ISSN 1330-3651 (Print), ISSN1848-6339(Online)

https://doi.org/10.17559/TV-20220822145848

Original scientific paper

A Test Case Generation Method for Workflow Systems Based on I/O_WF_Net

Shanling LI, Changyou ZHENG*, Yaqing SHI, Sainan ZHANG

Abstract: At present, the testing of the workflow system is mainly based on manual testing, and the functions of only some tools are relatively simple. The design of test cases mainly depends on the experience of testers, which makes the lack of test coverage. In this paper, a test case generation method based on the I/O_WF_Net model is proposed. A test case generation algorithm that satisfies the process branch coverage criterion is designed, which solves the problem of automatic test case generation for workflow systems. The algorithm divides the model according to "split-merge pairs" to generate a decomposition tree of the model, and then traverses the tree to generate test cases. A workflow system modelling and test case generation tool are designed and implemented, and an actual workflow system is used as the experimental object to verify the effectiveness of the method.

Keywords: test case generation, workflow systems, Petri net, I/O_WF_Net

1 INTRODUCTION

With the widely used workflow system, the theoretical research has been abundant. But these researches are mainly focused on the system modeling, task scheduling, verification and performance analyzing, and so on, the testing method of workflow is seldom mentioned. As we did more and more testing practice in these systems, the disadvantages were gradually exposed. The former test cases are mainly designed and executed manually by experience. The sufficiency and efficiency of the testing are degraded, and the reliability of the system cannot be guaranteed. So, a challenge is made due to the quick development of workflow technologies. And it is necessary to do some deep and targeting research in this field.

This paper proposed an I/O_WF_Net based workflow test case generation method, which solved the test case generation problem automatically. An experiment is made on a real system, which validated the effectiveness of the given method. The results indicated that, comparing with other testing methods, the given method could discover more defects by less test cases.

2 RELATED WORKS

Because of the high complication of workflow system, the testing technologies seem insufficient. Substantially, the testing technologies for other systems still can be used in workflow systems, but they are non-targeted, and the efficiency is low.

To model the workflow systems, van der Aalst has proposed a famous WF-net [1], and a workflow modeling language YAWL [2]. Aalst himself has published more than two hundred papers and several books in workflow modeling, which can be a technology base in this field. M. C. Zhou [3-5], Q. T. Zeng and H. Q. Wang [6, 7] also have done a lot of work in workflow modeling and performance analyzing.

Literature [8] proposed a directed-graph based workflow testing method. The activities of the workflow are modeled as the nodes in the graph. The testing information such as test cases, expected results, test data, and test scripts, is attached to the node. Test cases are generated by searching possible paths in the model, and with the information attached in the nodes. Due to all the possible paths searched, if there are many split nodes, the structure becomes more complicated, the amount of test cases will become very large, and test efficiency will be lower and lower. In addition, as the directed-graph is adopted, as a semi-formal one, the semantics is not strict enough. And it is hard to analyze other features of the system.

Literature [9] proposed a workflow automatic dynamic testing framework. First, the faults are divided into three classes: private-level faults, which only exist in a specific workflow instance; protect-level faults exist in multiple simultaneously executed workflow instances with the same definition; public-level faults exist in multiple simultaneously executed workflow instances with different definitions. Then, an Agent-based workflow automatically testing framework is given. The paper has just introduced the basics of the framework; the details such as test case generation methods are not given. The scripts of the framework are in a Backus-Naur form, which is not a formal method. The semantics is hard to understand, and it is difficult to implement.

Literature [10] introduced a Web application abstract model based on MVC and workflow. The workflow in that model is not a real work flow, but it is a control flow in the web application. The faults in the MVC based web application is divided into two classes: state-based and code-based. The typical cases for these two kinds of faults are introduced. They suggested that the state-based testing should be implemented in the develop stage, and code-based testing in the running stage. The paper introduces a general abstract strategy, the details of modeling and testing methods are not proposed.

Quan developed a workflow system automatic testing tool [11]. The tool evaluates the system performance by its response time, throughputs and fairness. Before testing, the test data is prepared by reading the testing templates and user configuration. While testing, the workflow engine is driven by test data, and workflow instances are generated by the strategies schedule settings. The system run-time data are recorded to take the performance evaluation. From the view point of workflow system testing, there are two problems in this method: first, the test target of the tool is the workflow engine, but not the workflow application; second, the goal of the testing is the system performance, but not the functional validity.

In our previous work [12], a testing oriented workflow systems modeling method is proposed, which has given an approach based on petri nets. It has not finished the testing work, because it just gave a modeling method, test case generating method has not been given. In this work, we will finish that job.

3 TEST CASE GENERATION METHOD BASED ON I/O_WF_NET

3.1 I/O_WF_Net

The details of modeling methods for I/O_WF_Net have been given in our previous work 0, here we just introduce a basic definition.

Definition 1: $\sum = (P,T,F,M_0)$ is an I/O_WF_NET if the following are satisfied:

(1) (P,T,F,M_0) is a Petri net;

(2) T is the activity set of a workflow;

(3) $P = P_{\text{in}} \cup P_{\text{out}}$, $P_{\text{in}} \cap P_{\text{out}} = \varnothing$, where P_{in} is the input places and P_{out} is the output places;

(4) $\forall t \in T$, $p_{\text{in}} \in P_{\text{in}}$, the execution of t needs input p_{in} if $p_{\text{in}} \subseteq {}^{\bullet}t$

(5) $\forall t \in T$, $p_{\text{out}} \in P_{\text{out}}$, the execution of t will output p_{out} if $p_{\text{out}} \subseteq {}^{\bullet}t$

From definition 1 it can be seen that I/O_WF_Net is essentially a WF-net, and the semantics of the elements is redefined. The activities of the workflow systems are modelled as the transitions of WF-net, and inputs and outputs are modelled as places. To perform software testing, an essential step is to design a suit of appropriate test cases. A key part of the test cases is to describe the inputs and outputs clearly. Because I/O_WF_Net clearly defined the inputs and outputs, so it will be easy to generate test cases. And this method is more targeted to workflow systems; that is the biggest difference between I/O_WF_Net and other models.

3.2 Basic Idea

Before we discuss the algorithm, some definitions and properties of I/O_WF_Net are given, as follows:

Definition 2 (split node): For a node P (or T) in I/O_WF_Net, if the number of post-nodes of P is more than one, then, P (or T) is a split node.

Definition 3 (merge node): For a node P (or T) in I/O_WF_Net, if the number of pre-nodes of P (or T) is more than one, then, P (or T) is a merge node.

Definition 4 (split-merge pair): For split node p_1 and merge node p_2, if all the processes split by p_1 are merged into one process at p_2, then, $p_1$and p_2 are called a split-merge pair. The part between p_1 and p_2 is called a workflow segment, and the paths in the workflow segment are called process segments. If a split-merge pair is started with a transition node, then it is called an *and-split-merge pair*, and if it is started with a place node, then it is called an *or-split-merge pair*.

Definition 5 (split-merge nesting): Considering a model segment shown by Fig. 1, the nodes p_2 and p_4 are split nodes, but there is only one join node responding to them. So, to some specific split/merge node, it may belong to several split-merge pairs. The case shown by Fig. 1 is called a split-merge nesting, the part between p_4 and p_{10} is called the inner split-merge pair, and the part between p_2 and p_{10} is called the outer split-merge pair.

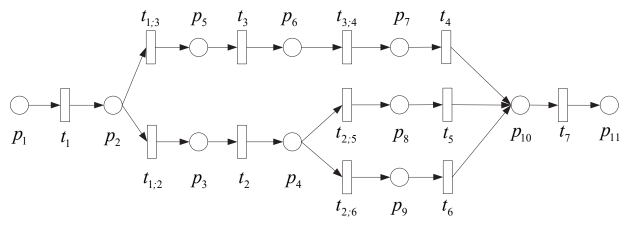


Figure 1 An example of split-merge pairs

For split-merge pairs, the following properties hold:

Property 1: The node type of the split node and the merge node in the split-merge pair must be the same.

Proof: In contradictory method, assuming that the split node and the merge node are of different types, there may be two situations:

First, the split node is a transition, but the merge node is a place. The two flow fragments can work in parallel. The process runs to the merge node since the merge node is a place, that is, the *or-merge* type. The transition follow by can be executed immediately, which may cause the entire workflow executed to the terminate node, but one of the flows may not be executed. In the case, the model does not conform to the basic properties of Petri nets.

Second, the split node is a place, but the merge node is a transition. Then the relationship between the branches is selective execution, that is, there may be only one branch executed to the merge node. Since the merge node is a transition, that is, an *and-merge* type, its conditions may never be satisfied, and the workflow will be blocked at the merge node and will not continue. The model does not conform to the basic properties of Petri nets.

To sum up, the assumption is not true, so the original proposition is true, that is, the types of the split node and the merge node in the split-merge node pair must be the same.

The basic idea of algorithm is introduced below. With the concept of split-merge pairs, the I/O_WF_Net model can be divided into multiple segments according to the split-merge pairs. If there are split-merge pairs in the segments, they are processed recursively according to the same method. So that a tree-like structure (called the I/O_WF_Net Decomposition Tree) will be formed as shown in Fig. 2.

Except for leaf nodes in the tree, each node has a corresponding type:

(1) Sequence indicates that the node contains multiple fragments, one of the fragments is a split-merge pair; these split-merge pairs need to be further processed.

(2) Or/and indicates that the node is a split-merge pair of type or/and, and each of its children can be considered a sub model.

When generating a test case, it is only necessary to traverse the spanning tree. If a node has child nodes, it is processed according to the node type:

(1) The node type is a sequence. It means that there is still a split-merge pair under the node, and the node can be replaced by the link of all its child nodes.

(2) The node type is/or. It means that the type of the split-merge pair is/or, and the node can be replaced by any of its child nodes;

(3) The node type is/and. It means that the type of split-merge pair is/and, the node can be replaced by the parallel form of all its child nodes.

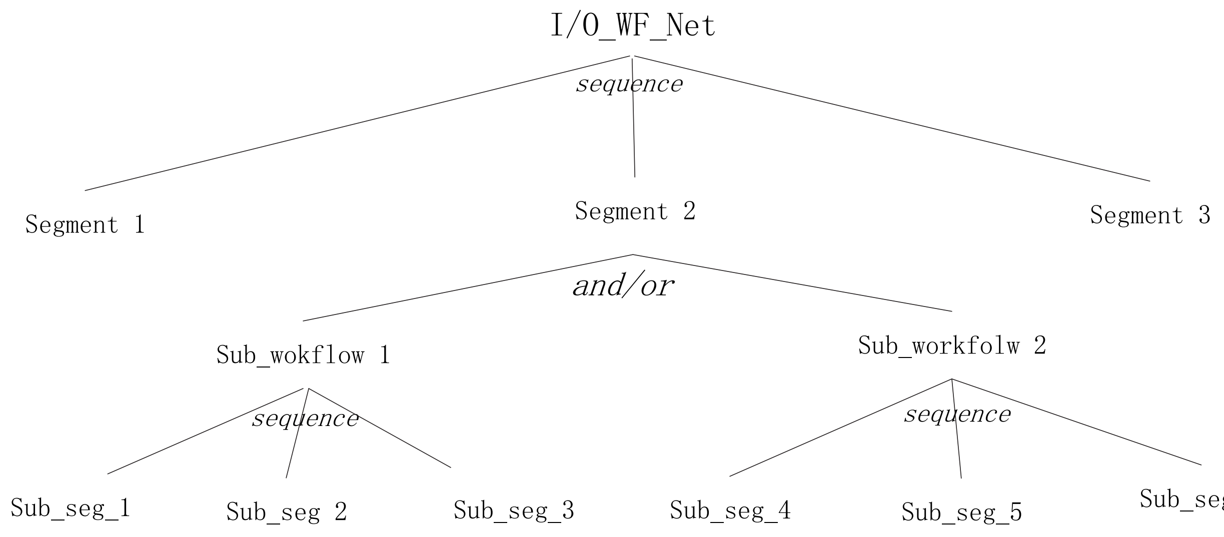


Figure 2 I/O_WF_Net Decomposition Tree

Considering the loops that may occur in the workflow or the rework phenomenon caused by unqualified work at a certain stage, a loop structure will be formed during modeling. When dealing with this situation, it can be regarded as a set of split-merge node pairs, in which the split node and the merge node are the same node. In this split-merge node pair, the loop body is one branch, and the other branch is empty. If there is a clear number of loops n, the loop body of this section is looped n times when generating test case. If there is no clear number of loops but a conditional jump, when generating test case, the loop body of this section is looped once.

3.3 Test Case Generation Algorithm

Based on the above analysis, the test case generation method based on the I/O_WF_Net model is summarized into the following three steps:

Step 1: Construct its decomposition tree according to the I/O_WF_Net model.

Step 2: Traverse the I/O_WF_Net model decomposition tree to generate the corresponding test sequence set TS.

Step 3: Generate test case TCs according to the information contained in each test sequence in the TS.

Among them, the construction process of the I/O_WF_Net Decomposition Tree is as follows:

Step 1: the I/O_WF_Net model is regarded as one node as a whole, and is divided according to the split-merge node pair, and each segment is regarded as a child node.

Step 2: process the nodes whose type is a split-merge pair in all child nodes, and treat each sub-segment as a child node of the split-merge pair node

Step 3: treat each child node of the split-merge pair as a new I/O_WF_Net model, and process the operation recursively according to step 1

The generation algorithm of the I/O_WF_Net decomposition tree can be expressed as Algorithm 1.

Algorithm 1: I/O_WF_Net Decomposition Tree generation

```
TSTreeGenerate( $\sum$ ,node)
Begin
node = $\sum$ ;
Find all split-merge pairs SMs in $\sum$, split the workflow into fragments WFSegment
If WFSegment.count = 1 then
/* There is only one workflow fragment, indicating that there are no split-merge pairs in it*/
    node.type =leaf;
    return;
else
/*The node contains split-merge pairs, which can be represented as a link of multiple fragment pairs*/
    node.type = sequence;
end if
for each $WFSegment_i$
    new node1 = $WFSegment_i$
    node1.father = node;
    if $WFSegment_i$ is not split-merge pair then
        node1.type =leaf;
    else
        /* Node type is marked as the type of split-merge pair */
        node1.type = $WFSegment_i.type$
        /* Search recursively for each process in a split-merge pair*/
        for each $FSegment_j$ in $WFSegment_i$
            new node2 = $FSegment_j$
            /* Treat each process as a new workflow*/
            node2.father = node1
            TSTreeGenerate ( $FSegment_i$ ,node2)/
        End for
    End if
end for
end
```

The process of generating the test sequence TS according to the I/O_WF_Net decomposition tree is as follows:

Step 1: Express the current node (denoted as node) as the link form of all its child nodes (denoted as node.son);

Step 2: Perform the following processing on all child nodes according to their types:

(1) If the child node type is or, and the child node has *n* child nodes (denoted as node.son.son), then copy all the

test sequences containing node.son *n* times, and Replace the node.son part of the sequence with node.son.son;

(2) If the child node type is and, replace all test sequences containing node.son with the parallel form of all node.son.son;

Step 3: Recursive traversal of node.son.son.

The test sequence generation algorithm can be expressed in pseudocode form as in Algorithm 2.

```
Algorithm 2: Test Sequence Generation Algorithm Based on I/O_WF_Net Model Decomposition Tree
TSGenerate(TS, node)
begin
if node.type = leaf return;
TS.segment(node)= $\sum$ node.sons
For each node.son
    If node.son.type='or'
        Copy thesegmentin withnode.son insidefor node.son.soncounttimes;
        For each node.son.son
            /* Replace the node.son part in TS with its child nodes respectively*/
            TS.segment(node.son)= node.son.son;
            TSGenerate(TS, node.son.son);
        End for
    else//type is 'and'
        /* Replace the node.son part in TS with the parallel form of all its children*/
        TS.segment(node.son) = parallel(node.son.sons);
        For each node.son.son
            TSGenerate(TS, node.son.son)
        end for
    end if
end for
end
```

After the test sequence is generated, the final test case can be generated according to the input and output information contained in each test sequence and the generation algorithm can be expressed in the form of pseudocode shown in Algorithm 3.

```
Algorithm 3: test case generation algorithm
INPUT: TS
OUTPUT: Test case collection TC
begin
for each $ts_i \in TS$ // Process each generated test sequence
    for j = start to end in $ts_i$
        /*definition of virtual transition is in article [12] */
        if $t_j$ is virtual transition then
            continue;
        else
            $TC_i.input = TC_i.input + {}^\bullet t_j$
            $TC_i.output = TC_i.output + t_j^\bullet$
        end if
    end for
end for
end
```

Among the above three algorithms, the complexity of Algorithm 1 and Algorithm 2 depends on the number of workflow segments and the number of split-merge pairs in each segment. Assuming that the number of activities in the workflow is n, in the worst case (may not exist at all) In this case), the number of workflow segments is n, and the number of split-merge pairs in each segment is also *n*, and its complexity is $O(n^2)$. The complexity of Algorithm 3 depends on the number of input transition sequences and the number of transitions in each sequence, and has a worst-case complexity of $O(n^2)$.

4 A CASE STUDY AND TOOL IMPLEMENTATION

4.1 Case Study

Fig. 3 is an I/O_WF_NET model established for a workflow system. Fig. 4 shows the recursive segmentation of this model and labels the relationships between the branches. When generating the test sequence, we only need to replace each parent node with its child nodes.

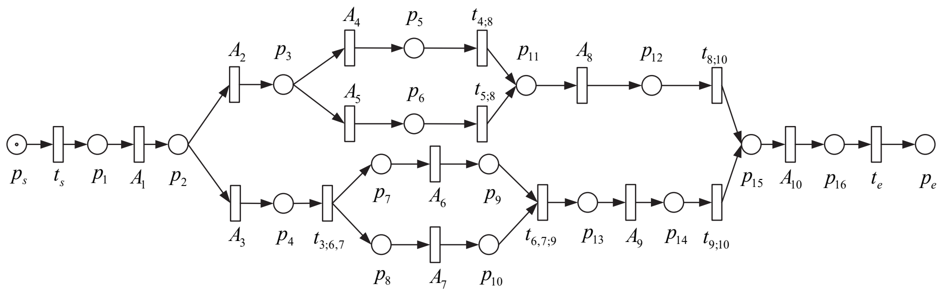
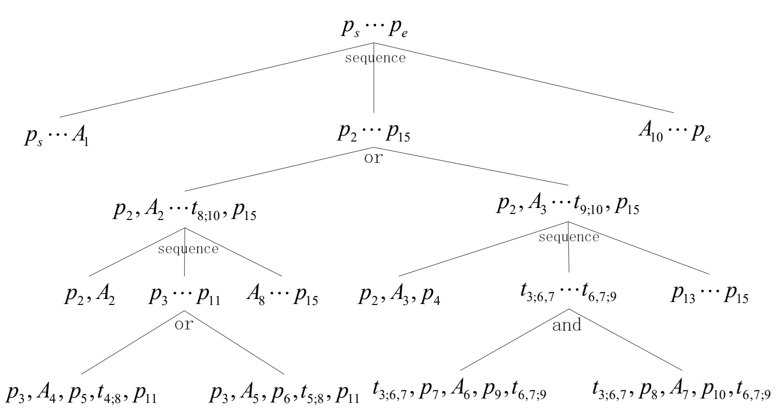


Figure 3 A I/O_WF_NET model for workflow system



$p_s \cdots p_e$
sequence
$p_s \cdots A_1$
$p_2 \cdots p_{15}$
$A_{10} \cdots p_e$
or
$p_2, A_2 \cdots t_{8;10}, p_{15}$
$p_2, A_3 \cdots t_{9;10}, p_{15}$
sequence
sequence
p_2, A_2
$p_3 \cdots p_{11}$
$A_8 \cdots p_{15}$
p_2, A_3, p_4
$t_{3;6,7} \cdots t_{6,7;9}$
$p_{13} \cdots p_{15}$
or
and
$p_3, A_4, p_5, t_{4;8}, p_{11}$
$p_3, A_5, p_6, t_{5;8}, p_{11}$
$t_{3;6,7}, p_7, A_6, p_9, t_{6,7;9}$
$t_{3;6,7}, p_8, A_7, p_{10}, t_{6,7;9}$

Figure 4 I/O_WF_Net Decomposition Tree for the model in Fig. 3

According to the test flow sequence generation method proposed by Algorithm 2, the specific construction process of the corresponding test sequences is shown in Tab. 1.

Table 1 The construction process of TS

Operation	TS
Initialization	$ps\ldots pe$
$TS.segment\left(node\right)=\sum node.sons$	$ps\ldots A_1+p_2\ldots p_{15}+A_{10}\ldots pe$
Nodes $p_2\ldots p_{15}$ are *or* nodes, copy the TS containing the segment 2 times, replace with child nodes	$ps\ldots A_1+p_2,A_2\ldots t_{8;10},p_{15}+A_{10}\ldots pe$; $ps\ldots A_1+p_2,A_3\ldots t_{9;10},p_{15}+A_{10}\ldots pe$;
For $p_2, A_2\ldots t_{8;10}, p_{15}$, recursively traverse $TS.segment\left(node\right)=\sum node.sons$	$ps\ldots A_1+p_2,A_2+p_3\ldots p_{11}+A_8\ldots p_{15}+A_{10}\ldots pe$; $ps\ldots A_1+p_2,A_3\ldots t_{9;10},p_{15}+A_{10}\ldots pe$;
Nodes $p_3\ldots p_{11}$ are *or* nodes, copy the TS containing the segment 2 times, and replace with child nodes	$ps\ldots A_1+p_2,A_2+p_3,A_4,p_5,t_{4,8},p_{11}+A_8\ldots p_{15}+A_{10}\ldots pe$; $ps\ldots A_1+p_2,A_2+p_3,A_5,p_6,t_{5,8},p_{11}+A_8\ldots p_{15}+A_{10}\ldots pe$; $ps\ldots A_1+p_2,A_3\ldots t_{9;10},p_{15}+A_{10}\ldots pe$;
For $p_2, A_3\ldots t_{9;10}, p_{15}$, recursively traverse $TS.segment\left(node\right)=\sum node.sons$	$ps\ldots A_1+p_2,A_2+p_3,A_4,p_5,t_{4,8},p_{11}+A_8\ldots p_{15}+A_{10}\ldots pe$; $ps\ldots A_1+p_2,A_2+p_3,A_5,p_6,t_{5,8},p_{11}+A_8\ldots p_{15}+A_{10}\ldots pe$; $ps\ldots A_1+p_2,A_3,p_4+t_{3;6,7}\ldots t_{6,7;9}+p_{13}\ldots p_{15}+A_{10}\ldots pe$;
Nodes $t_{3;6,7}\ldots t_{6,7;9}$ are *and* nodes, replaced by parallel form of all their children	$ps\ldots A_1+p_2,A_2+p_3,A_4,p_5,t_{4,8},p_{11}+A_8\ldots p_{15}+A_{10}\ldots pe$; $ps\ldots A_1+p_2,A_2+p_3,A_5,p_6,t_{5,8},p_{11}+A_8\ldots p_{15}+A_{10}\ldots pe$; $ps\ldots A_1+p_2,A_3,p_4+\begin{matrix}t_{3;6,7},p_7,A_6,p_9,t_{6;7;9}\\t_{3;6,7},p_8,A_7,p_{10},t_{6;7;9}\end{matrix}+p_{13}\ldots p_{15}+A_{10}\ldots pe$;

According to the generated test flow sequence TS, use the algorithm 3 to generate the corresponding test case example in the following Tab. 2.

Table 2 The generated test cases

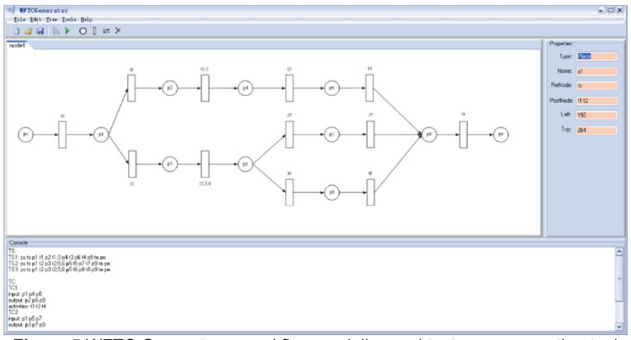
Number	Activity sequence	input sequence	output sequence
1	$A_1, A_2, A_4, A_8, A_{10}$	$p_1, p_2, p_3, p_{11}, p_{15}$	$p_2, p_3, p_5, p_{12}, p_{16}$
2	$A_1, A_2, A_5, A_8, A_{10}$	$p_1, p_2, p_3, p_{11}, p_{15}$	$p_2, p_3, p_6, p_{12}, p_{16}$
3	$A_1, A_2, \begin{matrix}A_6\\A_7\end{matrix}, A_8, A_{10}$	$p_1, p_2, \begin{matrix}p_7\\p_8\end{matrix}, p_{13}, p_{15}$	$p_2, p_4, \begin{matrix}p_9\\p_{10}\end{matrix}, p_{14}, p_{16}$

The example used in the experiment is small in scale, but it already contains the basic structure of the *sequence*, *and-merge*, *or-merge*, *and-split*, *or-split* and other models, so it has a certain representative significance.

4.2 Tool Implementation and Case Study

The main purpose of realizing the tool is to allow users to automatically generate test cases according to the method proposed in this article with the help of the tool, and at the same time, under the guidance of previous testing experience, further select appropriate test cases to improve test efficiency for workflow systems.

We developed a workflow modeling and test case generation tool, WFTCGenerator, as shown in Fig. 5. The tool integrates the I/O_WF_Net model modeling and simplification methods in paper [12], as well as the test case generation method proposed in this paper. It can read the workflow definition file conforming to the XPDL format, analyze each active node to generate the corresponding I/O_WF_Net model, or directly model in the tool, and then automatically analyze the model to generate the corresponding test process sequence TS and test cases TC.



```
TS:
TS1: ps ts p1 t1 p2 t1;3 p4 t3 p6 t4 p9 te pe
TS2: ps ts p1 t2 p3 t2;5,6 p5 t5 p7 t7 p9 te pe
TS3: ps ts p1 t2 p3 t2;5,6 p5 t6 p8 t8 p9 te pe

TC:
TC1:
input: p1 p4 p6
output: p2 p6 p9
activities: t1 t3 t4
TC2:
input: p1 p5 p7
output: p3 p7 p9
```

Figure 5 WFTC Generator: a workflow modeling and test case generation tool

To verify the defect detection capability of the test case generation method designed in this paper, a software testing technique of "mutation testing" proposed by hamlet [16] is adopted. The main idea of this technique is to artificially inject simulated defects into the software, and verify the test case's defect detection ability in a certain field by checking whether these injected defects are found in the test results. The method of defect injection is generally to simulate the actual error-prone situation in the source program, also known as mutation operator. When the mutation operator is applied to the source program, a series of errors similar to mutation like can be obtained from the source program.

In a general program, almost all elements can be used as mutation objects. Commonly used mutation operators have the following categories:

- Numeric value mutation, such as changing "100" to "10"
- Identifier mutation, such as changing the data type of a variable.
- Operator mutation, such as "and" changed to "or";
- Function mutation, which calls different functions that perform similar functions.
- Flow mutation, changing the flow of the program, such as swapping "if... The else..." The following statement;
- Statement mutation, such as changing a statement to do the exact opposite.

In this experiment, according to the main characteristics of workflow system, the following four mutation operators for workflow system are added on the basis of the above commonly used mutation operators. A total of 10 mutation operators are used in the experiment:

- Change the and-split node to or-split.
- Change the or-split node to and-split.
- Change the and-merge node to or-merge.
- Change the or-merge node to and-merge.

In this paper, 60 defects are embedded in the tested system using the above mutation method, and the following three methods are selected for comparison:

- Method 1: Complete random testing;
- Method 2: The test method based on directed graph proposed in literature [8];
- Method 3: Adopt the workflow test scheme proposed in this paper.

The experimental results are as in Tab. 3.

The experimental results are analyzed as follows:

Method of random testing (Method 1): In the design of actual test cases, testers design test cases for each workflow completely according to their own experience, so the number of test cases designed through random testing method is the least, and the corresponding test time used is the least. Since the use case design of random testing does not use any theoretical guidance, there are bound to be some omissions, so the number of defects found is also minimal.

Table 3 The experimental results

	Method 1	Method 2	Method 3
Number of test cases	837	1976	1113
Test time / hours	256	604	340
Number of defects found	45	58	59

Directed graph method (Method 2): Due to the combination of paths, there are many repeated tests for the same process in the test, so the number of use cases is large (second only to the theoretical value of the whole process coverage criterion), and the required test time is correspondingly long. Although this method has designed more test cases, it still fails to find all the defects in the system.

Method Based on I/O_WF_Net model (Method 3): The total number of test cases designed by this method is slightly more than that of the random test method, but far less than that of the directed graph method, and the testing time is in the middle of the two. However, the number of errors found by this method is not only more than that of the random test method, but also one more than that of the directed graph method, which shows the effectiveness of the testing strategy proposed in this paper.

5 CONCLUSION

This paper introduces a workflow test case generation method based on the I/O_WF_Net model: the model is divided according to split-merge pairs, a model decomposition tree is generated, and test cases are generated by traversing the tree. The validity of the algorithm is verified through experiments, and a workflow system modeling and test case generation tool is designed and implemented, which provides the basis for the automated testing of the workflow system.

6 REFERENCES

[1] van der Aalst, W. M. P. (2005). *Pi calculus versus Petri nets: Let us eat humble pie rather than further inflate the Pi hype*. BP Trends.

[2] van der Aalst, W. M. P. & ter Hofstede, A. H. M. (2005). YAWL: Yet another workflow language. *Information Systems*, *30*(4). https://doi.org/10.1016/j.is.2004.02.002

[3] Tang, X., Jiang, C., & Zhou, M. C. (2011). Automatic Web service composition based on Horn clauses and Petri nets. *Expert Systems with Applications*, *38*(10). https://doi.org/10.1016/j.eswa.2011.04.102

[4] Xiong, P., Fan, Y., & Zhou, M. C. (2009). Web service configuration under multiple quality-of-service attribute. *IEEE Transactions on Automation Science and Engineering*, *6*(2). https://doi.org/10.1109/TASE.2008.2009103

[5] Tan, W., Fan, Y. S., & Zhou, M. C. (2009). A Petri net-based method for compatibility analysis and composition of Web services in business process execution language. *IEEE Transactions on Automation Science and Engineering*, *6*(1). https://doi.org/10.1109/TASE.2009.2018013

[6] Sun, S., Zeng, Q.-T., & Wang, H.-Q. (2011) Process-mining-based workflow model fragmentation for distributed execution. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, *41*(2). https://doi.org/10.1109/TSMCA.2010.2069092

[7] Wang, H.-Q. & Zeng, Q.-T. (2008). Modeling and analysis for workflow constrained by resources and nondetermined time-an approach based on Petri nets. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, *38*(4). https://doi.org/10.1109/TSMCA.2008.923056

[8] Ding, D.-W. et al. (2010). A approach for workflow testing based on directed graph. *Microcomputer Information*, *46*(4-3).

[9] Hwang, G., Lin, C., Tsao, L. T. et al. (2009). A framework and language support for automatic dynamic testing of workflow management systems. *Third IEEE International Symposium on Theoretical Aspects of Software Engineering.* https://doi.org/10.1109/TASE.2009.22

[10] Karam, M., Keitouz, W., & Hage, R. (2006). An abstract model for testing mvc and workflow-based web applications. *Proceedings of the Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services*. https://doi.org/10.1109/AICT-ICIW.2006.42

[11] Lin, Q., Lin, X.-Z., & Wang, J.-M. (2008). An automatic and scalable testing tool for workflow systems. *The 3rd International Conference on Grid and Pervasive Computing-Workshops.* https://doi.org/10.1109/GPC.WORKSHOPS.2008.26

[12] Zheng, C., Yao, Y., Huang, S., & Ren, Z. (2015). Modeling workflow systems constrained by inputs and outputs - An approach based on petri nets. *Cybernetics and Information Technologies*, *15*(4). https://doi.org/10.1515/cait-2015-0052

[13] Pocatilu, P. (2013). A framework for test data generators analysis. *Economic Computation & Economic Cybernetics Studies & Research*, *47*(3).

[14] Bendovschi, A. C., Ionescu, B. S., & Ionescu, I. M. (2017). Statistical investigation into exploring the use of cloud computing technology by increasing the users' trust. *Economic Computation & Economic Cybernetics Studies & Research*, *51*(1).

[15] Zhang, D., Sui, J., & Gong, Y. (2017). Large scale software test data generation based on collective constraint and weighted combination method. *Tehnicki Vjesnik/Technical Gazette*, *24*(4). https://doi.org/10.17559/TV-20170319045945

[16] Hamlet, R. G. (1997). Testing programs with the aid of a compiler. *IEEE Trans on Software Engineering*, *3*(4). https://doi.org/10.1109/TSE.1977.231145

Contact information:

Shanling LI, PhD
Army Engineering University of PLA,
No. 1 Haifu Road, Qinhuai District, Nanjing, China
E-mail: lishanling_nj@sina.com

Changyou ZHENG, Associate Professor
(Corresponding author)
Army Engineering University of PLA,
No. 1 Haifu Road, Qinhuai District, Nanjing, China
E-mail: zhengchy@aeu.edu.cn

Yaqing SHI, Professor
Army Engineering University of PLA,
No. 1 Haifu Road, Qinhuai District, Nanjing, China
E-mail: shiyaqing@aeu.edu.cn

Sainan ZHANG, Associate Professor
Army Engineering University of PLA,
No. 1 Haifu Road, Qinhuai District, Nanjing, China
E-mail: zhangsainan@aeu.edu.cn