Circuits and Systems, 2016, 7, 1140-1154
Published Online May 2016 in SciRes. http://www.scirp.org/journal/cs
http://dx.doi.org/10.4236/cs.2016.77098

Aware Time in Bug Fix—A Novel Automatic Test Case Selection for Prioritization of Version Control

G. Parkavi[1], **D. Jeya Mala**[2]

[1]Department of MCA, Anna University Regional Campus, Madurai, India
[2]Department of MCA, Thiagarajar College of Engineering, Madurai, India
Email: Hirithick2015@gmail.com

Received 21 March 2016; accepted 27 May 2016; published 30 May 2016

Abstract

The supreme goal of the Automatic Test case selection techniques is to guarantee systematic coverage, to recognize the usual error forms and to lessen the test of redundancy. It is unfeasible to carry out all the test cases consistently. For this reason, the test cases are picked and prioritize it. The major goal of test case prioritization is to prioritize the test case sequence and finds faults as early as possible to improve the efficiency. Regression testing is used to ensure the validity and the enhancement part of the changed software. In this paper, we propose a new path compression technique (PCUA) for both old version and new version of BPEL dataset. In order to analyze the enhancement part of an application and to find an error in an enhancement part of an application, center of the tree has been calculated. Moreover in the comparative analysis, our proposed PCUA-COT technique is compared with the existing XPFG technique in terms of time consuming and error detection in the path of an enhancement part of BPEL dataset. The experimental results have been shown that our proposed work is better than the existing technique in terms of time consuming and error detection.

Keywords

Automatic Test Cases, Prioritization, Regression Testing, BPEL Dataset, Composite Services, PCUA Test Tree, COT Test Tree

1. Introduction

Software version control is a system or tool that captures the changes to source code elements. Version control tool tracks these changes and allows manipulation of versions and baselines. This is beneficial for many reasons, but most fundamental reason allows you to track changes on per file basis. While here the regression testing is used to seek, to uncover new software bugs, or regressions, in existing functional and non-functional areas of a system after changes such as enhancements or changes, have been need to them, rerunning of existing test suites is regression testing. The "ideal" process would be create an extensive test suit and run it after each and every change.

Unfortunately, for many projects this is just impossible because test suites are too large, because changes come in too fast, because people are in the test loop, because the tests must be done by many different hardware platforms and operating system. Regression testing is more useful to researches in case of retesting, they also try to improve the efficiency of regression testing for various techniques. Regression testing is performed to validate the modified software, to ensure that the modified portions of the software behave or intend and the unmodified sections of the software are not adversely affected by the modifications. Though, the software tester can give their work manually using some other methodologies, if it is also taken for testing automatically to execute the test cases in case of time consuming and expensive process. The cost of regression testing is reduced reusing the test suites that are used to test the original version of the software.

However, rerunning all test cases in the test suite may still require excessive time. To detect an enhancement part of an application, we propose the new technique by focusing the center of a tree for the given BPEL dataset (Business process execution language). At first, given data set is analyzed fully and the test cases are carried through four phases, then the services are presented in the dataset and are selected to bind information and to predict the constraints to execute the services, based on the services. We construct Center of Tree Using Automata test tree (COT) approach is developed to detect the enhancement part of an applications in an efficient manner. And another technique is proposed to compare the path using automata test tree (PCUA-Test Tree) in a better way to find an error in the modified portion of an application within a minimum spanning of time.

A general term for an automata, a machine can be a Turing machine, a finite state machine or any other restricted version of a Turing machine. In that we use the nondeterministic finite automata, NFA is defined as M = (q, sigma, delta, q_0, f) is the same as for deterministic finite automata above with the exception that delta can have sets of states. A string is accepted if any sequence of transitions ends in a final state. There can be more than one sequence of transitions ends in a final state. Each transition has more than one state as causing a tree to branch. All branches are in some state and all branches transition on every input. Any branch that reaches paths, the null are nonexistent state terminates. We proposed the COT-Test Tree approach along with nondeterministic finite automata for regression testing using BPEL dataset.

The following paper is structured as discuss about: section II describes the related works about the regression testing process. Section III describes the proposed work of COT-Test tree approach test case selection for regression testing and PCUA Test tree approach to detect an error for time consuming time in an effective manner. Section IV illustrated the experimental results. Section V describes the future work and conclusion of the paper.

2. Recent Research Works: A Brief Review

Melnikov *et al*. [1] have introduced a non-deterministic finite Rabin-Scott's automata. They employed a special structure to explain all the probable edges of non-deterministic finite automaton which can be utilized to crack several troubles in the finite automata theory. Edge-minimization of non-deterministic automata is one such trouble. In that paper they have introduced two forms of algorithm to crack that trouble to maintain the earlier sequence of articles.

Markus Holzer *et al*. [2] have explored reach ability tribulations on various types of labeled graphs forced to formal languages from a family LL. If all language in LL was recognized by a one-way nondeterministic storage automaton that yields an attractive description of the computational complexity of the labeled graph reach ability trouble based on two-way nondeterministic storage automata by auxiliary work tape which was logarithmic space bounded. In addition, a cyclic graph was considered so as to achieve a lesser bound outcome for auxiliary storage automata which were concurrently space and time constrained.

Fachin *et al*. [3] have intended the idea of systolic automata on regular Y-trees in column normal form and confirms that it was equal to one in the column normal form. Subsequently they describe those regular Y-trees entirely there the class of languages recognized by non-deterministic automata similar to that of deterministic automata. An analogous outcome was achieved for stability. Moreover, they reveal that the systolic automata over regular *Y*-trees distinguish only regular languages. Lastly, various closure properties and relations among classes of languages received by systolic automata on various *Y*-trees were analyzed.

Mohamed *et al*. [4] have projected three major chapters of software testing they were test case generation, test execution, and test evaluation. Test case generation was the heart for any testing process but it needs test data for executed that creates the test data generation not inferior than the test case generation which plays a crucial role in lessening the time and effort spent throughout the testing process. Various methods which were used in the earlier period based on the generation of test cases and test data was presented in that paper. A unified Modeling Language UML model is one of best amid those models. Most researchers were paying more importance for optimizing and reducing the test cases before generating them.

Rajvir Singh *et al*. [5] have projected a technique known as test case minimization technique. This technique was employed so as to reduce the charge of test on the basis of execution time, resources etc. The main goal of the test case minimization technique was to produce the representative set that fulfills each and every requirement present in the original test suite including the end of the test with minimum times. The core purpose for the test case minimization technique was to remove test cases those were outmoded and redundant. The main approaches present in the literature part of that paper includes Heuristics, Genetic Algorithm, Integer Linear Programming based techniques.

Parkavi *et al*. [6] have intended a technique which was employed to produce the test cases automatically to study the changes present in different kinds of BPEL (Business Process Execution Language) dataset. A Hierarchical Test Tree (HTT) was generated on the basis of the original and previous type of composite services. The difference in both the trees which were created using BPEL dataset were found out by way of assessing the control flow. The Test Case Prioritization Algorithm (TCPA) using the multiple criteria was employed for assigning priority for the test cases.

Hitesh Tahbildar *et al*. [7] have projected an impression of Automatic Test data generation. The key function of that paper was to obtain the fundamental ideas based on automated test data generation research. Several implementation approaches were explained based upon their advantages and disadvantages. The forthcoming confronts and tribulations of test data generation were described. As a final point they explores which area have been focused mainly for building Automatic Test data generation more successful in industry.

Isabella *et al*. [8] have initiated some techniques meant for test case generation and process for various GUI based software applications. In our time software testing became a tough job thus testing period during the progress of the software lifecycle was crucial, it has been classified into graphical user interface (GUI) based testing, logical testing, integration testing, etc. From the above types of testing GUI Testing was one of the best technique because of its sophisticated nature to deal with the software. However more complications were there in GUI ultimately. The test ought to be mandatory for achieving effectiveness, competence, better fault detection rate and fine path coverage.

D. Gong *et al*. [9] have inaugurated a novel technique so as to automatically get the branch correlations of dissimilar constrained statements and recognize infeasible paths. In the beginning, many theorems were detailed in order to spot the branch correlations on account of probabilities of the conditional distribution equivalent to various branches result (*i.e*. true or false); subsequently, the maximum likelihood estimation was used to attain the values of the probabilities; at last, infeasible paths were identified based on branch correlations. That approach method can identify the infeasible paths precisely thus it yields a successful and automatic approach of identifying infeasible paths, which was essential in enhancing the effectiveness of software testing.

Li *et al*. [10] have projected an investigation in order to recognize the susceptibilities from source code named as backward trace analysis and symbolic execution. The primary task was to found all hotspots in source code file. A data flow tree was built on the basis of every hot spot for attaining the probable execution traces. After that the symbolic execution was carried out so as to put up program constraint (PC) and security constraint (SC). A constriction which was imposed in the program variables using the program logic was termed as program constraint. A constriction on the variables of the program which fulfils to certify the security of the system was termed as security constraint (SC). As a final point, the hot spot was stated as vulnerability whether it includes the provision of values to program inputs which fits for PC but defies SC, *i.e*. it fulfils for $\mathrm{PC} \wedge \overline{\mathrm{SC}}$. From Juliet Test Suites obtained from US National Security Agency (NSA) the approach was experimented on the test cases.

Najumudheen *et al*. [11] have projected an innovative test coverage investigation approach for object-oriented programs. That approach includes three parts they were graph construction, instrumentation, and coverage analysis. The source program was transformed into a dependence graph-based representation, known as Call-Based

Object-Oriented System Dependence Graph (COSDG) in the graph construction phase. In the instrumentation phase includes the instrumentation of the source code at specific points. In the coverage analysis phase, the instrumented source code was performed for various test inputs, and the edges of COSDG are marked via graph marker. Many coverage measures were calculated from the marked COSDG via the coverage analyzer to create a coverage report. Moreover for traditional coverage measures the inheritance and polymorphic coverage were assumed.

Mishra *et al*. [12] have introduced a technique for pipelined processors based on graph coverage functional test program generation. Initially, it creates the graph model of the pipelined processor from the specification via functional abstraction automatically. Next, on the basis of the coverage of the pipeline actions it produces functional test programs. Lastly, the test generation time was diminished significantly because of the utilization of module level property checking. For analyzing the performance they have employed that technique on the DLX processor.

Hewelt *et al*. [13] have projected an effectual technique which employs heuristics on the basis of a software component test dependency graph in order to create a test order mechanically that desires (near) lower test stubs. For this reason, that approach diminishes testing effort and cost. That paper includes the evaluation of three renowned graph-based techniques; the standard Le Traon *et al*.'s technique employed for the actual software application approach as well as the proposed approach generates the finest number of stubs. Experiments concerned with the arbitrarily simulated reliance models with 100 to 10,000 components are carried out which exposes that the performance is better with a drop in the average running time of 96.01%.

Damini *et al*. [14] have introduced the Component Based Software Development (CBSD) so as to generate the software applications in a promptly mode. Primarily, the software result was formed by assembling dissimilar constituents of available software from many peddlers which lessens the charge and interval of the software product. On the other hand the tester faces several troubles in the test period as the tester has a minimal right for the entry of source code of reclaimed factor of the product. The notion was termed as Black-Box Testing (BBT) of software constituents since Black-Box Testing was employed when there is no availability of source code. That paper focused on a function by Finite Automata-based testing. That testing comprises two forms which were NFA-based testing and DFA-based testing. The function of the approach was described by means of UML diagrams.

Nathaniel *et al*. [15] have introduced the fault location measure appropriate for networks which use distributed routing control via routing tags and message transmission protocols. Faults present in the data lines can distort message routing tags transmitted and there may lead to misrouting of messages. When transmission of data has started, protocol lines if flawed, can avoid a message path from being recognized or else it cause the path to "lock up". These faults possess distinct effects on the performance of the network than the flaws which are measured earlier for centralized routing control systems. The single-fault location measure employs logical superset for the centralized control systems and was appropriate for both circuit and packet switching networks.

3. Proposed Method for Path Comparison Using Automata Test Tree

In our proposed work, to find an error in an enhancement part of the application, we used the technique center of the tree. Let *v* be a point in a connected graph *G*. An eccentricity $e(v)$ of *v* is defined by, $e(v)=\max\{d(u,v)|u\in v(G)\}$ the radius $r(G)$ is defined by, $r(G)=\min\{e(v)/v\in v(G)\}$. r is called a central point if, $e(v)=e(r)$ and the set of all central point is called center of *G*. Such that every tree has a center consisting of either one point or more than two adjacent points. Let us discuss about the center of the tree which has been discussed in general. The result is obvious for the tree T_1 to T_n. Now, let *T* be any tree with $p\geq n$ points. *T* has at least two end points and maximum distance from a given point u to any other points *v* occur only when, *v* is an end point. Now delete the entire end point tree *T*. The resulting graph T' is exactly one less than the eccentricity of the same point in *T*. Hence *T* and T' has the same center. If the process of removing the end point is repeated. We obtain successive tree having the same center *T* and we eventually obtain a tree which is either T_1 or T_2. Hence the center of *T* consists of either one point or more than two adjacent points. Any connected (p,q) graph with $(p+1=q)$ is a tree while, the components of a Tree from that we find out the in-degree and out-degree of a tree. If the degree of every point of a tree compared by others by using in-degree and out-degree technique, we found out the newly enhanced part of an application for various dataset in BPEL. Hence, the degree of a point v_i in a graph *G* is the number of lines incident with v_i. the degree of

v_i denoted by $d_G(v_i)$ or $d(v_i)$ is the definition of degree, where we found out the center of node in the tree by using the center of tree from that we compare the in-degree and out-degree of the tree such that we easily find out the enhancement part of the tree for the version of BPEL dataset.

$$B = \sum_{r \varepsilon v} d + (v) \tag{1}$$

$$C = \sum_{v \varepsilon V} d - (v) \tag{2}$$

An arc (u, w) contributes one to the out degree of *u* and one to the in-degree of *w*. Hence each are contributes 1 to the sum B and 1 to sum C. Hence $(B = C = q)$, from that, we achieve the enhance part of the version as newly done. To minimize the time consumption of comparing the node within the tree, we proposed another idea also to detect error in an enhanced part of various version of BPEL dataset along with non-deterministic finite automata.

3.1. COT-TEST TREE

In our proposed work the path comparisons between the old and new version of BPEL dataset was achieved by using COT-TEST TREE. At first the COT-TEST TREE will detect the modifications. Once the modifications are detected, then the test cases that are required to test the process found an error. The COT-TEST TREE algorithm describes the procedure to generate the test cases for detecting changes and an error in the new version.

The basic flow of an old version of the BPEL data set has been described in **Figure 1**. Moreover the calculation of the in-degree and out-degree of the version (1.0) that is an old part of a BPEL dataset which is described in **Figure 1** has been explained in **Table 1(a)** and **Table 1(b)**. The in-degree and out-degree has been calculated in order to found the center of the tree and the calculated value has been tabulated and given in **Table 1**.

In **Table 1(a)**, the calculation of the in-degree and out-degree for the layer 1(ATM-WSDL) and layer 2 (selection in) for the version (1.0) has been given. Similarly for the layer 3 the in-degree and out-degree of the ticket, account and ATM has been tabulated in **Figure 1(b)**. Here we calculate the in-degree and out-degree by focusing a center of tree to detect an enhancement part and to track errors by using COT-Test Tree algorithm. Let us see the algorithm how to detect an error in enchantment part, with an efficient manner. The COT-Test Tree algorithm has been explained below.

3.2. COT-Test Tree Algorithm

Trees are very important for sake of their applications to many fields. Further a tree is the simplest non-trivial type of a graph and in trying to prove a general result. In a tree every edge is a bridge. Every point of degree > 1 is a cutpoint. Any connected (*p*, *q*) graph with *p* + 1 = *q* is a tree. Every connected graph has a spanning tree. The components of a forest are trees. If the degree of every point of a graph is at least two, then G contains a cycle is the basic concept said about the tree [7]. By using the center of Tree, we proposed the COT-Test Tree

Table 1. Version (1.0) calculation of in-degree and out-degree of tree.

(a)

ATM WSDL		SELECTION IN	
In-degree	Out-degree	In-degree	Out-degree
0	1	1	3

(b)

TICKET		ACCOUNT		ATM	
In-degree	Out-degree	In-degree	Out-degree	In-degree	Out-degree
1	0	1	0	1	0
1	3	1	5	1	8
1	0	1	0	1	0

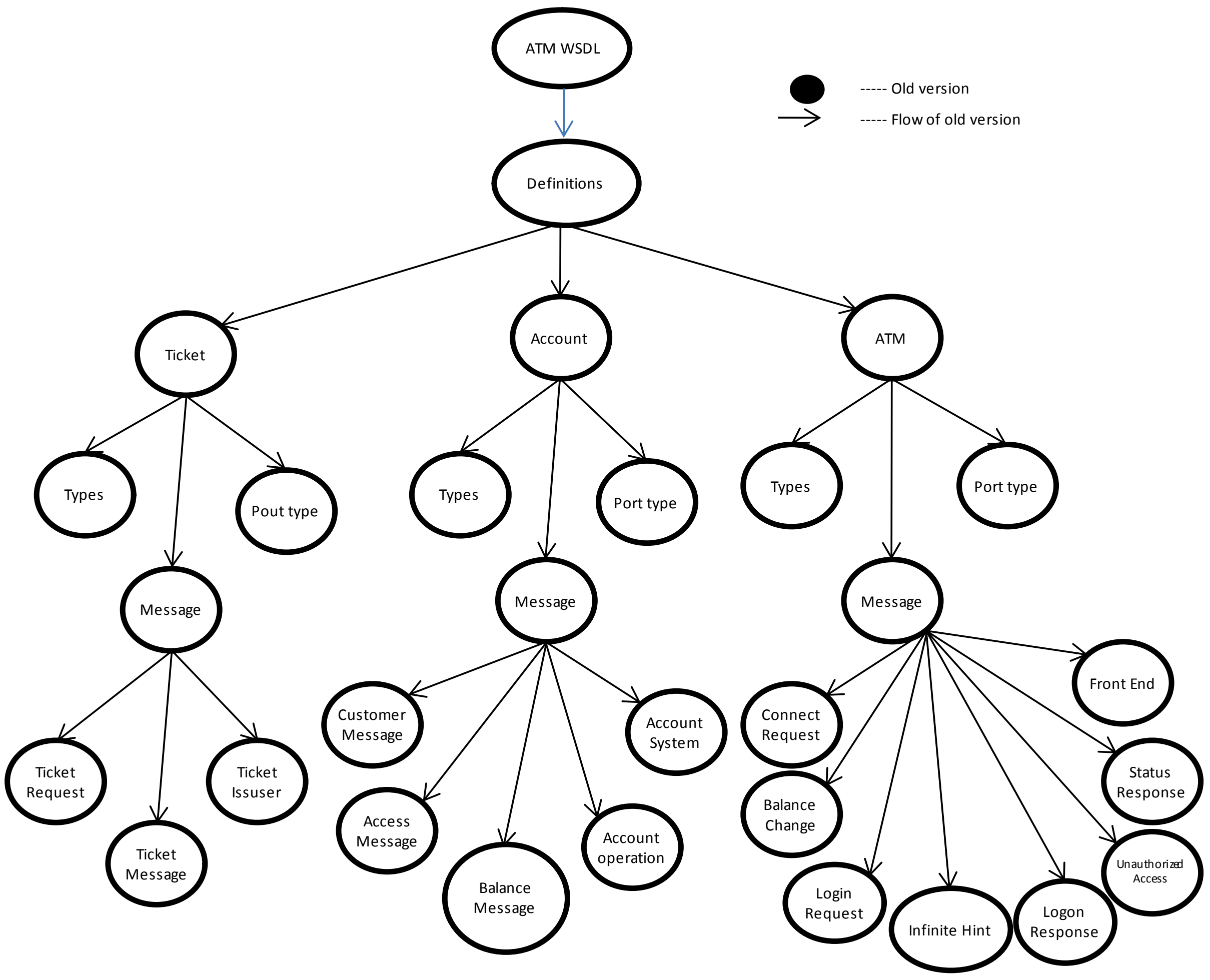

Figure 1. Version 1.0—an old part of a BPEL data set.

Algorithm to detect an error during enchantment happen in any BPEL data set. The algorithm has been given below.

To find the enhancement part of an application, we used the technique center of tree to detect the changes happened in an application during development.

```
if compNum = 0;
For $(\text{int } v = 0; v < |v|; v{+}{+})$
{// if $v$ is not yet visited, it's the start of newly discovered connected component containing
If $(!\text{visited } [v])\{$
 compNum++;
 cout << "component" << compNum << ":";
Int queue $q$;
 $q$.enqueue$(v)$;
 visited$[v]$ = true;
 while$(!q$.empty$(\ ))$;
 {int $w = q$.dequeue$(\ )$;
 cout << $w$ <<;
For each edge to some vertex $k$ {//***
 visited$[k]$ = true;
 $q$.enque$(k)$;}}}
cout << end1 << end1;
if $(\text{compNum} == 1)$
cout << "The graph is connected" <<;
else
cout << "There are" << compNum << "connected Graph";
Graph $G(v, E, r)$ // Graph (vertices,Edges,Eccentricity,radius)
if $(e(v) = \max\{d(u,v) \mid u \in v(G)\})$ // $v$ is defined
if $(r(g) = \min\{e(v) \mid v \in v(G)\})$
$\{e(v) = r(G)$ / /eccentricity = radius− > central point$\}$
}/*$T$ be any tree*/
if $(T1, T2, \cdots, TN = 0)$
{if $(p = N)$ {
Int queue $p$;
p.enqueue $(TN)$;
p.delete ( );
}}
if $(p+1=q)$ //components of a first tree//
//initially find out the in-degree and out-degree
if $(\text{compNum} = 0)$;
{if $(d(G)vi \mid d(v1))$ // To check the graph vertices.}
//let us assume that($u$,$v$) attributes to out-degree of U and one to the in-degree of $w$.
if $(B = c = q)$
//these value are equl to 1
Int $B$=1;
//$c$=1,$q$=1;
$B$++;
Cout<<"Equal(or) not"<<;
While(!$B$.is Empty()) {int $d$=$B$.dequeue():
Cout<<$d$<<;
For each vertices to max|min.value;
Visitd[$d$]=true;
if $(B = \&\& c =)$ {cout<< "The node has been printed:"<<1;
} $B$.enqueue( );}}
coout << end2 << end2;
```

The flow chart of the COT-Test tree algorithm has been given below in **Figure 2**.

In **Figure 3**, version 1.0 denotes an enhanced part of BPEL dataset and to track the modified portion work properly without affecting an old version and a newly enhanced part work correctly with or without occurring error. **Table 2** is the calculation of in-degree and out-degree to detect the newly constructed path through this, we can fix an error by focusing a center of tree by using COT-Test Tree Algorithm for newly enhanced part of BPEL dataset.

In the above table the center of the tree has been detected using automata (COT-TEST TREE) algorithm for BPEL dataset, in-order to express the behavior of the composite services presented in the BPEL dataset. Moreover it also detects the enhanced part of the version in BPEL dataset.

3.3. PCUA-TEST TREE Algorithm

```
1. GET $w$
2. SET $n+1$
3. GET $n$
4. OBTAIN $w = xa$
if $\hat{\delta}_D(\{q_0\}, x) = \hat{\delta}_N(q_0, x)$
```

```
5. Obtain states
6. $\{p_1, p_2, p_3, \cdots, p_k\}$
7. DEFINE $\hat{\delta}$ for NFA
    a. if $\hat{\delta}_N(q_0, w) = \bigcup_{i=1}^{k} \delta_N(p_i, a)$
    b. Else
8. $\delta_D(\{p_1, p_2, p_3, \cdots, p_k\}, a)$
9. OBTAIN $\bigcup_{i=1}^{k} \delta_N(p_i, a)$
10. SET $\hat{\delta}_D(\{q_0\}, x) = \{p_1, p_2, p_3, \cdots, p_k\}$
11. THEN
12. DEFINE $\hat{\delta}$ for DFA
13. OBTAIN $\hat{\delta}_D(\{q_0\}, w) = \delta_D(\hat{\delta}_D(\{q_0\}, x), a)$
14. GET $\delta_D(\{p_1, p_2, p_3, \cdots, p_k\}, a)$
15. $\bigcup_{i=1}^{k} \delta_N(p_i, a)$
16. SHOW $\hat{\delta}_D(\{q_0\}, w) = \hat{\delta}_N(q_0, x)$
17. OLD VERSION SATISFICES ABOVE SAME CONDITION ( OBTAIN AS TRUE)
18. SHOW
19. IF CONDITON OBTAIN
20. $\hat{\delta}_D(\{q_0\}, w) \neq \hat{\delta}_N(q_0, x)$
21. AN ENHANCED PART MAY BE HAPPENED
22. STOP
```

Figure 4 represents flow chart step by step description for PCUA-TEST tree algorithm. Let it clear an idea about the PCUA algorithm.

Figure 5 represents the version 1.0 (a) an old part of BPEL data set, Let w be of length $n + 1$, and assume the statement for length n, break up w up as $w = xa$, where a is the final symbol of w. by the inductive hypothesis,

$$\hat{\delta}_D(\{q_0\}, x) = \hat{\delta}_N(q_0, x) \tag{3}$$

Let both these of N's states be, $\{p_1, p_2, \cdots, p_k\}$. The inductive part of the definition of $\hat{\delta}$ for NFA tells us

Table 2. Version (2.0)—calculation of in-degree and out-degree of tree.

(a)

ATM WSDL		Selection IN	
In-degree	Out-degree	In-degree	Out-degree
0	1	1	4

(b)

Ticket		Account		ATM		Customer	
In-degree	Out-degree	In-degree	Out-degree	In-degree	Out-degree	In-degree	Out-degree
1	0	1	0	1	0	1	0
1	3	1	5	1	9	1	0
1	0	1	0	1	0	1	0
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	0

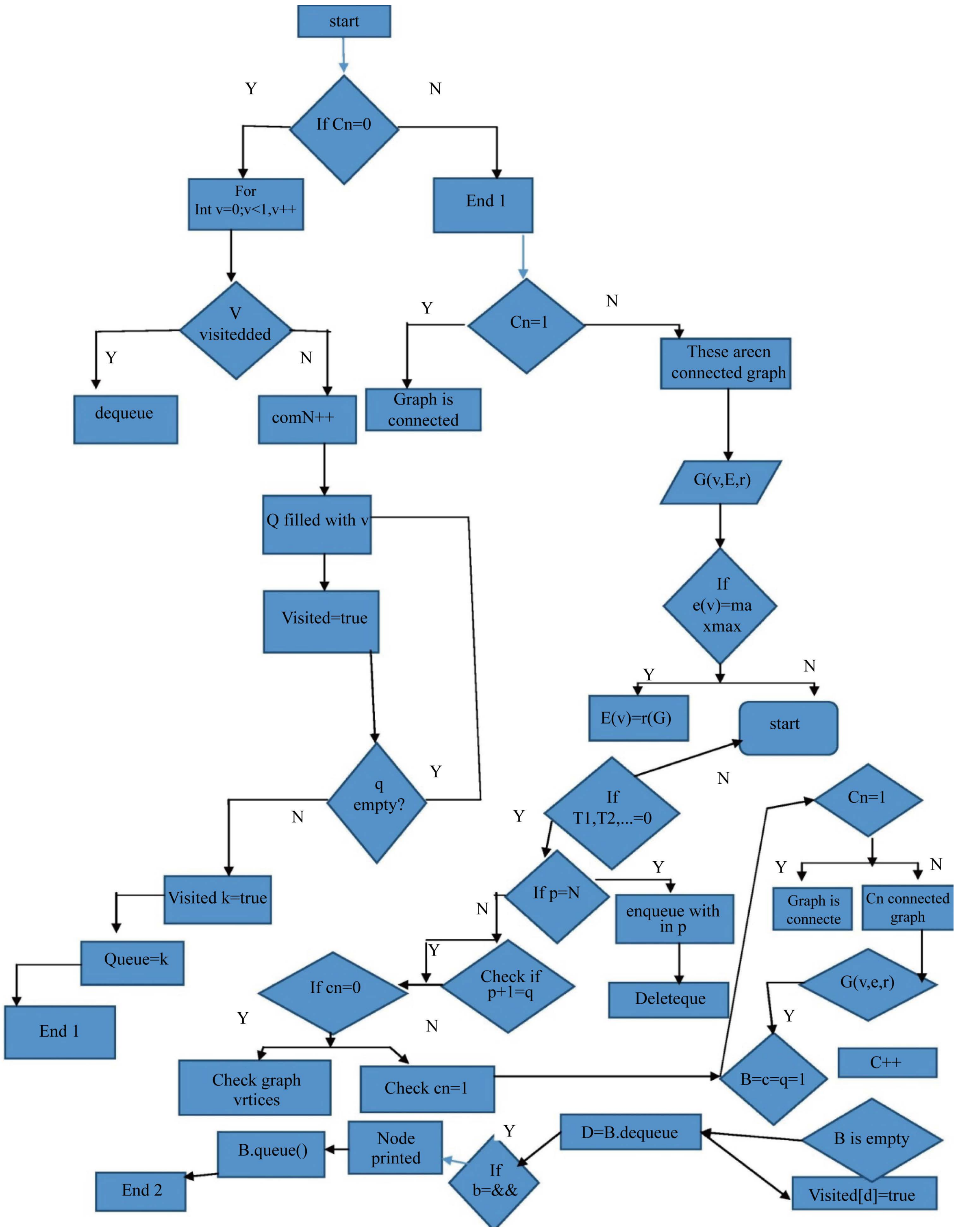

Figure 2. Architecture of the proposed COT-test tree algorithm.

$$\hat{\delta}_N(q_0, w) = \bigcup_{i=1}^{k} \delta_N(p_i, a) \tag{4}$$

The subset construction, on the other hand tells us that,

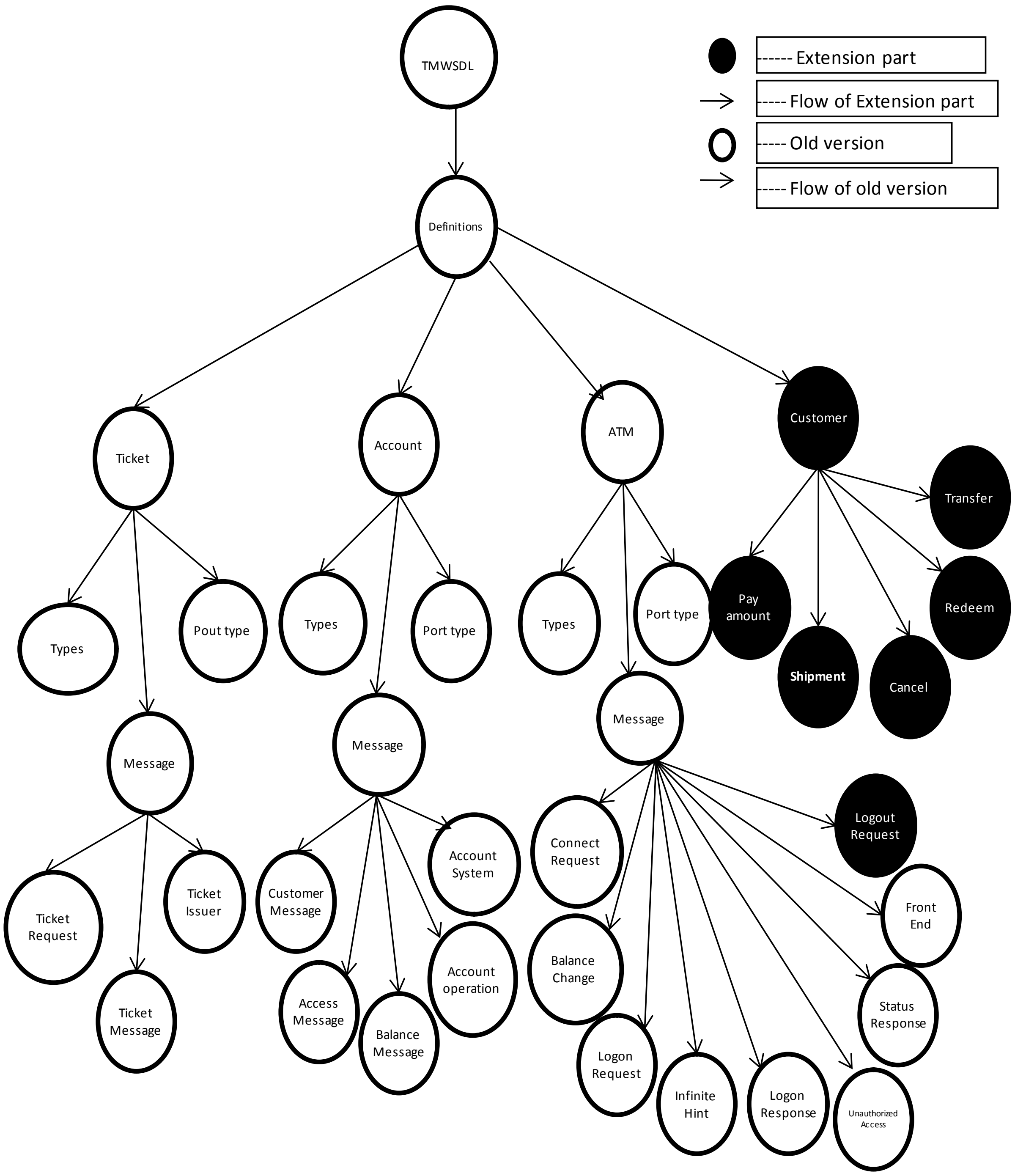

Figure 3. Version 1.0—enhancement part of a BPEL data set.

$$\delta_D\left(\{p_1, p_2, p_3, \cdots, p_k\}, a\right) = \bigcup_{i=1}^{k} \delta_N\left(p_i, a\right) \tag{5}$$

Now, let us use (2.3) and the fact that, $\hat{\delta}_D\left(\{q_0\}, x\right) = \{p_1, p_2, p_3, \cdots, p_k\}$ in the inductive part of the definition of $\hat{\delta}$ DFA is,

$$\hat{\delta}_D\left(\{q_0\}, w\right) = \delta_D\left(\hat{\delta}_D\left(\{q_0\}, x\right)\right) \tag{6}$$

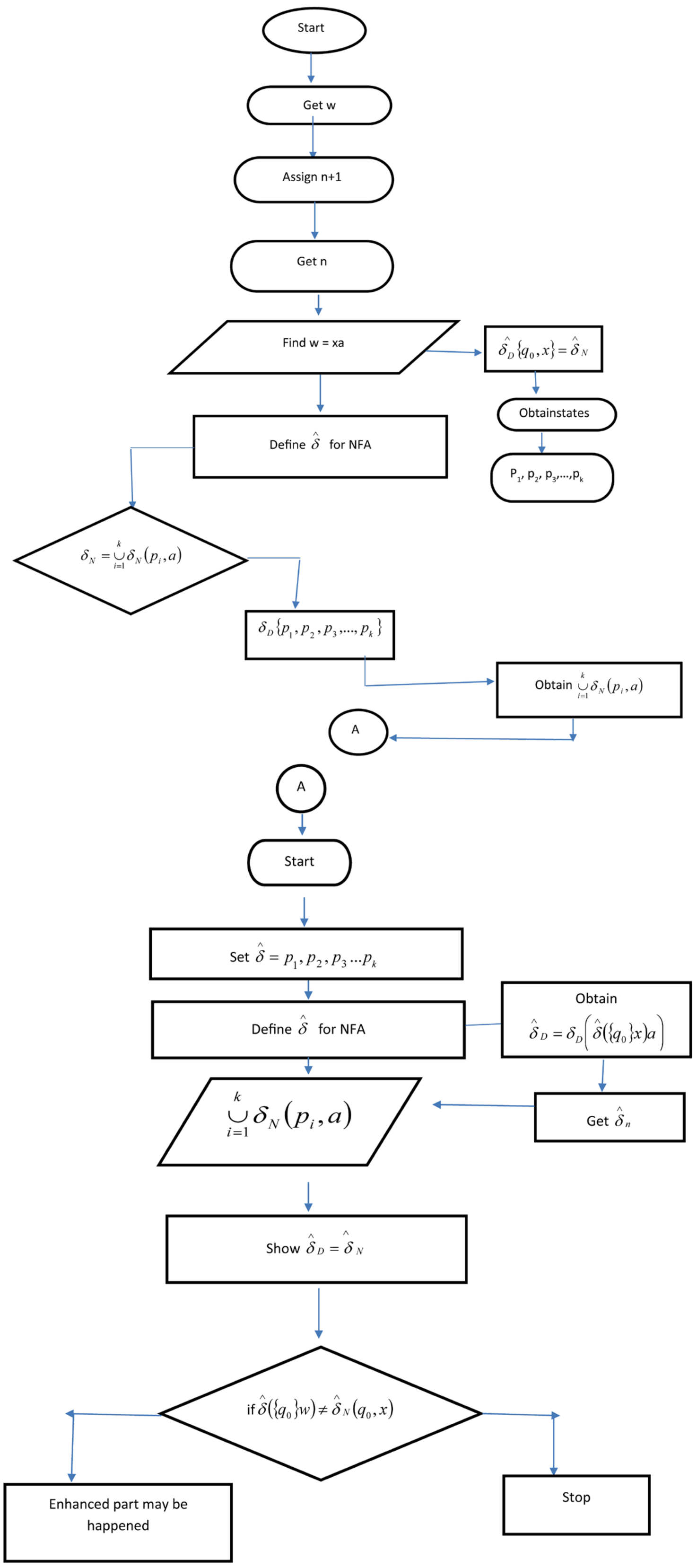
Start
Get w
Assign n+1
Get n
Find w = xa
$\hat{\delta}_D\{q_0,x\}=\hat{\delta}_N$
Obtainstates
$P_1, P_2, P_3,\dots,P_k$
Define $\hat{\delta}$ for NFA
$\delta_N=\bigcup_{i=1}^{k}\delta_N(p_i,a)$
$\delta_D\{p_1,p_2,p_3,\dots,p_k\}$
Obtain $\bigcup_{i=1}^{k}\delta_N(p_i,a)$
A
A
Start
Set $\hat{\delta}=p_1,p_2,p_3\dots p_k$
Define $\hat{\delta}$ for NFA
Obtain
$\hat{\delta}_D=\delta_D\left(\hat{\delta}(\{q_0\}x)a\right)$
$\bigcup_{i=1}^{k}\delta_N(p_i,a)$
Get $\hat{\delta}_n$
Show $\hat{\delta}_D=\hat{\delta}_N$
if $\hat{\delta}(\{q_0\}w)\neq\hat{\delta}_N(q_0,x)$
Enhanced part may be happened
Stop

Figure 4. Flow chart of the proposed PCUA-TEST TREE algorithm.

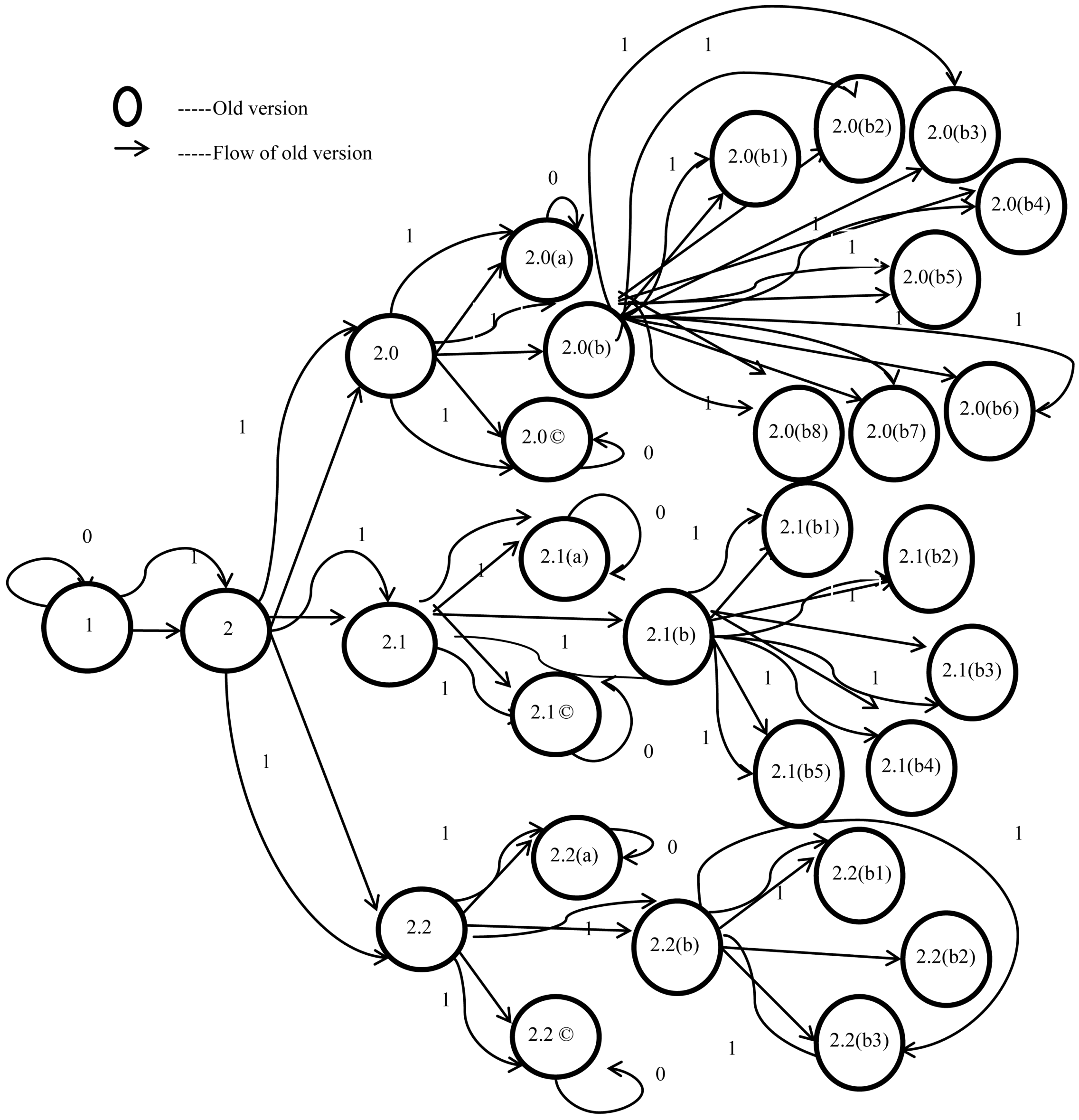

Figure 5. Version 1.0(a)—Old Part of BPEL data set.

$$= \delta_D\left(\{p_1, p_2, p_3, \cdots, p_k\}, a\right) \tag{7}$$

$$= \bigcup_{i=1}^{k} \delta_N\left(p_i, a\right) \tag{8}$$

While, it show where, $\hat{\delta}_D\left(\{q_0\}, w\right) = \hat{\delta}_N\left(q_0, w\right)$ at this state, we consider the part of an old version not obtain any more enhancement part. If the condition is obtain as, $\hat{\delta}_D\left(\{q_0\}, w\right) \neq \hat{\delta}_N\left(q_0, w\right)$. Then, there would be further more enhancement part had been happened at the version of BPEL dataset. By using this technique we compute an algorithm as PCUA-TEST TREE algorithm to detect an error happens in it. The process of constructing the path comparing using automata test tree (PCUA-TEST TREE) for different versions of the BPEL dataset, from PCUA-TEST TREE detection of path existing among the version is performed and computed to find the modifications are detected from the center of the tree by performing COT-TEST TREE technique for various BPEL data set.

Here, **Figure 6** shows that an enhanced part of a BPEL data set by using the condition, $\left(\{q_0\}, w\right) \neq \left(q_0, w\right)$,

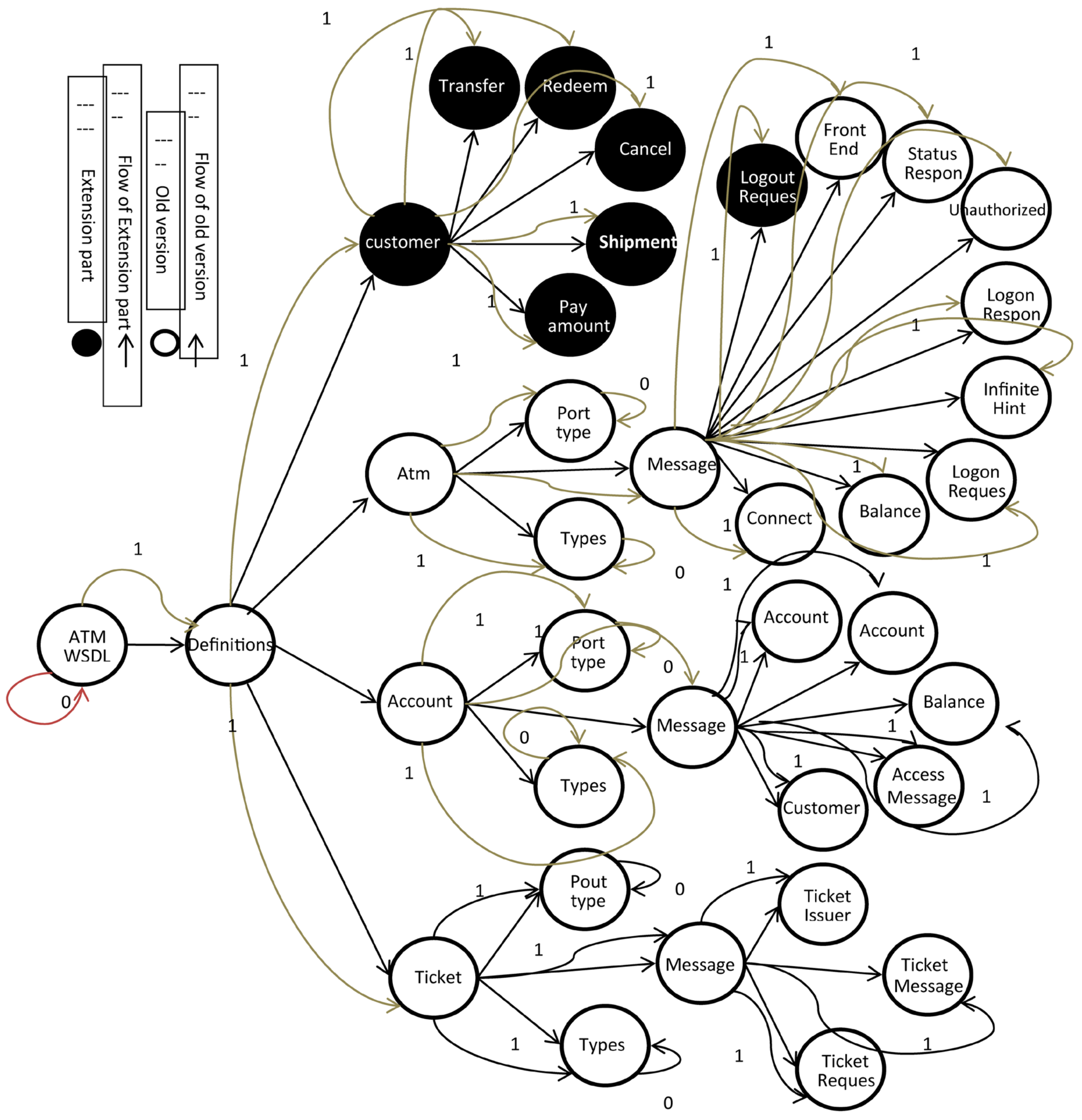

Figure 6. Version 2.0(a) enhancement part of BPEL data set.

the PCUA test tree which able detect the newly enhanced path and also to fix an Error in an efficient manner.

4. Result and Discussion

Figure 7 shows the Time consuming chart to detect an error using the COT test tree algorithm. Along with help of an In-degree and Out-degree in the case of changes happened in Ticket, Account and ATM which has been already shows in the above tree. It shows that, comparing with BPEL service, COT-Test tree which able to detect an error within the minimum spanning of time consumption.

Figure 8 has shown the time interval among the services, from the above graph it has been shown that BPEL service by using PCUA can fix an error within a minimum spanning of time compare to BPEL service by using XBFG service. Hence it has been proved that our proposed PCUA technique consumes less time while fixing errors.

Figure 9 shows the comparative analysis between PCUA and XBFG. The changes in the enhancement part of

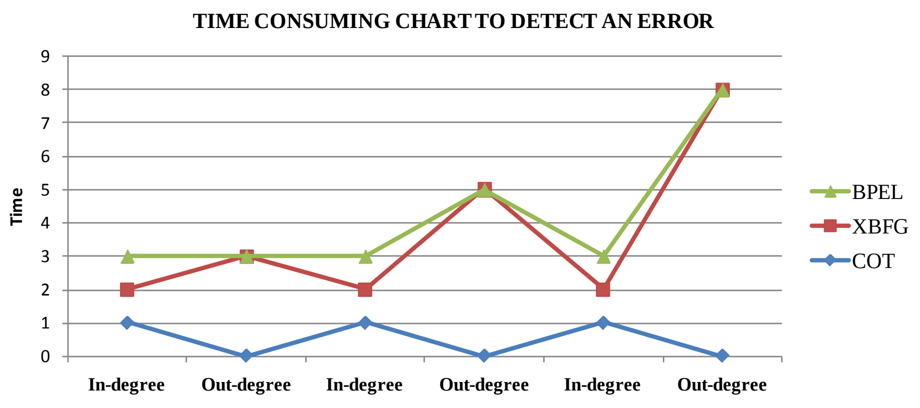

Figure 7. Time consuming chart to detect an error using COT test tree.

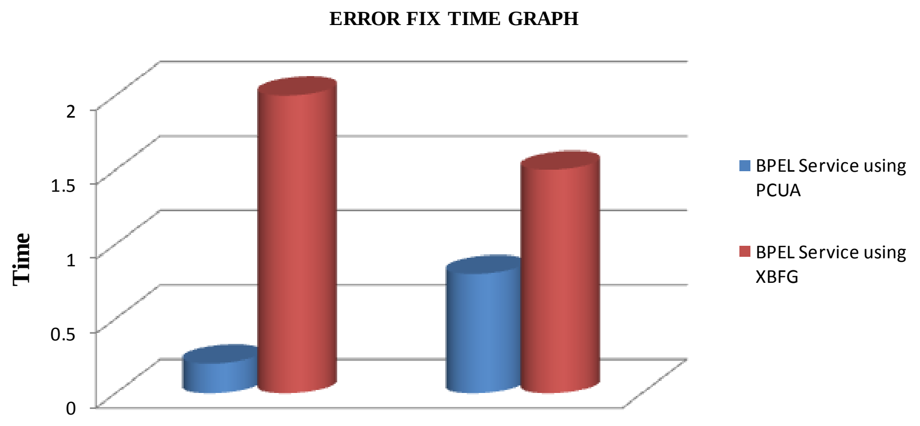

Figure 8. An error fix graph using PCUA test tree.

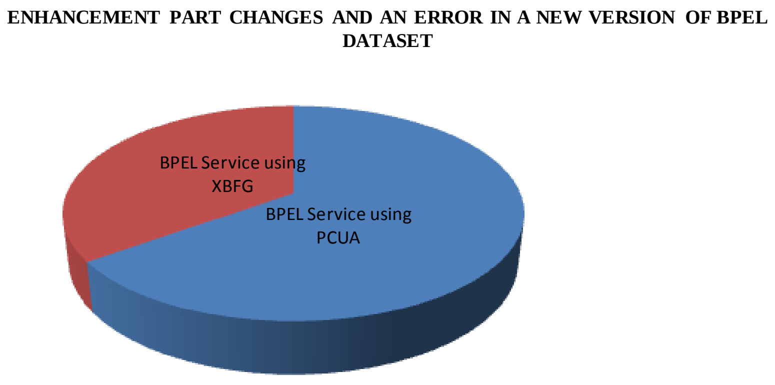

Figure 9. Detected enhancement part changes and an error fix in new version of BPEL dataset.

an application has been analyzed between PCUA and XBFG techniques. From the graph it has been shown that the Proposed PCUA fix more errors in less time consumption when comparing with the conventional BFG techniques. Hence it has been proved and showed that when compare to XBFG, PCUA which able to detect an enhanced path in more efficient manner in a new version of BPEL dataset.

5. Conclusion and Future Work

This paper proposed a COT-TEST TREE algorithm based on path comparison for regression testing using automata to detect an enhancement part of a version of BPEL dataset effectively and another new technique

PCUA-TEST TREE was computed to consume the time taken for detection of path of an enhanced part of a version. The experimental results revealed that our proposed technique performed well when compared with the existing technique. Our future work dealt with non-functional test case equation for the detection of an enhancement part of a version in an efficient manner for BPEL dataset with REST web services.

References

[1] Melnikov, B.F. and Melnikova, A.A. (2001) Edge-Minimization of Non-Deterministic Finite Automata. *Korean Journal of Computational and Applied Mathematics*, **8**, 469-479.

[2] Holzer, M., Kutrib, M. and Leiter, U. (2011) Nodes Connected by Path Languages. *Developments in Language Theory*, **6795**, 276-287. http://dx.doi.org/10.1007/978-3-642-22321-1_24

[3] Fachini, E., Gruska, J., Napoli, M. and Parentem, D. (1995) Power of Interconnections and of Non-Determinism in Regular Y-Tree Systolic Automata. *Mathematical Systems Theory*, **28**, 245-266. http://dx.doi.org/10.1007/BF01303058

[4] Boghdady, P.N., Badr, N.L., Hashem, M. and Tolba, M.F. (2011) Test Case Generation and Test Data Extraction Techniques. *International Journal of Electrical & Computer Sciences*, **11**, No. 82.

[5] Singh, R. and Santosh, M. (2013) Test Case Minimization Techniques: A Review. *International Journal of Engineering Research & Technology*, **2**, 105-156.

[6] Jeyamala, P. (2013) Automatic Test Case Generation and Prioritizing Intended for Regression Testing Using HTT and Multiple Criterion. *ASAR International Conference*, Bangalore, 2013.

[7] Tahbildar, H. and Kalita, B. (2011) Automated Software Test Data Generation: Direction of Research. *International Journal of Computer Science & Engineering Survey*, **2**, 99-120. http://dx.doi.org/10.5121/ijcses.2011.2108

[8] Isabella, A. and Retna, E. (2012) Study Paper on Test Case Generation for GUI Based Testing. *International Journal of Software Engineering & Applications*, **3**, 139.

[9] Yao, G. (2010) Automatic Detection of Infeasible Paths in Software Testing. *Institution of Engineering and Technology*, **4**, 361-370. http://dx.doi.org/10.1049/iet-sen.2009.0092

[10] Li, H., Bat-Eedene, K. and Lee, H. (2013) Software Vulnerability Detection Using Backward Trace Analysis and Symbolic Execution. 2013 *Eighth International Conference on Availability, Reliability and Security* (*ARES*), Regensburg, 2013, 446-454.

[11] Najumudheen, E.S.F. and Samanta, D. (2009) Dependence Graph-Based Test Coverage Analysis Technique for Object-Oriented Programs. 6*th International Conference on Information Technology*: *New Generations*, Las Vegas, 27-29 April 2009, 763-768. http://dx.doi.org/10.1109/itng.2009.284

[12] Mishra, P. and Dutt, N. (2004) Graph-Based Functional Test Program Generation for Pipelined Processors. *Proceedings on Design, Automation and Test in Europe Conference and Exhibition*, **1**, 182-187.

[13] Hewett, R. and Kijsanayothin, P. (2009) Automated Test Order Generation for Software Component Integration Testing. 24*th IEEE/ACM International Conference on Automated Software Engineering*, Auckland, 2009, 211-220.

[14] Verma, D. (2004) Component Testing Using Finite Automata. *Indian Journal of Computer Science and Engineering*, **3**, No. 658.

[15] Davis, N. (1985) Fault Location Techniques for Distributed Control Interconnection Networks. *IEEE Transactions on Computers*, **C-34**, No.10.