VALIDATION OF INFORMATION SYSTEM MODELS: PETRI NETS AND TEST CASE GENERATION

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

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

ABSTRACT

High-level Petri nets are a graphical language for the modeling of distributed information systems. Petri nets can be validated by simulation. In this paper, a technique is proposed which generates test cases for the simulation of high-level Petri nets in a systematic way. The approach is called *cause-effect-net-concept* and is derived from a program code testing concept, the so-called *cause-effect graphing*. As an extension of the concept, a method for test data generation is demonstrated.

1. INTRODUCTION

High-level Petri nets are a widely used modeling and specification language for information system behavior [10]. They combine the advantages of a simple graphical notation with mathematical foundation. Moreover, they allow to model the behavior of an information system and its data structures in one integrated scheme [7]. Petri nets are directly executable by a net interpreter. Thus, Petri net simulation may be used for an early validation of information system models. This can reduce testing effort in later stages of the information system development process.

An important sub-task of testing Petri nets is the identification of an appropriate set of input data (i.e. initial markings) which guarantees that all (functional) system requirements are inspected by the simulation. In this paper, we present a method for automated generation of input data for the simulation of high-level Petri nets, called *cause-effect-net-concept.*

Testing typically refers to program code and not to models used in earlier development stages, e.g. information system models based on Petri nets. Yet the similarities are apparent. The problem of generating input data refers to both simulation of a Petri net model and testing of program code. As far as program code testing is concerned, this is a well-known problem which is usually referred to as *test case generation*. According to [5], a test case consists of a context in which the software is to operate and a description of the behavior expected from the software within this context. A test case for a Petri net consists of input conditions (causes) and a description of the expected output conditions (effects) achieved by the execution of the Petri net under the defined input conditions. Initial markings and expected final markings for simulation can be generated from these sets of conditions.

We adopted a test case generation method called *cause-effect graphing* [6] and transformed it into an approach called *cause-effect-net-concept.* This approach allows the derivation of test cases for Petri nets in an efficient way. It automatically generates a special kind of Petri nets (so-called process nets [1], [9]) as an intermediate representation.

This concept is used for testing Predicate/Transition nets (Pr/T-nets) [4]. Using these generated test cases it is checked whether all specified functions are modeled completely and correctly in the Pr/T-net. In the following, this Pr/T-net is called *test object*.

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 means that a simulation run which starts with the selected initial marking does not lead to the specified final marking. It indicates modeling errors, such as missing places, missing transitions or incorrect arc inscriptions.

2. BASIC TERMINOLOGY

Test cases

A test case for a Petri net consists of

a) a combination of defined input conditions and
b) a description of the expected output condition achieved by the execution of the Petri net under the input conditions mentioned in a).

Test data

By marking the places in a Petri net input conditions can be represented. Test input data of a Petri net are initial markings corresponding to the defined input conditions of a test case. Also, a marking can be interpreted as an output condition: Test output data of a Petri net are final markings (e.g. of a simulation run) which correspond to an (expected) output condition of a test case. Test data is a collection of test input data and test output data.

Process nets

A process net describes one single behavior of a system or of a Petri net. A process net consists of a special kind of Petri net, a so-called *occurrence net* with labeled places and 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 [1], [9]. Minimum elements of a process net are elements without predecessor nodes and they are labeled according to the initial marking. Maximum elements of a process net are elements without successor nodes. In this paper, we use process nets for the derivation of test cases and for the test object inspection.

Cause-effect-net-concept

The *cause-effect-net-concept* was derived from *cause-effect graphing*, a black box program testing concept: The program is tested against its external specification. First, a natural-language specification is transformed into a formal specification. The logical relationships between inputs (causes) and outputs (effects) are represented in a *Boolean graph*. By tracing back from each effect to find all possible combinations of causes, the graph is converted into a so-called *test case table*. Each column in this table corresponds to a test case. As pointed out in [6], the most difficult aspect of cause-effect graphing is obtaining the test case table from the graph. We will present an efficient way to compute the relationship between causes and effects by generating process nets. The test case table is derived from the process nets by interpreting each process net as one test case.

In our concept, the relationships between causes and effects are not represented by a Boolean graph, but instead by a Condition/Event net (C/E-net), which is a low-level Petri net. In this C/E-net places are interpreted as conditions and transitions as events. We call this net *cause-effect-net*. Causes correspond to conditions without predecessors and effects to conditions without successors. Alternative causes which belong to the same effect are represented by an (exclusive) OR-relationship (see Fig. 1). Common causes which belong to one effect are represented by an AND-relationship (see Fig. 2). Negated causes are represented by so-called complement conditions. A cause which directly leads to one effect is represented by a SINGLE-relationship (see Fig. 3). To express more complicated relationships, intermediate nodes are introduced.

In a first step, the cause-effect-net is generated by reading the natural-language specification and by identification of all causes and effects. In a second step, the semantic content of the specification is analyzed and transformed into the causal-effect-net by linking the causes and effects using the OR-, AND- and SINGLE-relationships.

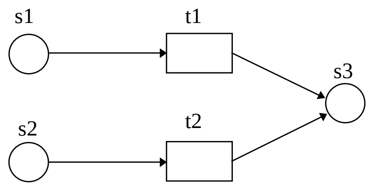

Fig. 1: OR-relationship

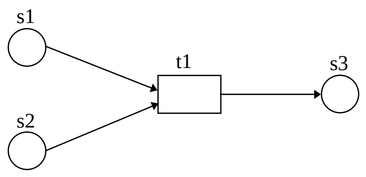

Fig. 2: AND-relationship

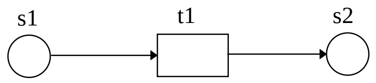

Fig. 3: SINGLE-relationship

Using the cause-effect-net, test cases can be derived by converting the direction of all arcs. The resulting Petri net is simulated. For every simulation run, only one effect is marked. The used simulation concept generates so-called process nets (see [3] for the description of a process generating tool). A process net describes one single behavior of a Petri net. Every process net represents a relationship between an effect and its causes. For each relationship a column is inserted into a test case table. Every column in the test case table corresponds to a test case.

ALTERNATE-relationship

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

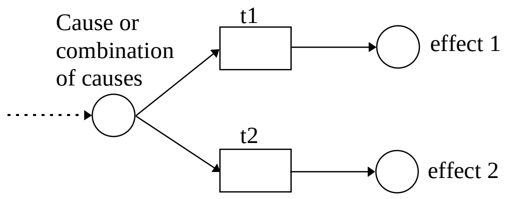

Fig. 4: ALTERNATE-relationship

In an ALTERNATE-relationship, at least two causes-effect relationships have the same cause or the same combination of causes. The causes-effect relationships are combined to one test case.

Test input data
The causes of a test case are transformed into test input data. Correspondingly, for the Petri net an initial marking which corresponds to the causes is generated. First, the attributes of the predicates representing the causes are investigated (the relationships between the causes and the attributes are identified). These relationships are described in a reference table *causes/attributes*.
Attributes of test input data may mutually depend on each other. If both attributes describe the same object then the assignment of an attribute depends on the assignment of the other attribute. In the following, these dependencies are called *referential integrity constraints for attributes*.
Using the reference table *causes/attributes* and respecting the referential integrity constraints for each test case, the test input data is generated. For this task a test data base which stores available example-data can be used.

Test output data
First, to each test case the causes and the effects are considered and all attributes of the Petri net are identified which must be assigned after the occurrence of the effects. These relationships are inserted into a reference table *test case/attributes*.
Using the initial marking (the test input data), the specification and the reference table *test case/attributes* for every test case the test output data is computed (without considering the test object!).

Test object inspection
The test object inspection is executed by checking all test cases. In a first step, the Pr/T-net is marked with the test input data (initial marking). In a second step, the initialized test object is simulated by a process net generator. In a third step, the test case is analyzed: The test case finds no error, if the final marking of the generated process net corresponds to the test output data. Using an ALTERNATE-relationship, the simulation of the test object generates several process nets. If the final markings of the created process nets correspond to the different effects of the test case (test output data) then no error is found.

3. EXAMPLES

Example 1
An example specification for the development of an information system with a Pr/T-net is a simplified library with the following functionality:
Available books can be lent, if a book order arrives. A book which is already lent can be reserved or returned. If a reserved book is returned, the reserving person will receive the book.
In the Pr/T-net of Fig. 5, the actions *lend book, lend reserved book, reserve book* and *return book* of the library are modeled.

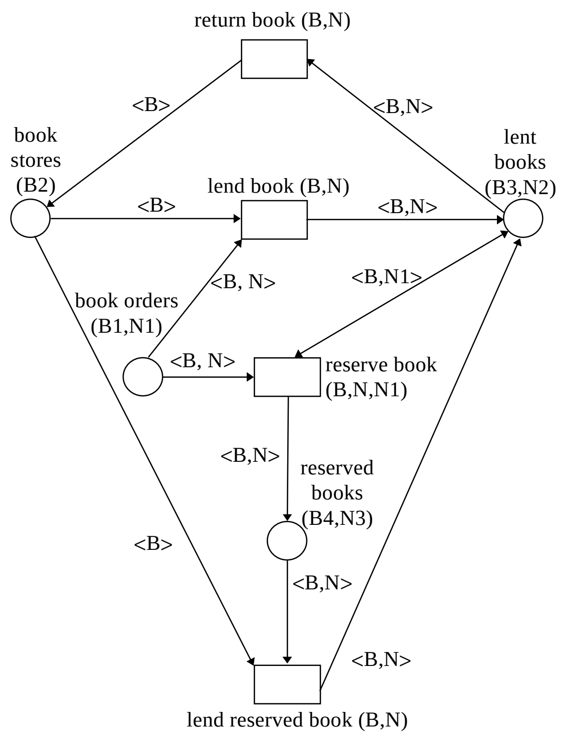

Fig. 5: Library modeled as a Pr/T-net

The generation of the cause-effect-net
In the first step, the following causes and effects are identified:

Causes: book is reserved, book is available, book order is arrived, book is lent.

Effects: book is lent, book is reserved, book is returned.

In the second step, the identified causes and effects are linked using the OR- and AND-relationships. The resulting cause-effect-net is represented in Fig. 6.

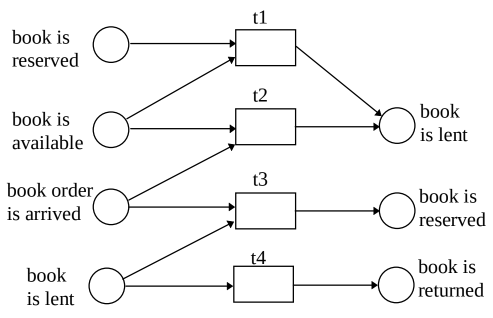

Fig. 6: Cause-effect-net of the library model

The generation of the test cases
In a first step, the direction of the arcs in the cause-effect-net are converted. In a second step to each of the

effects *book is lent, book is reserved* and *book is returned* a simulation run is executed. In every simulation run, only one effect is marked. The following four process nets (Fig. 7 - 10) are generated by simulation:

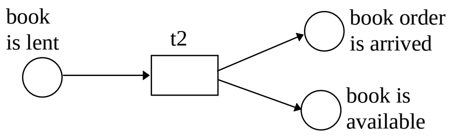

Fig. 7: Process net 1 to the causal-effect-net

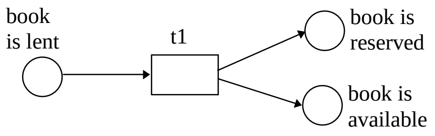

Fig. 8: Process net 2 to the causal-effect-net

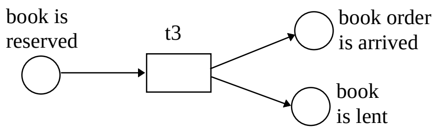

Fig. 9: Process net 3 to the causal-effect-net

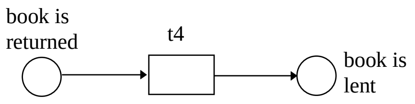

Fig. 10: Process net 4 to the causal-effect-net

From the process nets the test cases are derived: For example, the process net 1 has the maximum elements (causes) *book order is arrived*, *book is available* and the minimum element (effect) *book is lent*. From this process net, test case 1 is derived and inserted into the test case table (see Tab. 1). In this manner, all test cases are identified and recorded.

Test Case	1	2	3	4
Causes:				
book is reserved	0	1	0	0
book is available	1	1	0	0
book order is arrived	1	0	1	0
book is lent	0	0	1	1
Effects:				
book is lent	1	1	0	0
book is reserved	0	0	1	0
book is returned	0	0	0	1

Tab. 1: Test case table to Fig. 5

Test input data generation

To generate the test input data to every test case, the attributes of the predicates of the Pr/T-net representing the causes are identified. They are described in the reference table *causes/attributes* (see Tab. 2).

	Causes	Attributes (Predicates)
1	book is reserved	B4, N3 (reserved books)
2	book is available	B2 (book stores)
3	book order is arrived	B1, N1 (book orders)
4	book is lent	B3, N2 (lent books)

Tab. 2: Reference table *causes/attributes*

Analyzing the specification, the following referential integrity constraints for attributes are identified:

1. If cause 1 and cause 2 are part of a test case, then B2 = B4
2. If cause 2 and cause 3 are part of a test case, then B1 = B2
3. If cause 3 and cause 4 are part of a test case, then B1 = B3 and N1 ≠ N2

Respecting the referential integrity constraints for attributes, the initial markings to the test object are generated by using Tab. 1 and Tab. 2. For this task a test data base (for example B1, B2, B3, B4 ∈ {lord of the flies, faust, hamlet, dictionary}, N1, N2, N3 ∈ {miller, john, smith}) can be used (see Tab. 3).

Test Case	Predicates marked with test input data
1	book stores (B2:= faust) book orders (B1:= faust, N1:= miller)
2	reserved books (B4:= faust, N3:= smith) book stores (B2:= faust)
3	book orders (B1:= faust, N1:= miller) lent books (B3 := faust, N2:= smith)
4	lent books (B3:= lord of the flies, N2:= john)

Tab. 3: Test input data to the library model

Test output data generation

In a first step, to every test case the causes and the effects are considered and all attributes of the predicates which must be assigned after the occurrence of the effect are identified. The relationships are represented in the reference table *test case/attributes* (see Tab. 4).

Test Case	Predicates	Attributes of the Predicates
1	lent books	B3, N2
2	lent books	B3, N2
3	reserved books lent books	B4, N3 B3, N2
4	book stores	B2

Tab. 4: Reference table *test case/attributes*

In a second step, using the test input data, the specification of the information system and the reference table *test case/attributes* (Tab. 4), the test output data is generated. Tab. 5 contains the expected test output data for every test case.

Test Case	Predicates marked with test output data
1	lent books (B3:= faust, N2:= miller)
2	lent books (B3:= faust, N2:= smith)
3	reserved books (B4:= faust, N3:= miller) lent books (B3:= faust, N2:= smith)
4	book stores (B2:= lord of the flies)

Tab. 5: Test output data to the library model

Test object inspection

The inspection of the test object is done by simulation. The Predicate/Transition net is marked with the test input data of every test case and the simulator generates process nets. Since the maximum elements of the process nets (see Fig. 11 - 14) correspond to the test output data, no model errors are found in the Petri net.

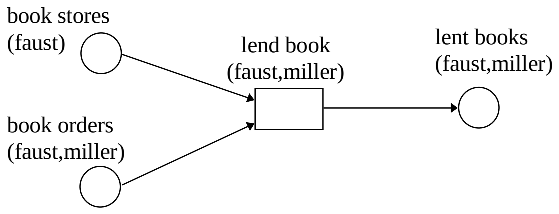

Fig. 11: Process net for the inspection of test case 1

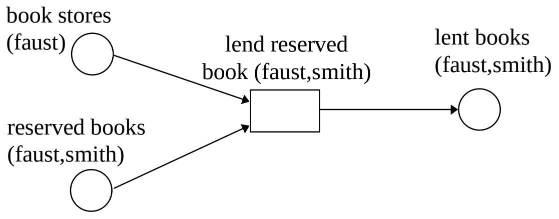

Fig. 12: Process net for the inspection of test case 2

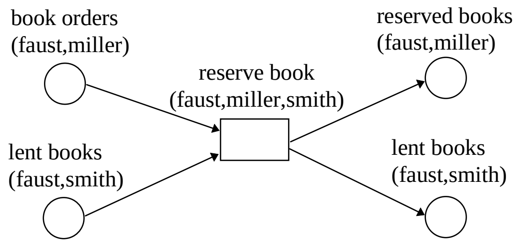

Fig. 13: Process net for the inspection of test case 3

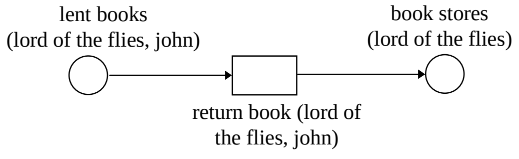

Fig. 14: Process net for the inspection of test case 4

Example 2

The following example demonstrates a modeling error and its detection with the cause-effect-net-concept. Additionally, the test output data generation is demonstrated.

The same specification as in example 1 is given, but the library Petri net model contains an error: An occurrence of the transition *reserve book* removes a tuple from the place *lent books*, but the tuple is not inserted back. To inspect the Pr/T-net, the test cases and test data generated in example 1 (Tab. 1, Tab. 3, Tab. 5) can be used.

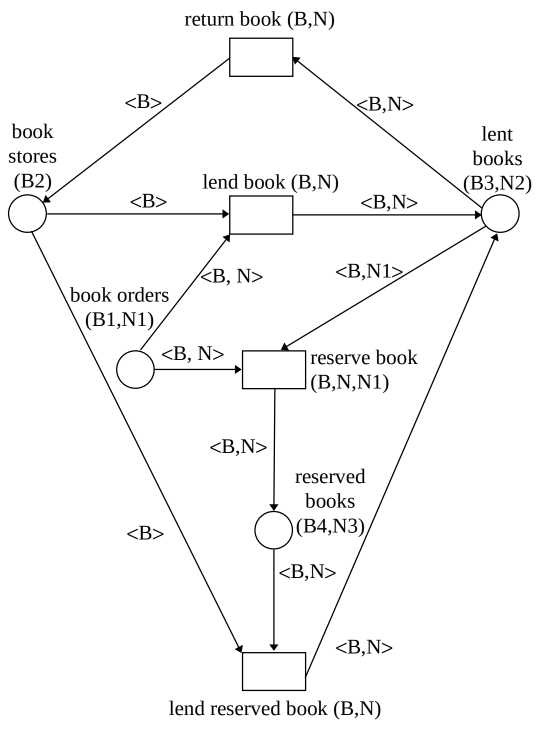

Fig. 15: Library model containing a model error

A simulation with the test input data from test case 3 (book order (faust,miller), lent books (faust,smith)) generates the process net of Fig. 16.

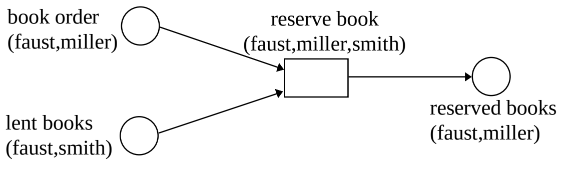

Fig. 16: Process net for the inspection of test case 3

The maximum elements of the process net (reserved books (faust,miller)) are compared with the test output data of test case 3 (reserved books (faust,miller), lent books (faust,smith)). The missing correspondence indicates a model error in the Pr/T-net.

The example demonstrates that causes of an effect also can remain valid after the occurrence of the effect.

Example 3

Example 3 demonstrates a cause-effect-net using an ALTERNATE-relationship. Fig. 17 shows a cause-effect-net which specifies the processing of two resources A and B to two different products (product 1, product 2).

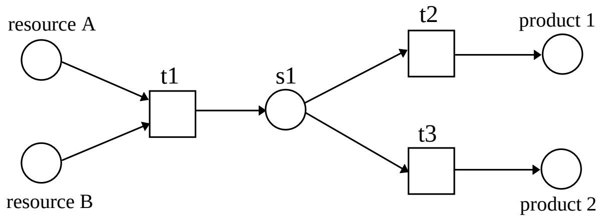

Fig. 17: Cause-effect-net with ALTERNATE-relationship

The process net generation to the marked effect 1 and effect 2 leads to the same combination of causes (the same final markings of process nets). If the causes are transformed into the same test input data, an inspection of the test object leads to two process nets with different final markings. Therefore, both effects and the combination of causes are combined to one test case.
The corresponding test case is represented in the test case table of Tab. 6.

Test Case	1
Causes:	
resource A	1
resource B	1
Effects:	
product 1	1
product 2	1

Tab. 6: Test case table to Fig. 17

6. CONCLUSION

In this paper, we have presented a technique which generates test cases for the validation of high-level Petri nets in a systematic way. This approach is derived from the cause-effect graphing (a method for program code testing). The difference between both methods is the representation of the relationships between causes and effects. In our approach this task is done by using a Condition/Event net. Then, the test cases can be derived in an efficient way. A simulator generates so-called process nets and each process net corresponds to one test case.
Additionally, in this paper a concept is proposed which creates test input and test output data to the test cases. The test input data is generated by identification of the relationships between the causes of the test case and the (attributes of the) predicates of the test object. The test output data is created by using the test input data, the specification of the information system and the relationships between effects and (attributes of the) predicates of the Pr/T-net. Some examples have demonstrated how the test object inspection using the cause-effect-net-concept and the generation of test data is performed.
The work is part of a project called **VIP** (**V**erification of **I**nformation systems by evaluating partially ordered **P**etri net runs) [2].
We use the concept in an information system development environment called INCOME/STAR, which was designed and implemented as a research prototype at the University of Karlsruhe [8].

7. ACKNOWLEDGEMENTS

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

8. REFERENCES

[1] E. Best and C. Fernandez, *Nonsequential processes: A Petri net view*, EATCS monographs on Theoretical Computer Science, Springer-Verlag, 1988.
[2] J. Desel, T. Freytag, A. Oberweis and T. Zimmer, A partial-order-based simulation and validation approach for high-level Petri nets, to appear in: *Proc. of the 15th IMACS World Congress on Scientific Computation, Modeling and Applied Mathematics*, Berlin, Aug. 1997.
[3] J. Desel, A. Oberweis and T. Zimmer, Simulation-based Analysis of Distributed Information System Behaviour, in: *Proc. of the 8th European Simulation Symposium ESS 96*, Genua, A. Bruzzone and E. Kerckhoffs (eds.), 1996, pp. 319-323.
[4] H. Genrich, Predicate/Transition Nets, in: *Petri Nets: Central Models and Their Properties*, W. Brauer, W. Reisig and G. Rozenberg (eds.), Springer-Verlag, 1987, pp. 207-247.
[5] J.D. McGregor, An overview of testing, in: *Journal of Object-Oriented Programming*; Vol. 9, No. 8, 1997, pp. 5-9.
[6] G.J. Myers, *The art of software testing*, New York, NY: John Wiley & Sons, Inc., 1979.
[7] A. Oberweis, An integrated approach for the specification of processes and related complex structured objects in business applications, *Decision Support Systems*, Vol. 17, 1996, pp. 31-53.
[8] A. Oberweis, G. Scherrer, and W. Stucky, INCOME/STAR: Methodology and tools for the development of distributed information systems, *Information Systems*, Vol. 19, No. 8, 1994, pp. 641-658.
[9] W. Reisig, *Petri nets: an introduction*, EATCS monographs on Theoretical Computer Science, Springer-Verlag, 1985.
[10] W. Reisig, *A Primer in Petri Net Design*, Springer-Verlag, 1992.