stand for an invalid deadlock scenario, we need to improve the accuracy of both models and perform the synchronized simulation procedure again.

Rule 1 (synchronized firing rule to input transition). $\exists\, t \in T^I_S$ and $t \in T^I_{TP}$, $(t, b_S) \in \mathrm{BE}_S$ and $(t, b_{TP}) \in \mathrm{BE}_{TP}$ are all enabled: generating $t \in T^I_{TC}$ and $\forall p \in \{p \in P^I_S \mid p \in \mathrm{pre}(t)\} : p \in P^I_{TC}$,

keeping token data in $M(p)$, and generating a new internal place $p' \in P^E_{\mathrm{TC}} \mid p' = \mathrm{post}(t)$.

Firing (t, b_{S}) and (t, b_{TP}), that is, $M(p') = \{M \mid M(p) \xrightarrow{(t,b_{\mathrm{S}})}\}$, to accomplish a synchronized firing of input transitions.

Rule 2 (synchronized firing rule to internal transition). $\exists\, t \in T^E_S$ and $t \in T^E_{\mathrm{TP}}$, $(t, b_{\mathrm{S}}) \in \mathrm{BE}_S$ and $(t, b_{\mathrm{TP}}) \in \mathrm{BE}_{\mathrm{TP}}$ are all enabled, if these two internal transitions could be fired to produce the same token data, then just fire them to accomplish a synchronized firing of internal transitions. Otherwise, fire (t, b_{S}) time after time until $t \in T^E_{\mathrm{TP}}$; that is, certain internal transition appearing only in the CT-CPN$_{\mathrm{S}}$ model should be firstly fired several times before reaching the state that synchronized internal transition appearing in both CT-CPN$_{\mathrm{S}}$ model and CT-CPN$_{\mathrm{TP}}$ model is able to execute. However, such internal behaviors do not need to be handled by test case models.

Rule 3 (synchronized firing rule to output transition). $\exists\, t \in T^O_S$ and $t \in T^O_{\mathrm{TP}}$, $(t, b_{\mathrm{S}}) \in \mathrm{BE}_S$ and $(t, b_{\mathrm{TP}}) \in \mathrm{BE}_{\mathrm{TP}}$ are all enabled: generating $t \in T^O_{\mathrm{TC}}$ and $\forall p \in \{p \in P^O_S \mid p \in \mathrm{post}(t)\} : p \in P^O_{\mathrm{TC}}$.

If $\exists p \in \{p \in P^E_S \mid p \in \mathrm{post}(t)\}$, we generate a new internal place $p' \in P^E_{\mathrm{TC}} \mid p' = \mathrm{post}(t)$.

Firing (t, b_{S}) and (t, b_{TP}) to accomplish a synchronized firing of output transitions and constructing a test verdict unit for every $p \in P^O_{\mathrm{TC}}$ are as follows:

(i) generating $q \in P^{\mathrm{TO}}_{\mathrm{TC}}$, $t_v \in T^V_{\mathrm{TC}}$, $p_v \in P^V_{\mathrm{TC}}$: $\{\mathrm{pre}(t_v) = p \cup q\}$ and $\{\mathrm{post}(t_v) = p_v\}$; $M(q) = M(p)$, where $M(q)$ records the test oracle data with respect to p;

(ii) generating $a \in (t_v, p_v)$, $r = \mathrm{pre}(t_v)$ $(r \in P^O_{\mathrm{TC}})$, and, without loss of generality, $E(a)$ could be specified as follows: if $M(r) = M(q)$, then $1'pass + 1'covered$ else $1'fail$; that is, if observed test output in $M(r)$ is the same as the test oracle data in $M(q)$, and test purpose is definitely covered, a *pass* token and a *covered* token are both generated into the test verdict place p_v;

(iii) connecting $t \in T^O_{\mathrm{TC}}$ with newly generated internal place or existing observable place, that is, $r \in P^E_{\mathrm{TC}} \cup P^O_{\mathrm{TC}} \mid r = \mathrm{pre}(t)$ to keep the connectivity of current test case model.

Rule 4 (adding suspension transition). Consider $\exists p \in P^I_{TC}$, if quiescence is allowed, adding a suspension transition with p; that is, $t \in T^{\delta}_{\mathrm{TC}} : p = \mathrm{pre}(t) \wedge p = \mathrm{post}(t)$.

Given sets of initial markings M_{S} and M_{TP}, through applying a suitable rule of aforesaid four test generation rules step by step, CT-CPN$_{\mathrm{TC}}$ models covering specific test purpose are generated for testing corresponding software behaviors. Besides, it is guaranteed that the scenario never exists where transitions in CT-CPN$_{\mathrm{TP}}$ model are enabled but related transitions in CT-CPN$_{\mathrm{S}}$ model are not enabled, because a CT-CPN$_{\mathrm{TP}}$ model is constructed from the corresponding CT-CPN$_{\mathrm{S}}$ model. Furthermore, based on above test selection method, each SPtrace has finite length, so the test selection algorithm is terminated in finite steps with generating a CT-CPN$_{\mathrm{TP}}$ model that has finite behaviors for practical test executions.

Two more aspects should be noted. First, CT-CPN models are not modified with new kinds of model elements, and modeling constraints are just used to avoid generating infeasible traces for testing scenarios. So, the semantic rules defined in [10] are all kept in CT-CPN models; that is, we still use its original enabling rules and occurrence rules to generate CT-CPN$_{\mathrm{TC}}$ test case models. Second, this test selection approach can be applied into the hierarchical CPN model without modification. The reason comes from two aspects: first, a hierarchical CPN model could definitely be unfolded to a behavioral equivalent nonhierarchical CPN model, and, then, test case selection process just utilizes ordinary simulation techniques which could be applied into hierarchical or nonhierarchical CPN models without differences. Thus, our test selection method has better scalability to deal with most of actual software behavior models.

We could compare the computation cost in test purpose oriented test selection methods. In context of LTS, the state space produced by synchronous product of two LTS models tends to grow exponentially, which needs enormous even unpractical computation resources to generate suitable test cases. But, in context of CPN, synchronized model simulation based selection approach is irrespective of their model scale, so it just needs linear computation cost to produce feasible test cases. The effectiveness and practicability of our test selection method are enhanced with better scalability.

5.4. An Example: File Sharing Software System. We now apply the test purpose oriented IOCO test selection method into a file sharing software system to illustrate its feasibility and effectiveness. The CT-CPN$_{\mathrm{S}}$ model is presented in Figure 2, and one example CT-CPN$_{\mathrm{TP}}$ model is presented in Figure 3. Through synchronized simulating of such two models, a CT-CPN$_{\mathrm{TC}}$ model is generated and shown in Figure 4.

In this file sharing software system, peer nodes that have same functionalities could share the same resource file via network. The file is composed of several file segments which are identified as [SN, DATA]. When a peer node receives a file segment request (req), it searches such data segment from its data segment sets (DATASET) according to the segment number (SN) in request packet. If this peer node gets the requested data segment (DATAENTRY), it immediately sends the segment to the requesting peer node. While as a peer node receives a requested file segment, it stores this segment data into data segment sets firstly and goes on requesting new file segments which this peer does not have yet. In its CT-CPN$_{\mathrm{S}}$ model, *rv*/*sd*/*set* are observable places. *recv* is an input transition to specify the receiving of segment request or data. *send*/*newreq* are output transitions to specify sending segment data, storing segment data and requesting new file segment, respectively. For example, If *send* fires, we could observe which packet is sent according to the tokens in *sd*. The rest are internal places and transitions. In particular, the place *sp* stores the total set of segment number beforehand

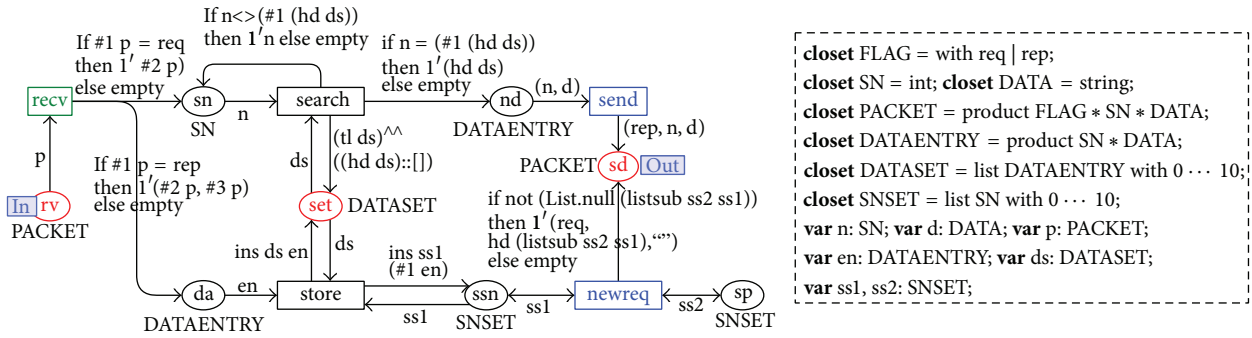

Figure 2: The CT-CPN$_S$ model of the file sharing software system.

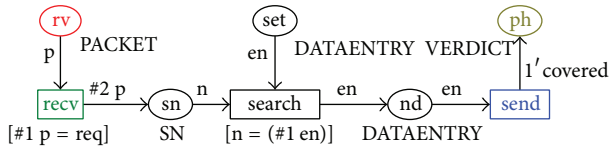

Figure 3: CT-CPN$_{TP}$ model of file sharing software system.

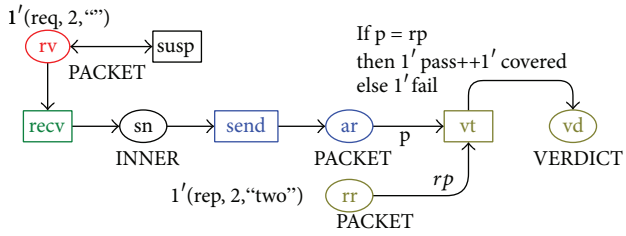

Figure 4: CT-CPN$_{TC}$ model of file sharing software system.

to help to choose which segment should be requested. In this way, necessary external behaviors of a system for its conformance testing are modeled accurately. However, in system modeling practice, the selection of observable places should also consider the actual observation points in the actual test execution, such as the points of control and observation [1].

In Figure 3, we present a test purpose CT-CPN$_{TP}$ model as a representative. This test purpose focuses on the function that whether a peer node could get the requested date segment from its data segment sets according to the segment number (constrained by the guard function $[n = (\#1\text{en})]$), when it receives a file segment request (constrained by the guard function $[(\#1p) = \text{req}]$). If a *covered* token appears in *ph* place, it indicates that generated test sequence just covers the behaviors specified in this test purpose model. The CT-CPN$_{TC}$ model in Figure 4 is used to check whether a peer node gets the requested date segment from its data segment sets according to the segment number. The detailed test selection procedure is discussed as follows.

(i) Initial marking is assigned as $M_S(rv) = \{1'(\text{req},2,\text{“”})\}$, $M_S(sp) = \{[1,2,3,4]\}$, $M_S(set) = \{[(1,\text{“one”}), (2,\text{“two”})]\}$; $M_{TP}(rv) = \{1'(\text{req},2,\text{“”})\}$, $M_{TP}(set) = \{[(2,\text{“two”})]\}$.

(ii) Rule 1 is applied to deal with input transition *recv*. This transition could be fired as $(recv, \{p = (\text{req},2,\text{“”})\})$ in a synchronized execution in both models. A new internal place *sn* is added as the postset of *recv* and $M_{TC}(rv) = \{1'(\text{req},2,\text{“”})\}$.

(iii) Rule 2 is applied to deal with internal transition *search*. In the CT-CPN$_S$ model, this transition relates two enabled scenarios, that is, $(search, \{n = 2, \text{ds} = [(1,\text{“one”}),(2,\text{“two”})]\})$ and $(search, \{n = 2, \text{ds} = [(2,\text{“two”}),(1,\text{“one”})]\})$, while, in the CT-CPN$_{TP}$ model, it only relates one enabled scenario, that is, $(search, \{n = 2, \text{en} = (2,\text{“two”})\})$. Thus, we fire *search* in CT-CPN$_S$ model twice, where at the first time it is fired independently, and in the second time it is synchronized fired with that in the CT-CPN$_{TP}$ model. Consider $M_S(nd) = M_{TP}(nd) = \{(2,\text{“two”})\}$.

(iv) Rule 3 is applied to deal with output transition *send*. This transition could be fired as $(send, \{n = 2, d = \text{“two”}\})$ and $(send, \{\text{en} = (2,\text{“two”})\})$ in a synchronized execution in each model and get $M_S(sd) = \{(\text{rep},2,\text{“two”})\}$, $M_{TP}(ph) = \{covered\}$, which indicates that the test purpose covered sequence is executed. The output transition *send* and its postset observable place *ar* are both generated in CT-CPN$_{TC}$ model. Furthermore, its test oracle place *rr*, test verdict transition *vt*, and test verdict place *vd* are all generated as an integrated test verdict unit for validating whether the actual output data is the same with the test oracle data. Consider $M_{TC}(rr) = M_S(sd) = \{(\text{rep},2,\text{“two”})\}$. Finally, we need to relate the output transition *send* with internal place *sn*, that is, $\text{pre}(send) = sn$, to keep the connectivity of generated CT-CPN$_{TC}$ model.

(v) Rule 4 is applied to add a suspension transition towards the input transition *rv*, which allows for waiting to *send* the data packet in a quiescence scenario.

Using above test case model, we perform actual test executions to further illustrate feasibility and effectiveness of our test selection method. We program eight software

Table 1: System implementations and test results.

(a)

Software implementations		
Number	Type	Description
i1	Totally correct	Implementing all functions correctly
i2	Faulty	Error on sending request packet
i3	Faulty	Error on matching segment number
i4	Faulty	Error on storing segment data
i5	Faulty	Error on calculating the new segment number
i6	Faulty	Error on parsing received segment packet
i7	Partially correct	Not supporting segment retrieval
i8	Partially correct	Not supporting segment storage

(b)

Testing results							
i1	i2	i3	i4	i5	i6	i7	i8
Pass	Fail	Fail	Pass	Pass	Fail	Fail	Pass

implementations of the file sharing system with preinjected errors to act as SUT.

In Table 1, software implementations description and testing results are all listed. (1) i1 passes this test case, so it conforms to the specification model in Figure 2. (2) i2, i3, and i6 have fatal errors, respectively, which this test purpose just covers, so they do not pass this test case where *fail* token appears in test case executions. (3) i4 and i5 pass the test case, but the fact is that error behaviors in i4 and i5 are not tested at all by this test case. Test purpose model in Figure 3 does not contain such behaviors; thus definitely the generated test case model does not aim to test these implementation errors. (4) i7 only implements partial functions; that is, it does not support segment retrieval. However, this function is just to be tested by this test case, so i7 does not pass as we do not observe the output segment data packet in actual test execution. Compared with basic IOCO relation definition, i7 does not conform to the specification according to PN-ioco relation, so partially correct implementations are no longer determined with conformance. (5) i8 passes this test case, because the test case does not touch such segment storage functionality.

From the above analysis, we could see that test case models generated using our test selection method are quite feasible for guiding actual test execution intuitively and also effective for finding various implementation faults. According to testing results, conformance relation between a specific implementation and its specification model could be accurately determined.

6. Proof of Soundness and Test Purpose Coverage

The conformance relation between a software implementation *ii* and its specification model *ss* is determined through test executions, specified as *ii* PN-ioco *ss* $\Leftrightarrow$ *ii pass* T_S. If all test cases in completer set T_S are passed, "*ii* PN-ioco *ss*" is consequently determined. However, in practical conformance testing, generating all test cases in the T_S is almost infeasible. Moreover, conformance testing just aims to find nonconformance faults rather than to completely prove the conformance. Thus, a weaker requirement is usually considered; that is, as long as the implementation does not pass one test case, it definitely does not conform to its specification. This weaker requirement corresponds to the left-to-right implication of *ii* PN-ioco *ss* $\Leftrightarrow$ *ii pass* T_S and is referred as the soundness of test case generation approach.

Theorem 7. *Let ss* $\in$ *CT-CPN*$_S$ *be a specification model, and tp* $\in$ *CT-CPN*$_{TP}$ *a test purpose model; let* T_S *be a complete set of test cases that generated from ss and tp with our test selection algorithm, that is, TestSel; let CT-CPN*$_S \times$ *CT-CPN*$_{TP} \rightarrow$ *CT-CPN*$_{TC}$ *be the test case selecting function that satisfies TestSel*$(ss, tp) \subseteq T_S$*; then TestSel is sound for ss with respect to PN-ioco.*

Proof. Supposing $\exists ii \in$ CT-CPN$_I$ with corresponding $ss \in$ CT-CPN$_S$, $\exists tp \in$ CT-CPN$_{TP}$ and $\exists tt \in$ *TestSel(ss,tp)* satisfying *not* (*ii pass tt*) and *ii* PN-ioco ss, then

not (ii pass tt)

$\Rightarrow \exists e \in M(p), p \in P_{TS}^{V}, e \in \{fail\}$; [a *fail* token appears in the test verdict place p]

$\Rightarrow \exists r \in P_{TS}^{O} \wedge \exists q \in P_{TS}^{TO}, M(r) \neq M(q)$; [token data in the observable place r and its coupled test oracle place q are different]

$\Rightarrow \exists \sigma \in$ SPtrace(M_S): outtoken(M_I fires σ) $\neq$ outtoken(M_S fires σ), where $M(r) \subseteq$ (M_I fires σ) and $M(q) \subseteq$ (M_S fires σ). [trace σ results in unexpected output observations in the practical test execution].

Obviously, there exists a contradiction with the assumption *ii* PN-ioco *ss*. Therefore, we could conclude that if one test case does not pass, that is, *not* (*ii pass tt*), then, *ii* PN-ioco *ss* does not hold definitely; that is, *TestSel* is sound for *ss* and *tp* with respect to PN-ioco. However, it should be noted that *TestSel* is not empty since SPtrace is always produced as a specific initial marking is assigned in actual *ss* and *tp* models. □

It should be noted that as CT-CPN$_{TP}$ model is constructed from its corresponding CT-CPN$_S$ model, when valid specific initial markings are assigned in actual CT-CPN$_S$ model and CT-CPN$_{TP}$ model, at least one SPtrace exists definitely, so at least one test case model is generated. Given sets of initial markings, several test case models covering specific test purposes are generated consequently. That is, the special case where empty set of *TestSel(ss,tp)* tends to be sound never exists.

Based on the guarantee that test case selection is sound, we need further guarantee that test selection should cover test purposes; that is, any passed test case is definitely testing

and only testing those software behaviors which are specified in corresponding test purpose models. The coverage towards test purpose is formally described as *cover-pass* relation as follows:

$\exists ii \in \text{CT-CPN}_I$ with related $ss \in \text{CT-CPN}_S$, $\exists tp \in \text{CT-CPN}_{\text{TP}}$, $\exists tt \in T_S$(a complete set of test cases generated from *ss* and *tp*):

ii cover-pass tt $\rightarrow$ *ii exhibit tp:*

ii cover-pass tt $=_{\text{def}}$ *ii pass tt* and $\{covered\} \in M(P^V_{\text{TC}})$;

ii exhibit tp $=_{\text{def}} \forall \sigma \in \text{SPtrace}(M_{\text{TP}})$: outtoken$(M_I$ fires $\sigma) \supseteq$ outtoken$(M_{\text{TP}}$ fires $\sigma)$.

From the angle of converse negative proposition in above relation, we could conclude that if a system implementation does not perform the behaviors which are specified in a test purpose model, it must not pass any test case generated using such test purpose model.

Theorem 8. *Let $ss \in$ CT-CPN$_S$ be a specification model and $tp \in$ CT-CPN$_{TP}$ a test purpose model; let T_S be a complete set of test cases that generated from ss and tp with our test selection algorithm, that is, TestSel, let CT-CPN$_S \times$ CT-CPN$_{TP} \rightarrow$ CT-CPN$_{TC}$ be the test selecting function that satisfies TestSel(ss, tp) $\subseteq T_S$; then TestSel covers behaviors in tp model.*

Proof. $\exists ii \in \text{CT-CPN}_I$ with corresponding $ss \in \text{CT-CPN}_S$, $\exists tp \in \text{CT-CPN}_{\text{TP}}$ and $\exists tt \in$*TestSel(ss,tp)*:

ii cover-pass tt

$\Rightarrow$ *ii pass tt* $\wedge \exists p \in P^V_{\text{TC}}, \{covered\} \in M(p)$

$\Rightarrow \forall p' \in P^V_{\text{TC}}, \{fail\} \notin M(p') \wedge \exists p \in P^V_{\text{TC}}, \{covered\} \in M(p)$ [only *pass* and *covered* token data can appear in the test verdict place]

$\Rightarrow \forall \sigma \in \text{SPtrace}(M_S \| M_{\text{TP}})$: outtoken$(M_{\text{TP}}$ fires $\sigma) \subseteq$ outtoken$(M_S$ fires $\sigma)$ and outtoken$(M_S$ fires $\sigma) =$ outtoken$(M_I$ fires $\sigma)$

$\Rightarrow \forall \sigma \in \text{SPtrace}(M_S \| M_{\text{TP}})$: outtoken$(M_{\text{TP}}$ fires $\sigma) =$ outtoken$(M_I$ fires $\sigma)$ [trace σ results in expected output observations in the practical test execution].

□

7. Conclusion

To make the best of advantages of the IOCO testing theory and the CPN modeling, we integrate them directly to develop a novel test purpose model oriented IOCO test selection method. Based on conformance testing oriented CPN models for specifying software functional behaviors and specific test purposes, respectively, guided by the CPN based IOCO relation, feasible test cases are generated using synchronized model simulation with the proof of the soundness of test generation and the coverage towards test purposes. Throughout practical test selection and test execution for a file sharing software system as a representative, the feasibility and effectiveness of the preceding test selection method are well elaborated.

Our CPN model based IOCO test selection method has several advantages. First, conformance test cases are generated through synchronized simulation process with actual test input data and test oracles, so they are well feasible for guiding practical testing executions. Second, as synchronized model simulations with two CPN models are irrespective of their model scale, their effectiveness and practicability are enhanced with better scalability. Therefore, our CPN model based IOCO test selection method is promising and competent for validating the correctness of reactive network software systems more efficiently and more effectively.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China (61262017 and 61262082), the Key Project of Chinese Ministry of Education (212025), the Inner Mongolia Science Foundation for Distinguished Young Scholars (2012JQ03), and the Introduction Foundation for High-Level Talents of Inner Mongolia University. The authors wish to thank the anonymous reviewers for their helpful comments in reviewing this paper.

References

[1] International Organization for Standardization, *Information Technology—Open Systems Interconnection—Conformance Testing Methodology and Framework—Part 1: General Concepts*, ISO/IEC 9646-1, International Organization for Standardization, Geneva, Switzerland, 2nd edition, 1994.

[2] R. M. Hierons, K. Bogdanov, J. P. Bowen et al., "Using formal specifications to support testing," *ACM Computing Surveys*, vol. 41, no. 2, article 9, 2009.

[3] S. R. Dalal, A. Jain, N. Karunanithi et al., "Model-based testing in practice," in *Proceedings of the International Conference on Software Engineering (ICSE '99)*, pp. 285–294, Los Angeles, Calif, USA, May 1999.

[4] J. Yan, J. Wang, and H. W. Chen, "Survary of model-based software testing," *Computer Science*, vol. 31, no. 2, pp. 184–187, 2004 (Chinese).

[5] M. Broy, B. Jonsson, J. P. Katoen, M. Leucker, and A. Pretschner, *Model-Based Testing of Reactive Systems*, vol. 3472 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, 2005.

[6] J. Tretmans, "Model based testing with labelled transition systems," in *Formal Methods and Testing*, vol. 4949 of *Lecture Notes in Computer Science*, pp. 1–38, Springer, Heidelberg, Germany, 2008.

[7] J. Tretmans, "Test generation with inputs, outputs and repetitive quiescence," *Software-Concepts and Tools*, vol. 17, no. 3, pp. 103–120, 1996.

[8] J. Tretmans and E. Brinksma, "TorX: automated model based testing," in *Proceedings of the 1st European Conference on Model-Driven Software Engineering (ECMDSE '03)*, pp. 1–13, Nuremberg, Germany, December 2003.

[9] C. Jard and T. Jéron, "TGV: theory, principles and algorithms. A tool for the automatic synthesis of conformance test cases for non-deterministic reactive systems," *International Journal on Software Tools for Technology Transfer*, vol. 7, no. 4, pp. 297–315, 2005.

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

[11] J. Liu, X.-M. Ye, and J. Li, "Colored Petri nets model based conformance test generation," in *Proceedings of the 16th IEEE Symposium on Computers and Communications (ISCC '11)*, pp. 967–970, Corfu, Greece, July 2011.

[12] R. G. Vries and J. Tretmans, "Towards formal test purposes," in *Proceedings of the 1st International Workshop on Formal Approaches to Testing of Software (FATES '01)*, pp. 61–76, Aarhus, Denmark, August 2001.

[13] M. Weiglhofer, G. Fraser, and F. Wotawa, "Using coverage to automate and improve test purpose based testing," *Information and Software Technology*, vol. 51, no. 11, pp. 1601–1617, 2009.

[14] T. Jéron, "Symbolic model-based test selection," *Electronic Notes in Theoretical Computer Science*, vol. 240, no. 7, pp. 167–184, 2009.

[15] H. Watanabe and T. Kudoh, "Test suite generation methods for concurrent systems based on coloured Petri nets," in *Proceedings of the 2nd Asia-Pacific Software Engineering Conference (APSEC '95)*, pp. 242–251, Brisbane, Australia, 1995.

[16] U. Farooq, C. P. Lam, and H. Li, "Towards automated test sequence generation," in *Proceedings of the 19th Australian Software Engineering Conference (ASWEC '08)*, pp. 441–450, Perth, Australia, March 2008.

[17] H. Zhu and X.-D. He, "A methodology of testing high-level Petri nets," *Information and Software Technology*, vol. 44, no. 8, pp. 473–489, 2002.

[18] Y. Ledru, L. du Bousquet, P. Bontron, O. Maury, C. Oriat, and M. L. Potet, "Test purposes: adapting the notion of specification to testing," in *Proceedings of the 16th Annual International Conforence on Automated Software Engineering (ASE '01)*, pp. 127–134, Antwerp, Belgium, November 2001.

[19] K. P. Chan, T. Y. Chen, and D. Towey, "Good random testing," in *Proceedings of the 9th International Conference on Reliable Software Technology (Ada-Europe '04)*, pp. 200–212, Palma de Mallorea, Spain, 2004.

[20] C. Pachecol, S. K. Lahiri, M. D. Ernst, and T. Ball, "Feedback-directed random test generation," in *Proceedings of the 29th International Conforence on Software Engineering (ICSE '07)*, pp. 75–84, Minneapolis, Minn, USA, May 2007.

[21] J. Fernandez, L. Mounier, and C. Pachon, "Property oriented test case generation," in *Proceedings of the 3rd International Workshop on Formal Approaches to Testing of Software (FATES '03)*, pp. 147–163, Montreal, Canada, October 2003.

[22] G. Fraser, F. Wotawa, and P. E. Ammann, "Testing with model checkers: a survey," *Software Testing Verification and Reliability*, vol. 19, no. 3, pp. 215–261, 2009.