Analyzing Artifact Anomalies in a Temporal Structural Workflow for SBS

Feng-Jian Wang[1]
Parameswaramma Mandalapu[2]
Dept. of Computer Science
National Chiao Tung University
Hsinchu, Taiwan
[1]fjwang@cs.nctu.edu.tw
[2]Parameswari.as400@gmail.com

***Abstract*— A service-based system (SBS) defined with minimum and maximum execution time can be easily transferred into a temporal structured workflow. The analysis technique on a temporal structured workflow can thus be applied on SBS. In the past, there were several researches working on artifact anomaly detection in a workflow. Their results are useful and have been published. However, their works does not consider temporal factor and the anomalies detected may not exist in a real system. In other word, they are less effective in a temporal structural workflow (TS workflow). Neither for SBS. Besides, the time complexity of these methods are NP, not efficient either. In this paper, we re-define the anomalous behaviors and develop an approach to discover artifact anomalies in a TS workflow. In the approach, we design several algorithms to detect the anomalies defined. By using our approach, workflow and SBS designers can detect artifact anomalies more precisely inside their TS workflow and thus might prevent run-time errors more effectively.**

Keywords: workflow, temporal structured workflow, artifact anomaly, anomalous data manipulation

I. INTRODUCTION

Consider a service-based system (SBS), a remote service can be useful only if the work can be done within a predicted time interval. Such a constraint of SBS's services is like time constraints in a real-time system, and a pair (minimum execution time, maximum execution time) is useful for the selection of services. On the other hand, a workflow where each process is associated with a pair of (min, max) execution time interval is named as a temporal (structured) workflow. Therefore, an SBS can be simplified as a temporal workflow system since the execution time intervals of services are an important factor for the selection of each service. The analysis based on a temporal workflow can be applied or studied further to help improve the development of an SBS.

In the past, there were lots of research results presented for workflow analysis. Typical examples of control analysis include the detection and deletion of structural conflicts among tasks, inconsistent dependencies [1], verification of deadlock, live locks (infinite loops), and dead tasks in workflow specifications [2-3] by mapping workflow specifications into Petri-nets. In previous study, [4] defined the structured workflow model which is free from deadlock and multiple active instances of the same activity and claimed that most arbitrary well-behaved workflows can be transformed into a structured workflow for analysis. Besides, a temporal factor is introduced to improve the related control analysis. [1], [5-9]. Many control analysis works on a temporal workflow are done based on timed Petri Nets, translated from workflows. On the other hand, a structured workflow may produce an unanticipated run-time behavior because of abnormal data manipulation, named artifact anomalies. Detecting artifact anomalies in a workflow can help checking data misuse inside the workflow. Various methodologies have been developed for detecting artifact anomalies between activities in a structured workflow [10-14].

The above approaches work based on a general assumption: There is no explicit lower and upper bounds of time consuming for each process to count the feasibility or correctness of control in a workflow. However, the temporal data associated with the processes inside an SBS do help on static analysis of concurrency. For example, Li and Yang [15] analyzed the resource and temporal constraints between distinct process instances.

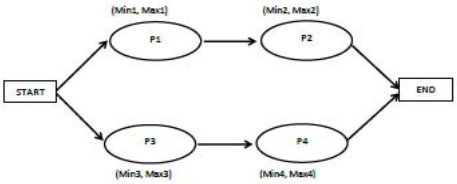

Fig.1. A simple temporal workflow example

Figure.1 indicates a parallel temporal workflow diagram where each process pi, 1<=i<=4, is associated with an estimated working duration (mini, maxi), i.e., the (minimum execution time, maximum execution time) of pi. Let the value of artifact 'a' is deleted (such an operation can also be called undefined, and noted as "u" in the paper), in p1, defined (noted as "d") only in p2, read ("r") in p4 and not operated in p3. Along the path, s, p1, p2, to t, the operation sequence on a are "ud" and such a behavior is normal in logic. On the other path: s, p3, p4, to t, 'a' is read without definition, and the behavior is abnormal logically. However, these two paths are concurrent, and "r" for 'a' in the second path may or may not appear after 'd' completes in path one. Thus "ud" may not appear during execution. For example, consider the estimated execution time, if max1+max2<min3, the "r" operation for 'a' cannot occur before "d" since p4 cannot work before p2 completes and there

978-1-4799-3578-9/14 $31.00 © 2014 IEEE
DOI 10.1109/COMPSACW.2014.81

is no artifact anomaly for 'a' in this workflow diagram. Obviously, the anomaly detection might be more precise if the estimated execution time is added.

In this paper, we first analyze existing approaches for deficiency and anomaly detection on a workflow. Then, we give a set of new definitions for artifact abnormal behaviors based on our observation from SBSs. With the definitions, we construct a serious of algorithms to detect the corresponding anomalies in a workflow. Based on temporal data, we further present the corresponding algorithms to detect the anomaly more effectively, i.e., by deleting the anomalies which never occur once the temporal data are added at each process. Finally, we compare our approach with existing ones.

The rest are organized as followings. In section 2, a conventional TS workflow is described, and the structural and temporal relationships between processes are analyzed. In section 3, the abnormal behavior due to continuous operations for the same artifact is introduced. Section 4 presents a methodology to detect these artifact anomalies in workflow. In section 5, the methodology to detect the anomalies inside temporal workflow is presented. Finally, the conclusion and future work are described in section 6.

II. A TEMPORAL STRUCTURED WORKFLOW

2.1 Fundamental Techniques for Workflow

A workflow contains a set of tasks systematized to achieve certain business goals by completing the tasks in a particular order under automatic control [16]. Structural conflicts among tasks such as deadlocks might cause run-time errors, and need be eliminated, if necessary. There are many methods developed to detect structured conflict(s), such as inconsistent dependencies [1], the verification of deadlock, live-locks (infinite loops), and dead tasks in workflow specifications [2-3] in a workflow by mapping workflow specifications into Petri-nets. [4] Defined the structured workflow model which is free from deadlock and multiple active instances of the same activity and claim that most arbitrary well-behaved workflows can be transformed into a structured workflow for analysis. Besides, a temporal factor is introduced to improve the related analysis. [1], [5-9].

A structured workflow may produce an unanticipated run-time behavior because of abnormal data manipulation, named artifact anomalies. Detecting artifact anomalies in a workflow can help checking data misuse inside the workflow. Various methodologies have been developed for detecting artifact anomalies between activities in a structured workflow [10-14]. [10] Present seven basic data validation problems to be detected. [12] Defined preliminary improper artifact usages anomalies, and introduced the analysis of such anomalies in design phase of a well-structured workflow [11, 12]. [13] Introduced a model to describe the artifact behavior in a workflow to improve the efficiency of the work in [12]. [14] Analyzed artifact anomalies in workflows by adopting message passing data models.

2.2 Basic elements for a Structured Workflow

Based on (WfMC, 2010), a workflow diagram is a four tuples W= (N, A, S, E), where N is a set of nodes, of which each represents a process. A is a set of directed arcs, where each arc connects two nodes to represent the control (flow) from the tail process to the head. There are 4 types of control processes, split, joint, begin and end processes. Besides, a process is a complicated process (CP) if it can be decomposed into another workflow diagram or an activity process (ACT) if it contains one or a sequence of activities only. To simplify the discussion, the complex process is skipped in the paper.

A split process is a process which instantiates its successor(s) when its work completes. There are two types of split processes defined in general: an And-Split (AS) process instantiates all its immediate successor processes while an Xor-Split (XS) process instantiates only one of them. There are two types of Joint processes: an And-Joint (AJ) process is instantiated when all of its immediate predecessor processes complete, while an Xor-Joint (XJ) process is done whenever one of them completes.

Figure.2 is a sample structured workflow; the path <v1, xs1, v2, as1, v3> indicates that v1 is reachable to v3. v3 and v4 are parallel because they reside on different "and" branches split from as1. v2 and v8 are exclusive because they reside on different branches of the decision structure quoted by xs1 and xj1. The path <ls1, v6, v7, le1, ls1> indicates a loop. In this paper, each control process of structural relationships is associated with a Boolean function.

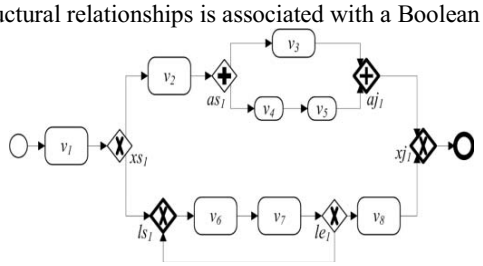

Fig.2. A Sample Structured Workflow with loop

[12, 13] claimed that in a structured workflow, the states of the artifact operated in loops cannot increase after the iteration is done twice.

2.3 A Temporal Structured Workflow

A temporal workflow is modeled by describing the maximal and minimum working durations for each activity or process [17]. In this paper, a timed and structured workflow named as Temporal Structured Workflow (TS workflow). To facilitate discussion, we assume that if p is an activity process, $0 < d(p) \le D(p)$; otherwise, $d(p) = D(p) = 0$. Figure.3 illustrates a sample TS workflow.

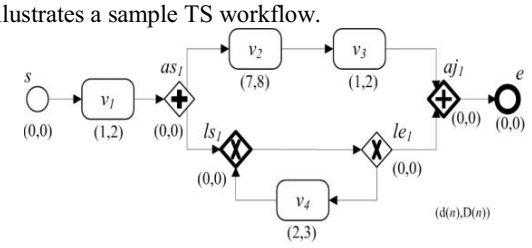

Fig.3. A Sample TS workflow

The structural and temporal relationships between processes are the bases for the analysis in a TS workflow. There are many approaches [10, 11] adopted to reduce the structured loops in a TS workflow as decision structures. However, the loop reduction may bring inaccuracy to the analysis of temporal data, and is therefore not feasible for a TS workflow. [18] Developed a methodology for loops to detect whether the workflow possibly exceeds its deadline during run-time. However, it is still an NP problem to detect all possible anomalies based on above definition in a TS workflow, we discuss the artifact anomalies from another point view in this paper to simplify the detection.

III. ANOMALOUS BEHAVIORS OBSERVED IN A TEMPORAL WORKFLOW

There are various artifact anomalies observed on workflow described in Section 2. The categorizations do not consider the concurrency factor, since there is no event to indicate whether two processes in two distinct parallel paths can be executed in a sequential order. Thus a fundamental concurrency assumption for workflow is that the processes in two parallel paths are concurrent. Conventionally, the artifact anomalous behaviors are usually categorized based on execution order only and few papers consider the temporal factor. Their detection techniques are lack of the abnormal behaviors due to the temporal factor.

Based on our observation, workflow anomalies in a temporal workflow in section 2 can be categorized into as following:

1. *Concurrent anomalies:*
   If two activities for the same artifact are concurrent, the two activities, (W, W) and (K, W) are abnormal obviously, and (K, R) is a potential anomaly since R might be after K. There are three sets CCA1, CCA2, and CCA3, used to represent each type of the anomalies defined above, where each element in these sets contains three tuples, the first one is an artifact and the rest two represent two concurrent nodes which contain activity W for the artifact. For example, artifact x has an activity W in nodes m and n, if and only if the element (x, m, n) is in CCA1. Algorithms in Sections 4.1 and 5.1 are applied to detect the concurrent anomalies and put them into CCA1, CCA2, and CCA3 correspondingly.
2. *Continuous Anomalies:*
   Two continuous activities are abnormal, if they are (K, K), (K, R), (K, W), and (W, W). There are four sets CNA1, CNA2, CNA3 and CNA4, used to represent each of the anomalies defined above. Similarly, each element in these sets contains three tuples correspondingly. For example, artifact x have two continuous activities (K, K) run in two distinct nodes, m and n, if and only if (x, m, n) is in CNA1. Algorithm in Sections 4.2 and 5.2 are used to detect the continuous anomalies and put them into CNA1, CNA2, CNA3 and CNA4 correspondingly.

Before considering the algorithms to calculate the concurrent set from a workflow W, we define the following items to help the description of the algorithms:

1. An execution path *EP*, is a linked list where the LET of the former is less than the LET of the later in the EP. A *CEP* is an EP who's starting and ending nodes are the starting and ending nodes of the workflow correspondingly. A set containing all CEPs in W is called W's CEP_set.
2. A concurrency pair set *PS*, PS= {(x, y) | x, y are two processes in W, $x \neq y$, x and y are concurrent}, is used to simplify input/output. A full PS, *FPS*, contains all the concurrency pair set, FPS= {(x,y)| $\forall$(x,y) in W and (x, y)$\in$PS}.
3. The concurrency process set of a process x, *CPS*(x) = { y | $\forall y$ in W, $x \neq y$, x and y are concurrent}
4. A *branch* from a split node include all the EP's which have the same starting node, a distinct immediate successors of the split node to the immediate predecessor(s) the corresponding joint node of split node. Two branches with the same split node are named as *cs_branches*.

IV. DETECTING ANOMALIES IN A WORKFLOW

4.1 Detection of Concurrent Anomalies

For workflow W, Algorithm 4-1 can be applied to return a PS on W. The algorithm is recursive and done based on depth first approach. In the algorithm, when it reaches an AND approach, it collects the pair(s) for the nodes in one branch with the nodes in the rest right cs_branches. When p is an XOR or sequence node, the algorithm is done by calling each of p's successors left to right, i.e., calling Construct_Potential_Concurrent_Pairs_Set(W, p's successor, PS).

```
Algorithm 4-1:Construct_ Potential_Concurrent_ Pairs_ Set
Input: a TS workflow W, a process p, an empty set of
concurrency pair set PS
Output: FPS={(x,y)| ∀(x,y) in W and (x, y)∈ PS}
Begin
01.  if (p is an AND split process) {
02.        for each immediate successor of p, from left to right, x do {
03.        make each node in the branch starting from x and
04.        each one in the right cs_branches as a pair and put it in PS};
05.         PS=Construct_Potential_Concurrent_Pairs_Set(W, x, PS)};
06.   else if (p is an XOR split process) {
07.        for each immediate successor of p, from left to right, x do
08.        PS=Construct_Potential_Concurrent_Pairs_Set(W, x, PS) };
09.        else{ PS=Construct_Potential_Concurrent_Pairs_Set(W,
                 p's successor, PS) };
10.  Return PS; //**in a structured workflow,**//
End.
```

Fig.4. Algorithm 4-1

After applying Algorithm 4-1, it is returned the FPS in W. According to artifact set A, the set of artifacts applied in W, Algorithm 4-2 accepts W and the concurrency pair set FPS in W, and A and replies CCA1, CCA2, and CCA3 by detecting the property in each pair of PS.

```
Algorithm 4-2: Detect _Concurrent _ Anomalies _in_ a_ Workflow
Input: W, a workflow; FPS, a full PS in W; A,
the set of artifacts applied in W;
Output: CCA1, CCA2, and CCA3;
Begin
01.  CCA1 = Ø;
02.  CCA2 = Ø;
03.  CCA3 = Ø;
04.  for each artifact a in A do
05.        for each element (x, y) in FPS do{
06.        If a has activity W in both nodes x and y, put (a, x, y) in CCA1;
           If a has activities K and W, one in each node, put (a, x, y) in CCA2;
           If a has activities K and R, one in each node, put (a, x, y) in CCA3}
07.  Return CCA1, CCA2, and CCA3;
End.
```

Fig.5. Algorithm 4-2

```
Algorithm 4-3: Construct_All_Static_CEP_Set
Input: a workflow W; a path zp from starting node, represented by an EP;
Pset = {x, | x, is a CEP in W, and i>=0}
Output: CEP_set={ x |∀x, x is a CEP}
Begin
01.    if (zp contains a starting node only) {
02.        if (the starting node in W has all immediate successor marked)
03.              return Pset
04.        else { find an unmarked immediate successor of the starting
                      node in W
05.              Unmark the successor and all of successors in W;
06.              append a corresponding node to zp;
07.              Pset=Construct_All_Static_CEP_set(W, zp, Pset)}
08.    else if (zp's tail corresponds to the ending in W) {
09.        replicate a new path zp' from zp;
10.        put zp as an element in Pset;
11.        repeat {delete the tail from zp';
12.            if (the node in W for the deleted tail has an unmarked
                   right sibling){
13.               append a corresponding one at zp for the sibling;
14.               unmark all successors of the sibling in W ;
15.               Pset=Construct_All_Static_CEP_set(W, zp, Pset)}
16.        until (zp' is containing the starting node of W only);
17.        Pset = Construct_All_Static_CEP_set(W, zp, Pset)}
18.    else if (zp's tail corresponding to a split node in W) {
19.        find a unmarked leftmost immediate successor in
           W for the tail in zp;
20.        if the successor exists{
21.           {append a corresponding node at zp;
22.           unmark all successors in W of the successor;
23.           Pset=Construct_All_Static_CEP_set(W, zp, Pset)}
24.        else {
25.              delete the tail in zp;
26.              Pset=Construct_All_Static_CEP_set(W,zp, Pset)}}
27.        else {
28.                 mark the immediate successor in W;
29.                 append a corresponding one at zp;
30.                 Pset=Construct_All_Static_CEP_set(W, zp, Pset)}
End.
```

Fig.6. Algorithm 4-3

```
Algorithm 4-4: Detect_ Continuous_ Anomalies_ Along_ Static_ Paths
Input: W, a workflow; CEP_set, the set derived by
Construct_All_Static_CEP_set (W); A, the set of artifactsapplied in W;
Output: CNA1, CNA2, CNA3, and CNA4 described as above
Begin
01.  CNA1 = Ø;
02.  CNA2 = Ø;
03.  CNA3 = Ø;
04.  CNA4 = Ø;
05.  for each artifact a in A do
06.       for each element p in CEP_set do{
07.                 n = the initial node in p;
08.       while (a has no activity in n and n's successor is not NULL)
          do{n = n's succ};
09.                 if a has an activity in n {
10.                 m = n's succ;
11.                 while (m<>null) do{
12.                 if (a has an activity in m){
13.                 if (both activities in (n, m) are one of above 4
                           CAN types)
14.                 put (a, n, m) into the corresponding set};
15.                       n = m;
16.                       m = m's succ
17.                             }
18.                       }
19.                 };
20.  return (CNA1, CNA2, CNA3, CNA4);
End
```

Fig.7. Algorithm 4-4

4.2 Continuous Anomaly Detection in a Workflow

Algorithm 4-3, composed of three parameters described below, is applied on workflow W to find W's CEP_set based on a depth first concept. In the very beginning, zp is set as an empty path and Pset is an empty set of path. The algorithm is recursive: when all nodes are reached, it completes and CEP_set is returned. During the algorithm, if a CEP is found, i.e., the ending node is reached, the CEP is sent into Pset and control goes back to the nearest split node which has a successor not reached yet and the forward searching continuous from the successor. To clear the past record, the successors of the node found in W at this step are unmarked before starting forward search. In the search, if the next node is an activity (simple) node or joint node, a corresponding node is add to the end of zp and the search continues.

After applying Algorithm 4-3, all the static CEPs are put together as a set Pset and returned as CEP_set. Along with CEP_set and artifact set 'A' in workflow W, Algorithm 4-4 returns the anomalies along all the CEPs.

V. DETECTING ANOMALIES IN A TEMPORAL WORKFLOW

CNA and CCA are too rough to be useful in SBS's because many anomalies detected based on this model do not exist since the temporal data associated with each process might kill the corresponding continuous behaviors. Therefore, detecting both anomalies in a temporal workflow is worth of being studied further.

5.1 Detection of Concurrency Anomalies in a Temporal Workflow

As defined in Section 2, each process has its execution time interval (Min, Max) in a temporal workflow. Thus, the earliest starting time (EST) and latest ending time (LET) for each process can be computed too [19]. Based on this time pair, if two processes whose execution time cannot be overlapped, there is no CCA anomaly between them. Therefore, a process pair (x, y) detected to be concurrently in Algorithm 4-1 might not be executed concurrent if one of the following conditions holds:

(x.LET < y.EST) or (x.EST > y.LET)

```
Algorithm 5-1: Construct_Temporal_FPS
Input: a temporal workflow TW;
Output: TFPS, where TFPS= {(x, y) |∀(x, y)  FPS and ¬
 ((x.LET<y.EST) or (x.EST> y.LET))}
Begin
01.  TFPS = Construct_Potential_Concurrent_Pairs_Set
       (TW, starting process in W, PS= Ø)
02.  for each element p in TFPS do {
03.  if(p.x.LET <p.y.EST) or (p.x.EST>p.y.LET)
04.  TFPS = TFPS – {p}}
05.  Return TFPS;
End.
```

Fig.8. Algorithm 5-1

On the other hand, if none of these conditions hold, it indicates that x and y might be executed concurrently in a temporal workflow. TFPS is a temporary FPS, where each element in FPS follows the above rule. Algorithm 5-1 is defined to extract the pairs of processes whose execution time must be overlapped, counted based on EST and LET from FPS derived from Algorithm 4-1.

Lemma 5-1:

In a temporal workflow, a concurrent pair detected with Algorithm 4-1 can be detected in with Algorithm 5-1 too.

Proof:

Line 1 in Algorithm 5-1 indicates TFPS is the same as FPS for the same temporal workflow. Lines 2-4 in Algorithm 5-1 indicates whenever the condition in Line 3 succeeds, the corresponding element p, a pair of processes which cannot run concurrent based on the calculation of execution time, is deleted from TFPS. Therefore, TFPS is a subset of FPS, and the lemma is OK.

Lemma 5-2:

In a temporal workflow, an element detected with Algorithm 4-1, might be detected in Algorithm 5-1.

Proof:

Lemma 5-1 indicates TFPS is a subset of FPS; therefore Lemma 5-2 works too.

Based on Lemmas 5-1 and 5-2, for a temporal workflow, the work done with Algorithm 5-1 is more precise than that with Algorithm 4-1. Algorithm 4-2 can be applied to detect the concurrency anomalies in a temporal workflow, since the concurrency anomaly (ies) exists as long as the artifacts activities are the same and their execution time overlap exists. The anomaly detection is thus more precise here, since the work is done on the set of activities whose concurrency might occur.

5.2 Detection of Continuous Anomalies in a Temporal Workflow

CEP constructed in Algorithm 4-3 might not work in a temporal workflow. For example, before an AND joint process, if the EST of the last process in one branch is larger than the LET of the last process in another branch, there exists no CEP containing the latter process and the joint process as two continuous processes. Therefore, such a CEP in a temporal workflow does not exist and the anomaly detection is not necessary for the CEP. First delete these CEPs based on the following two steps:

1. Compute the EST and LET for each process first;
2. For each and joint process, backtrack all its concurrent branch as follows:

   If the LET of the last process in one branch is less than the EST of the last process of another branch, the CEP containing two continuous processes: the last process in the former branch and the joint process are deleted.

The rest CEPs are called TCEPs. Here, Algorithm 5-2 is designed to calculate the EST, LET, and AJE of each process in a workflow and booted before executing above deletion work in Algorithm 5-3 systematically. Especially, AJET is the time that represent the largest EST among the last processes of the branches entering to AND joint process, thus its computations are done only at an AND joint nodes. The set returned from Algorithm 5-3 is named as TCEP_Set.

```
Algorithm 5-2: Construct_PTE_in_a_TW
Input: TW: a temporal workflow, where the EST, LET, and AJE are set to
 be zero. p: a process in TW.
Output: TW: the same as input temporal workflow, but the EST and LET
of each process are computed.
Begin
01.  switch p is
02.  "Startingprocess":{p.EST=0;    p.LET=0;TW=Construct_PTE_in_a_TW
          (TW,  p.succ)};
03.  "endingprocess":{p.EST=p.pre.EST;p.LET=p.pre.LET; return TW};
04.  "AND split process":{p.EST=p.pre.EST+p.pre.MIN;
05.                p.LET=p.pre.LET;
06.                for each of p's successors, s, do{
07.                    TW = Construct_PTE_in_a_TW (TW, s)}};
08.  "AND joint process":{if (the LET all p's predecessors are not "0") {
09.               if (the largest one in{ x | ∀x=p.pre.EST} < the smallest one in
10.              { x | ∀x=p.pre.LET}
11.               p.EST= the smallest one in { x | ∀x=p.pre.LET}
12.              else p.EST = the largest one in {x | ∀x=p.pre.EST};
13.               p.LET=the largest one in {x| ∀x=p.pre.LET}
14.               p.AJET=the largest one in {x| ∀x=p.pre.EST}
15.               TW=Construct_PTE_in_a_TW(TW, p.succ)}
16.              else return TW;};
17.  "XOR split process": { p.EST=p.pre.EST+p.pre.MIN;
18.               p.LET=p.pre.LET;
19.               for each of p's successors, s, do
20.               { TW=Construct_PTE_in_a_TW (TW, s)}};
21.  "XOR joint process": {if (the LET all p's predecessors are not "0"){
22.               p.EST= the smallest one in { x | ∀x=p.pre.LET};
23.               p.LET=the largest one in {x| ∀x=p.pre.LET};
24.               TW=Construct_PTE_in_a_TW(TW, p.succ)}
25.               else return};
26.  "Activity process": {p.EST=p.pre.EST+p.pre.MIN;
27.               p.LET=p.pre.LET+p.MAX
28.              Construct_PTE_in_a_TW (TW, p.succ)}
End.
```

Fig.9. Algorithm 5-2

```
Algorithm 5-3: Construct_TCEP_Set
Input: a temporal workflow TW;
Output: TCEP_Set in TW
Begin
01.  Set the EST and LET of each process inTW to be 0;
02.  TW=Construct_PTE_in_a_TW(TW, the starting process of TW);
03.  CPS = Construct_All_Static_CEP_Set(TW, the starting process of TW,
          CEPset=∅);
04.  for each CEP in CPS do {
05.      Let the CEP be pointed by x;
06.      discontinuous = false;
07.      while (x <> null and not discontinuous) do {
08.          if (the process pointed by x is an AND joint process) {
09.            if (x^.pre^.LET <= x^.AJET) discontinuous=true
10.            else x=x^.succ}
11.          else x=x^.succ};
12.  if (discontinuous = true) delete the CEP from CPS};
13.  return CPS;
End
```

Fig.10. Algorithm 5-3

Lemma 5-3:

The potential execution time intervals of two continuous processes in each element of TCEP_Set, i.e., constructed by Algorithm 5-3 are overlapped.

Proof:

Each element in TCEP_Set, i.e., each TCEP, is a CEP. Since CEPs are derived from a workflow diagram, by default, two continuous nodes in a CEP represent two distinct nodes connected by an arc in a workflow diagram. After the possible execution time is counted, for an AND joint node, the LET of one of its immediate predecessors might be less than the EST of another immediate predecessor. If such a case occurs, the former immediate predecessor cannot work right before the joint process. Therefore, such a CEP cannot run correspondingly. Algorithm 5-4 deletes this kind of CEPs. The rest CEPs, TCEP, does not allow the joint node to have the case, i.e., since the execution time interval between an AND joint node and each of its predecessors is overlapped based on the viewpoint of EST and LET.

Consider Algorithm 4-4, if the first two inputs are changed as a temporal workflow and its TCEP_Set, the results returned are also the sequence anomalies for the element in the TCEP. Obviously, for a workflow W and its temporal workflow TW, TCEP_Set is a subset of CEP_Set, and Algorithm 4-4 applied to TCEP_Set will return less anomalies according to lemma 5-3.

Let ATCEP be a TCEP where each node of ATCEP is additionally associated with a set (named as Tconcur_set) of processes being concurrent with the process represented by the node according to temporal data: EST and LET. In other word, each element of the set associated with a node in ATCEP and the process represented by the node is an element can be derived by Algorithm 5-1. Algorithm 5-4 is applied to compute ATCEP_Set, the set for ATCEP's where each of its elements corresponds to a distinct TCEP. Algorithm 5-5 is applied to detect the continuous anomaly in each element of ATCEP_Set.

```
Algorithm 5-4: Construct_ATCEP_Set
Input: a temporal workflow TW;
Output: ATCEP_Set in TW
Begin
01.  TCEP_Set = Construct_TCEP_Set(TW);
02.  ATCEP_Set = ∅;
03.  TFPS = Construct_Temporal_FPS(TW);
04.  TFPS={ (x, y)| ∀(x,y) FPS and ¬((x.LET<y.EST)
          or (x.EST> y.LET))}
05.  For each element in TCEP_Set do {
06.  Instantiate a new element in ATCEP_Set
          which is pointed by x;
07.  Construct x's path and copy the data into each
          node according to the element in TCEP_Set;
08.  While x<>null do{
09.          x^.Tconcur_set = {y|∀(x^.process,y)
          or (y,x^.process) TFPS} ;
10.          x = x^.succ}
11.  };
12.  return ATCEP_Set;
End
```

Fig.11. Algorithm 5-4

```
Algorithm 5-5: Detect_Continuous_Anomalies_along_ATCEPs
Input: TW, a temporal workflow;
ATCEP_set, the set derived with Construct_ATCEP_Set(TW);
A, the set of artifacts applied in TW;
Output: CNA1, CNA2, CNA3, and CNA4 described as above
Begin
01.   CNA1 = ∅;
02.   CNA2 = ∅;
03.   CNA3 = ∅;
04.   CNA4 = ∅;
05.   for each artifact a in A do
06.   for each element p in ATCEP_set do{
07.       n = the address of the starting node in p;
08.       Act(a) = { (x, y) | x = a's activity and y is n or n's
          Temporal concurrent process,where x occurs};
09.   While (n^.succ <> null) do {
10.   if (Act(a)==∅) {Act(a) = {(x, y)| x=a's activity and y is n or
                n's temporal concurrent process where x occur};
11.       n=n^.succ}
12.   else if (a has an activity in m, where m is n's successor or
              the successor's temporal concurrent process) {
13.         for each element in Act(a) {
14.         if ((the element's activity (x), a's activity in m)
               is one of above 4 CNA types)){
15.         if (the element's y <>m)
16.         put (a, the element's y, m) into the corresponding set};
17.         if (a has an activity in n's successor){
18.         Act(a) =Act(a)-{(x,y)|y is n or a predecessor of n}
              ∪{(a's activity x in n's successor, n's successor };
19.         if (the element's y is not the temporal concurrent process
            of n' successor){ Act(a) = Act(a) – {the element} }};
20.         For (each temporal concurrent process of n's successor
                that has an activity on a) do{
21.         Act(a)=Act(a)∪{(the activity on the process, the process)}};
22.                 };
23.               };
24.           };
25.   return (CNA1, CNA2, CNA3, CNA4);
End
```

Fig.12. Algorithm 5-5

VI. COMPARISONS AND FUTURE WORK

A Service Based System (SBS) is composed of services connected with structured logic (AND, XOR, SEQ, and LOOP). Because each service is selected based on their abstract, the performance is one necessary factor and both minimum and maximum execution times are two major data. A workflow is usually treated as a fundamental technique adopted to construct an SBS. Thus, an SBS can be modeled as a temporal structured (TS) workflow intuitively. Conventional detection techniques were developed to detect two continuous activities of an artifact during run time. Our paper presents a set of new definitions for anomalous behavior to simplify the corresponding detection.

By comparing conventional techniques and ours, there are at least three contributions introduced here:

1. Conventional definitions do not discuss and detect the anomalies occurring in two concurrent processes. These anomalies might occur repeatedly in different execution paths and make the detection and thus discussion more complicated. In this paper, we define these anomalies distinctly and thus the detection can be done easier.
2. Our definition set of continuous artifact anomalies is a subset of conventional ones. Thus, the detection in our approach is much easier. The interpretation of anomalies detected by our algorithms is simpler, since the effect of concurrency has been deleted.
3. In the past, the temporal data were applied to analyze the control inside timed Petri nets for distributed systems. There is no effective study on artifact anomaly detection based on a TS workflow, applied to model an SBS intuitively.

However, the algorithms in the paper are not well concerned with complexity or effectiveness. The useful interpretations of anomalies detected are not studied either. Furthermore, the loop logic, which has been studied to be transferred into a pattern of XOR branch, might be studied to improve the detection for continuous anomalies. These problems are being studied and planed in our future work.

REFERENCES

[1] N. R. Adam, V. Atluri, and W.-K. Huang, 1998, Modeling and Analysis of Worfklows Using Petri Nets, Journal of Intelligent Information Systems, Vol. 10, Issue 2, pp. 131-158.

[2] W. M. P. van der Aalst and A. H. M. ter Hofstede, 2000, Verification of Workflow Task Structures: A Petri-net Approach, Information System, Vol. 25, Issue 1, pp. 43-69.

[3] W. M. P. van der Aalst, K.M. van Hee and R.A. van der Toorn, 1999, Adaptive Workflow: An Approach Based on Inheritance, the Proceedings of the Workshop on Intelligent Workflow and Process Management, The New Frontier for AI in Business, pp. 36-45.

[4] B. Kiepuszewski, A.H.M. ter Hofstede, and C. Bussler, 2000, On Structured Workflow Modelling, Lecture Notes in Computer Science, Vol. 1789, pp. 431-445.

[5] J. Eder, E. Panagos, H. Pozewaunig, and M. Rabinovich, 1999, Time Management in Workflow Systems, the Proceedings of International Conference on Business Information Systems, pp. 266-280.

[6] J. Eder, E. Panagos, and M. Rabinovich, 1999, Time Constraints in Workflow Systems, Lecture Notes in Computer Science, Vol. 1626, pp. 286-300

[7] O. Marjanovic, 2000, Dynamic Verification of Temporal Constraints in Production Workflows, the Proceedings of the 11th Australian Database Conference, pp. 74-81.

[8] J. Li, Y. Fan, and M. Zhou, 2004, Performance Modeling and Analysis of Workflow, IEEE Transaction on Systems, Man, and Cybernetics - Part A: Systems and Humans, Vol. 34, Issue 2, pp.229-242.

[9] Chen, and Y. Yang, 2008, Temporal Dependency based Checkpoint Selection for Dynamic Verification of Fixed-time Constraints in Grid Workflow Systems, the Proceedings of the 30th International Conference on Software Engineering, pp. 141-150.

[10] S. Sadiq, M. E. Orlowska, W. Sadiq, and C. Foulger, 2004, Data flow and validation in workflow modeling, the Proceedings of the 15th Conference on Australasian Database, Vol. 27, pp. 207-214.

[11] F.J. Wang, C.L. Hsu and H.-J. Hsu, 2006, Analyzing Inaccurate Artifact Usages in a Workflow Schema, the Proceedings of the 30th Annual International Computer Software and Application Conference, Vol. 2, pp. 109-114.

[12] C.L. Hsu, H.J. Hsu and F.J. Wang, 2007, Analysing Inaccurate Artifact Usages in Workflow Specifications, IET Software, Vol. 1, Issue 4, pp. 188-205.

[13] C.-H. Wang and F.J. Wang, 2009, Detecting Artifact Anomalies in Business Process Specification with a Formal Model, Journal of Systems and Software, Vol. 82, Issue 10, pp. 1064-1212.

[14] H.J. Hsu, and F.J. Wang, 2009, Using Artifact Flow Diagrams to Model Artifact Usage Anomalies, the Proceedings of 33rd Annual IEEE International Computer Software and Applications Conference, Vol. 2, pp.275-280.

[15] H. Li, and Y. Yang, 2005, Dynamic Checking of Temporal Constraints for Concurrent Workflows, Electronic Commerce Research and Applications Vol. 4, pp. 124-142.

[16] Workflow Management Coalition (WfMC), 1999, WFMC-TC-1011 Ver3 Terminology and Glossary English, Workflow Management Coalition.

[17] H. Zhuge, T.Y. Cheung, and H.K. Pung, 2001, A Timed Workflow Process Model, Journal of Systems and Software, Vol. 55, Issue 2, pp. 231-243.

[18] I.F. Leong and Y.W. Si, 2009, Temporal Exception Prediction for Loops in Resource Constrained Concurrent Workflows, the Proceedings of 6th IEEE International Conference on e-Business Engineering, pp. 310-315.

[19] J. F. Allen, 1983, Maintaining knowledge about temporal intervals, Communication of the ACM, Vol. 26, Issue 11, pp.832–843.