A Test Case Generator for the Validation of High-Level Petri Nets

Jörg Desel
Institut AIFB
Universität Karlsruhe
D 76128 Karlsruhe
Germany
E-mail: desel@aifb.uni-karlsruhe.de

Andreas Oberweis, Torsten Zimmer, Gabriele Zimmermann
Lehrstuhl für Wirtschaftsinformatik II
J.W. Goethe-Universität Frankfurt/Main
D 60054 Frankfurt am Main
Germany
E-mail: {oberweis|zimmer|zimmermann@wiwi.uni-frankfurt.de}

***Abstract* – Test concepts mostly refer to program code and not to models used in earlier stages of the software development process. High-level Petri nets are a widely accepted graphical language for the representation and simulation of requirements, analysis and design models. In this paper, a technique is proposed which generates test cases for the validation of high-level Petri nets in a systematic way. Our approach is derived from *cause-effect graphing*, a concept that was originally developed for testing of program code. However, in our approach the relationship between pre-specified causes and effects is not represented by a Boolean graph, but instead by a Petri net. The test cases are created in a quite efficient way by generating so-called process nets.**

I. INTRODUCTION

The usual definition of software testing typically refers to the testing of program code and not to the testing of models used in earlier development stages of the software development process such as requirements engineering, analysis or design. Model testing could identify many faults earlier and hence decrease maintenance costs [4].

High-level Petri nets are a widely used modeling and specification language for information system behavior. In this paper, a technique for testing high-level Petri net models, called *cause-effect-net-concept*, is proposed. We use the concept for testing Predicate/Transition nets (Pr/T-nets) [3], but it may be applied to other kinds of high-level Petri nets as well. It was derived from a program testing concept, called *cause-effect graphing* [5], [6]. Cause-effect graphing is a Black-Box-Test, where a program is tested against its external specification (function test). First, a natural-language software specification is transformed into a formal-language specification. A logical relationship between inputs (causes) and outputs (effects) is represented by a Boolean graph. By tracing back from each effect, all combinations of causes are found. The graph is then converted into a test case table. Each column in this table corresponds to a test case.

In our concept, the relationship between causes and effects is not represented by a Boolean graph, but by a Condition/Event net (C/E-net), which is a low-level Petri net. In this C/E-net, which we call *cause-effect-net*, causes are represented by places without predecessors and effects by places without successors.

A test case table is automatically created by first converting the direction of all arcs and then simulating the resulting C/E-net. The used simulation concept generates so-called process nets [1], [7]. Each process net describes a relationship between an effect and its causes. All relationships are represented by columns of the test case table which can be interpreted as test cases.

The cause-effect-net-concept is also a Black-Box-Test. The generated test cases check whether all functions described in the natural-language specification are completely and correctly modeled in the Pr/T-net.

Two questions are to be answered during simulation:
1. Is a specified function missing in the Pr/T-net?
2. Is a function modeled incorrectly?

Failure of a test case indicates modeling errors. In this case the internal structure of the Pr/T-net has to be investigated. Possible errors are:

- missing predicates or transitions,
- incorrect arc inscriptions (incorrect arity of the variable tuples or incorrect variable names),
- incorrect transition inscriptions or
- incorrect arity of the predicates.

Additionally, cause-effect graphing and the cause-effect-net-concept have a beneficial side effect. Incompleteness and ambiguities in the natural-language specification can be identified during the investigation of the relationship between causes and effects.

As pointed out in [6], the most difficult aspect of cause-effect graphing is the conversion of the graph into the test case table. We will present an efficient way to compute the relationship between causes and effects by generating process nets [2]. Deriving the test case table from the process nets is simple: each process net is represented by a column of the test case table.

II. BASIC TERMINOLOGY

A. Test cases

Usually, a program cannot be completely tested due to the large number of possible input values. To find errors under time and cost restrictions, a subset of all possible test cases has to be generated which has the highest probability of detecting errors [6]. According to [4] *test cases are created as part of the testing process to direct the search [for errors]. A test case presents a context in which the software is to operate and a description of the behavior expected from the software within that context. ... If the product under test is a model rather than actual code, the test case may contain a textual description of the context as well as some specific input values.*

A simple way to generate test cases is random-input testing, i.e. an arbitrary subset of test cases is selected, but the probability to get an effective set of test cases is rather low. Therefore, random-input testing leads to an inefficient test. Instead, structured methods should be used. There exist many methods for the generation of test cases, an overview is given in [6]. An example of a method for test case design in a systematic way is cause-effect graphing.

B. Process nets

A process net is a net with a special restricted structure, [1], [7]. It describes one single behavior of a system or of a Petri net. A process net is itself a special kind of Petri net, a so-called *occurrence net*, inscribed with labels for the places and the transitions. The labels describe the relationship between the tokens and transition occurrences in the system run and the places and transitions of the occurrence net. In this paper, we use process nets for the derivation of test cases and for the test object inspection.

C. Cause-effect graphing

In [5], [6] a software testing technique called cause-effect graphing is proposed which is performed in a systematic way and selects a high-yield set of test cases. A cause-effect graph is a formal description which is derived from a natural-language specification. First the causes and effects in the specification are identified. Causes are distinct input conditions or equivalence classes of input conditions. Effects are output conditions or system transformations. Then, causes and effects are joined in a structure of logical relationships which is represented by a Boolean graph, the so-called cause-effect graph. By tracing back from each effect in the graph, all combinations of causes which lead to this effect are identified. All causes-effect relationships are represented in a test case table. Every column in this table corresponds to a test case.

D. Cause-effect-net

In this paper, the concept of cause-effect graphing is modified to generate test cases for the validation of a Petri net. The representation of the relationship between causes and effects is not performed using a Boolean graph, but using a Condition/Event net (C/E-net). In the C/E-net, the causes are represented by conditions without predecessors and the effects are represented by conditions without successors. The Petri net, which has to be validated, is called *test object* in the following. The Petri net, which represents the causes-effect relationships is called *cause-effect-net.*

From the *cause-effect-net*, test cases are derived, which describe the relationship between causes and effects. Each relationship represents one possible system behavior that can be executed by the test object. The idea is to convert the direction of all arcs in the cause-effect-net and to simulate the modified net by generating process nets. Every marked effect is simulated separately. Every process net corresponds to exactly one relationship. To ensure objective testing, modeling the test object and generating the cause-effect-net have to be done by different persons.

E. OR-, AND- and SINGLE-relationship

Alternative causes which can belong to the same effect, are represented by an (exclusive) OR-relationship. Common causes which belong to one effect are represented by an AND-relationship. Negated causes are represented by complement conditions. A cause which leads directly to one effect is represented by a SINGLE-relationship. To express more complicated relationships intermediate nodes are introduced (e.g. effect M is present under the condition: cause A AND (cause B OR cause C) occur). Fig. 1 - 3 show the graphical representation of the OR-, AND- and SINGLE-relationship.

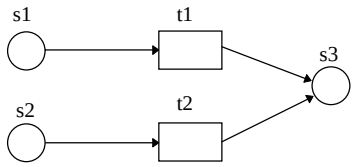

Fig. 1 OR-relationship

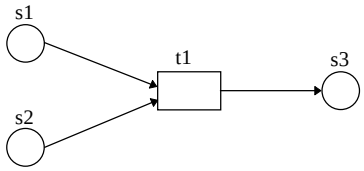

Fig. 2 AND-relationship

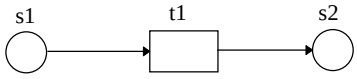

Fig. 3 SINGLE-relationship

F. Non-determinism

A single cause or a combination of causes may lead to different effects (alternative effects). This situation is represented by an ALTERNATE-relationship which is modeled by a forward branched place. This means, that the place has more than one successor transition. Fig. 4 shows the ALTERNATE-relationship.

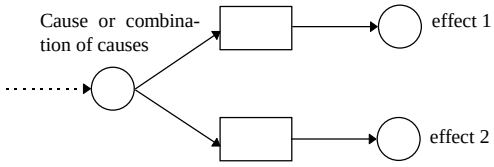

Fig. 4 ALTERNATE-relationship

III. TEST CASE GENERATION AND INSPECTION

A. The generation of the cause-effect-net

The prerequisite for a successful function test is a precise and accurate external specification [5]. The generation of the *cause-effect-net* is similar to the generation of the Boolean graph.

Step 1: Identification of the causes and effects:
First, the causes and effects are identified by reading the natural-language specification word by word and underlining words or phrases that describe causes and effects [6]. They are modeled as conditions of a C/E-net.

Step 2: Linking the causes and effects:
The semantic content of the specification is analyzed and transformed into the *cause-effect-net* by linking the causes and effects using the OR-, AND-, SINGLE- and ALTERNATE-relationship.

B. Identification of the causes-effect relationships

From the cause-effect-net, all relationships between causes and effects are derived by the algorithm represented in Fig. 5. The algorithm is based on the generation of process nets and may be executed automatically (see [2] for the description of a process generating tool).

C. Generation of the test case table

For every relationship between causes and an effect a column is inserted into a so-called *test case table*. Every column in the test case table corresponds to a test case.

Using an ALTERNATE-relationship means, that at least two causes-effect relationships have the same cause or the same combination of causes. Then the causes-effect relationships are combined in one test case.

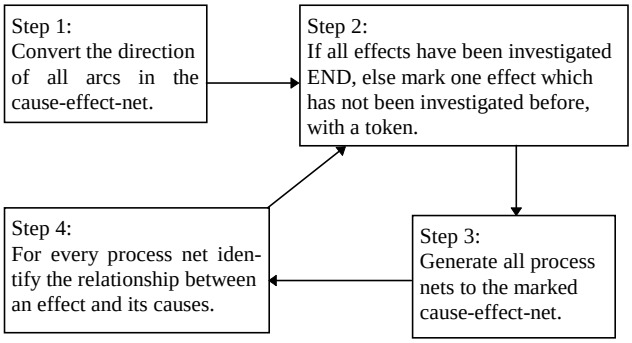

Fig. 5 Algorithm for derivation of the causes-effect relationship

ad Step 2: In every simulation run, only one effect is marked.

ad Step 4: Effects correspond to the sources in the process net (places without any predecessors) and causes correspond to the sinks in the process net (places without any successors).

D. Test object inspection

The test object inspection is executed by checking all test cases. The causes of a column in the test case table correspond to the initial marking of the test object. In a first step, test data is manually generated by transforming the causes of a test case to an initial marking of the Pr/T-net. In a second step, the initialized test object is simulated by generating process nets. In a third step, the test case is analyzed manually: The test case finds no error, if the final marking of the generated process net corresponds to the effect. Using an ALTERNATE-relationship, the simulation of the test object generates several process nets. If the final markings of all created process nets correspond to the different effects of the test case then no error is found.

IV. EXAMPLE

A. External specification and test object

To demonstrate the principle of the concept we use the following specification of a simplified workflow *traveling expenses accounting*: Applicants for a *traveling expenses accounting* fill in a document. In the document they select the method of payment (cash or transfer payment) and the method of notification (internal or external mail). First, the submitted document is checked for correctness. If the submitted document is correct, a payment is ordered by sending a copy of the document to the payment office. The amount of the traveling expenses is refunded by cash or transfer payment. A notification (copy of the document) is sent to the applicant according to the chosen notification mode. The original document is stored in an archive. Incorrect documents are returned to the applicant by internal or external mail and the payment is rejected.

This specification is modeled by the Pr/T-net represented in Fig. 6.

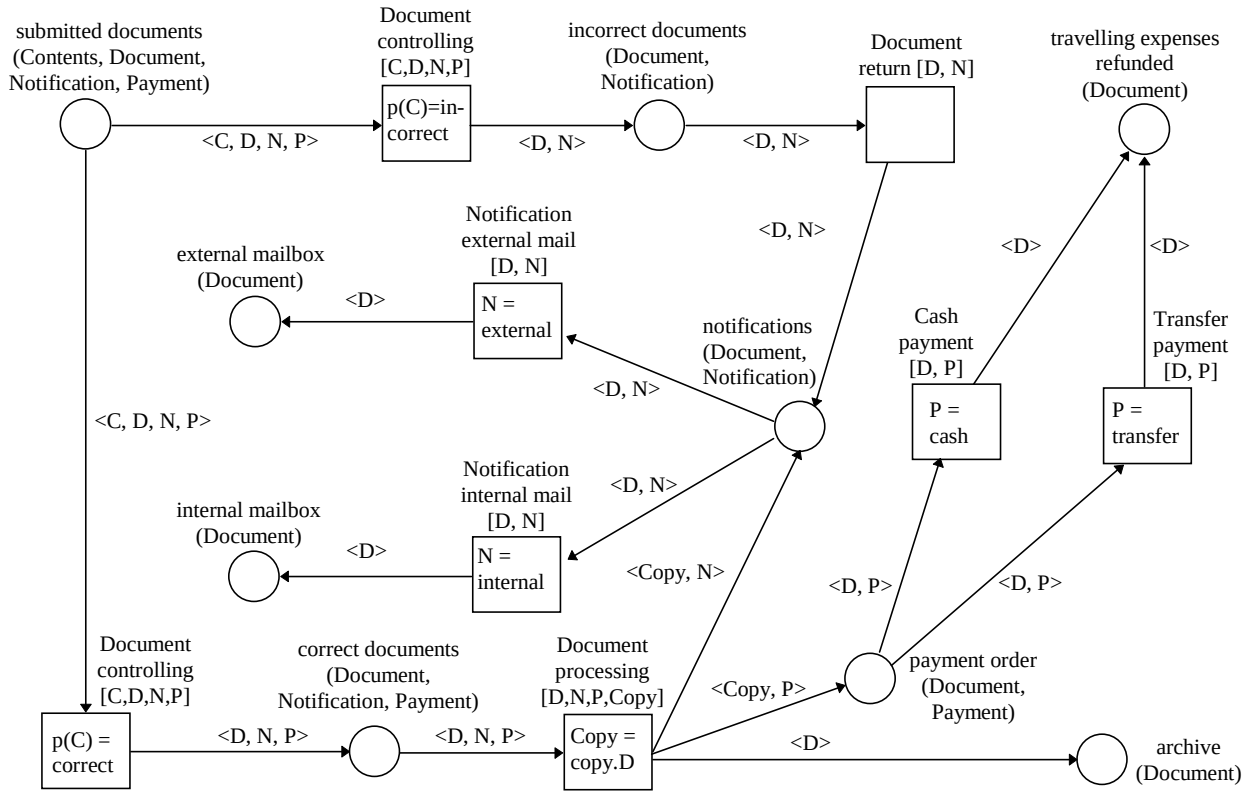

Fig. 6 Traveling expenses accounting modeled as a Predicate/Transition net

The validation of the model is performed by testing. Test cases are created in a systematic way by using the cause-effect-net-concept.

B. The generation of the cause-effect-net

In the first step, the following causes and effects are identified:

Causes: document is submitted, cash payment selected, transfer payment selected, notification by internal mail, notification by external mail, document filled in correctly, document not filled in correctly.

Effects: traveling expenses refunded and document stored, traveling expenses refund rejected and document sent back.

In the second step, the causes and effects are linked by OR- and AND-relationships. The resulting cause-effect-net is represented in Fig. 7.

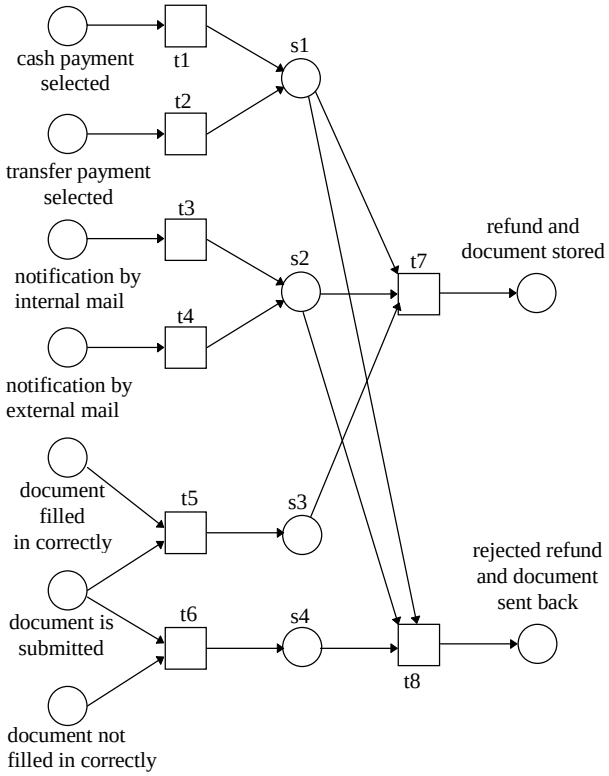

Fig. 7 Cause-effect-net to the Pr/T-net of Fig. 6

C. Identification of all causes-effect relationships

The relationships between causes and effects are derived from the cause-effect-net by generating process nets. Before simulation, the direction of all arcs is converted. The simulation where the effect *refund and document stored* is marked generates four process nets. The simulation where the effect *rejected refund and document sent back* is marked also creates four process nets. Fig. 8 and Fig. 9 show one process net to each effect.

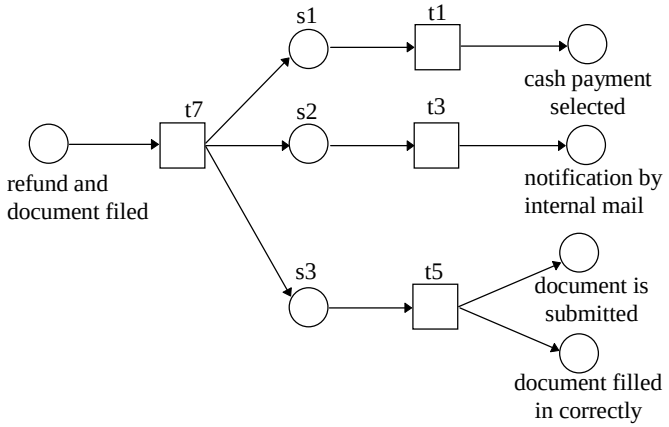

Fig. 8 A process net to the effect *refund and document stored*

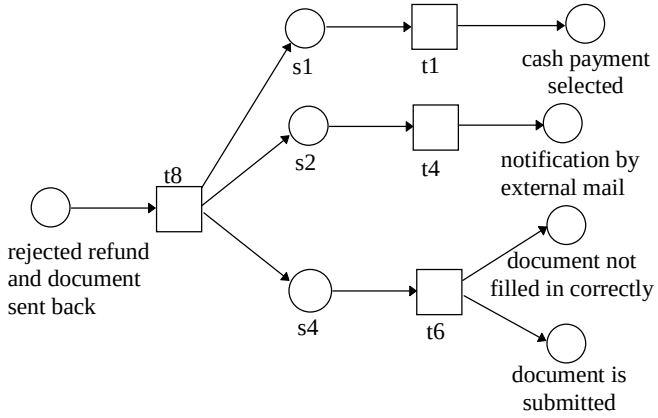

Fig. 9 A process net to the effect *rejected refund and document sent back*

D. Generation of the test case table

The relationships between the causes and the effects are represented in the columns of a test case table. The process net of Fig. 8 is represented in column 1, the process net of Fig. 9 in column 6 of the test case table (see Tab. 1).

Test Cases	1	2	3	4	5	6	7	8
Causes:								
cash payment selected	1	1	0	0	1	1	0	0
transfer payment selected	0	0	1	1	0	0	1	1
notification by internal mail	1	0	1	0	1	0	1	0
notification by external mail	0	1	0	1	0	1	0	1
document is submitted	1	1	1	1	1	1	1	1
document filled in correctly	1	1	1	1	0	0	0	0
document not filled in correctly	0	0	0	0	1	1	1	1
Effects:								
refund and document stored	1	1	1	1	0	0	0	0
rejected refund and document sent back	0	0	0	0	1	1	1	1

Tab. 1 Test case table derived from the process nets

E. Test object inspection

The test object is checked with respect to the generated test cases. The causes of a test case are converted to test data which is a set of tuples. The test data represents the initial marking of the Pr/T-net. The Pr/T-net is also simulated with a process net generator and the test case succeeds, if the final marking of the process net corresponds to the effect of the investigated test case.

Example: For the test case 1 the predicate *submitted documents* in the Pr/T-net of Fig. 6 is marked with the tuple <correct,trav.exp.acc.1,internal,cash>. The simulator generates a process net with the final marking *traveling expenses refunded (copy.trav.exp.acc.1), archive (trav.exp.acc.1), internal mailbox (copy.trav.exp.acc.1)* which corresponds to the effect of test case 1 *refund and document stored*.

V. SUMMARY AND OUTLOOK

In this paper, an approach for generating test cases for the validation of high-level Petri nets is proposed. The concept of cause-effect graphing is transformed into the cause-effect-net-concept. The relationships between causes and effects are not represented by a Boolean graph, but instead by a C/E-net. The computation of the relationship between an effect and its causes can be done in an efficient way. Only the investigated effect is marked in the C/E-net to which a simulator generates all process nets. From the sources and the sinks of the process nets test cases are derived.

The concept has a beneficial side effect: The test tool (simulator) finds one or more additional final markings which are not specified, for example notification by internal and external mail.

The quality of the test case generator itself depends on the correctness and completeness of the cause-effect-net, i.e. the C/E-net is to be validated as well.

The described work is part of a larger project which aims at verification of information systems by analysis of partial ordered Petri net simulation runs [2].

The concept is currently implemented as a prototype system. Future extensions will include a test data generator for Predicate/Transition nets. This generator divides the test data in test input and test output data. The test input data is generated by identification of the relationships between causes and attributes of the predicates of the test object. The test output data is identified using the test input data, the specification and the relationships between effects and attributes of the test object.

Furthermore test concepts are to be generated which check functions which must not be executable, for example a test about multiple payment the traveling expenses (cash and transfer).

VI. ACKNOWLEDGEMENTS

The authors wish to thank Volker Guth for many valuable comments on an earlier version of this paper.

VII. REFERENCES

[1] E. Best, and C. Fernandez, *Nonsequential processes: A Petri net view*, EATCS monographs on Theoretical Computer Science, Springer Verlag, Berlin, 1988.

[2] J. Desel, A. Oberweis and T. Zimmer, Simulation-based Analysis of Distributed Information System Behaviour, in: *Proceedings of the 8th European Simulation Symposium ESS 96*, A. Bruzzone, and E. Kerckhoffs (eds.), Genua, 1996, pp. 319-323.

[3] H. Genrich, Predicate/Transition Nets, in: *Petri Nets: Central Models and Their Properties*, W. Brauer, W. Reisig, and G. Rozenberg (eds.), Springer Verlag Berlin, 1987.

[4] J.D. McGregor, An overview of testing, *Journal of Object-Oriented Programming*; vol. 9, no. 8, Jan. 1997, pp. 5-9.

[5] G.J. Myers, *Software Reliability: Principles and Practices*, John Wiley & Sons, Inc., New York, USA, 1976, pp. 216-227.

[6] G.J. Myers, *The art of software testing*, John Wiley & Sons, Inc., New York, USA, 1979, pp. 56-76.

[7] W. Reisig, *Petri nets: an Introduction*. EATCS monographs on Theoretical Computer Science, Springer Verlag, Berlin, 1985.