Chapter 11

Predicting Temporal Exceptions in Concurrent Workflows

Iok-Fai Leong

University of Macau, Macau

Yain-Whar Si

University of Macau, Macau

Robert P. Biuk-Aghai

University of Macau, Macau

ABSTRACT

Current Workflow Management Systems (WfMS) are capable of managing simultaneous workflows designed to support different business processes of an organization. These departmental workflows are considered to be interrelated since they are often executed concurrently and are required to share a limited number of resources. However, unexpected events from the business environment and lack of proper resources can cause delays in activities. Deadline violations caused by such delays are called temporal exceptions. Predicting temporal exceptions in concurrent workflows is a complex problem since any delay in a task can cause a ripple effect on the remaining tasks from the parent workflow as well as from the other interrelated workflows. In addition, different types of loops are often embedded in the workflows for representing iterative activities, and presence of such control flow patterns in workflows can further increase the difficulty in estimation of task completion time. In this chapter, the authors describe a critical path based approach for predicting temporal exceptions in concurrent workflows that are required to share limited resources. This approach allows predicting temporal exceptions in multiple attempts while workflows are being executed. The accuracy of the proposed prediction algorithm is analyzed based on a number of simulation scenarios. The result shows that the proposed algorithm is effective in predicting exceptions for instances where long duration tasks are scheduled (or executed) at the early phase of the workflow.

INTRODUCTION

In a highly dynamic business environment, simultaneously executing departmental workflows form digital business ecosystems (Boley & Chang, 2007) which are designed to share limited resources. In this chapter we address the temporal exception prediction problems of a digital business ecosystem involving concurrent workflows.

A workflow specification defines how a business process functions within an organization (Son, Kim, & Kim, 2005). Based on these specifications, Workflow Management Systems (WfMS) allocate and dispatch work to users (Li & Yang, 2005). A workflow instance is an execution of a workflow specification. During the execution of a workflow instance, some events that are not defined in the workflow specification may occur. These events are typically considered as exceptions in workflow management systems.

In the area of programming languages, exceptions may interrupt or abort the execution of a program. Exceptions in WfMS are also similar to those of programming languages. Two types of exceptions (Casati, 1998) can be defined in WfMS; expected exceptions which are the results of predictable deviations from the normal behavior of a process and unexpected exceptions which are the outcomes of inconsistencies between the business process in the real world and its corresponding workflow description.

A workflow specification may consist of several workflows that execute concurrently and share the same pool of resources. In large organizations, a number of different workflows could be executed simultaneously. These concurrent workflows are often interdependent since they are required to share limited resources. For example, two concurrent workflows designed, respectively, for an inventory management process and a delivery process may share the same human resources (e.g. a group of technicians) who are responsible for managing a storage facility and a

fleet of vehicles. Delays occur when only a limited number of resources are available during a given interval. We denote deadline violations caused by such delays as temporal exceptions.

In workflow management systems, control-flow patterns are used to describe the order of tasks that make up a process and the relationship between them (Van der Aalst, Ter Hofstede, Kiepuszewski, & Barros, 2003). In this chapter, we focus on predicting temporal exceptions for workflows which include iteration patterns (loops). Iteration pattern refers to a repeated execution of one or more tasks within a workflow. Although iteration patterns are extensively used in workflow specifications, less attention has been devoted to understanding their implications for temporal exceptions. Specifically, rapid changes in the number of iterations can cause deadline violations as well as conflicts in resource usage.

Our approach can be divided into two phases: *design time* phase and *run time* phase. During the design time, temporal and resource constraints are calculated for each task within the workflow specifications. During the run time, an algorithm is used to predict potential deadline violations of workflow instances by taking into account constraints calculated at design time.

The remainder of this chapter is organized as follows. In the following section we give a brief introduction to resource and temporal constraints in a workflow. Then we describe the critical path based method for exception prediction. Next we illustrate an example of temporal exception prediction, and evaluate the proposed method based on a simulation experiment. We then review recent related work and finally summarize our ideas.

OVERVIEW

A workflow management system can be used to host several simultaneously executing workflows. Conflicts usually occur when tasks from

instances of different workflows compete for limited resources (e.g. human resources, financial resources, etc).

Figure 1 (a) shows an example of conflict involving tasks from two concurrent workflows when only four units of resources are available for sharing. In this example, resource conflicts occur at tasks T_{22} , T_{23} , T_{24} and T_{25} . Furthermore, these resource conflicts are linked to the underlying temporal constraints of a task within the workflow.

During the design time, each task T within a workflow can be specified with a value $D(T)$ (see Figure 1 (b)) which is the maximum allowable execution time of task T . Based on $D(T)$, we can derive two temporal constraints: start time $ST(T)$ and end time $ET(T)$.

These constraints are crucial in determining the deadline of a workflow. Basically, temporal constraints of a task are related to its predecessor and successor tasks. For instance, if a task is scheduled to execute immediately after another task, its start time will be the end time of its predecessor task. However, depending on the control flows (Van der Aalst et al., 2003) within the workflow, derivation of these constraints can be different.

A workflow may also have loops. A loop is a repeated execution of one or more tasks within a workflow. Such repetition has a significant impact

on temporal constraints of a task which may belong to one or more loops. Since tasks within a loop can be repeated a number of times, calculating temporal constraints of a task also needs to take into account possible repetition of its ancestors and descendants as well. Such calculation can be even more problematic when a task belongs to one or more nested loops. Based on the definition from (Van der Aalst et al., 2003), loops can be classified as structured loops and arbitrary loops. In this chapter, we further divide arbitrary loops into nested loops and crossing loops. In a nested loop, all tasks within a loop appear in another loop. In a crossing loop, only a certain number of tasks within a loop appear in another loop.

Let I_N be the maximum number of iterations that a loop N within a workflow wf can be executed, and let S_N be the set of tasks in loop N in wf . We assume that $I_N > 0$. We also assume that each task within a workflow is atomic, i.e., a task must be executed entirely until it is completed and cannot be interleaved with other tasks.

Structured Loop. The simplest form of a loop is the Structured Loop (Van der Aalst et al., 2003). An example of a structured loop is shown in Figure 2.

Figure 1. (a) Resource conflict in workflows; (b) Temporal constraints of task T

Figure 2. Structured loop

Figure 3. Nested loop

Definition 1: Structured Loop. A loop N in a workflow wf is a Structured Loop if for all tasks T in S_N , T does not exist in any other loop in wf .

Nested Loop. In a nested loop, all tasks within a loop are also members of another loop. Such loops are also classified as Arbitrary Loops (Van der Aalst et al., 2003). For instance, in Figure 3, a loop N is nested within another loop M .

Definition 2: Nested loop: A loop M in a workflow wf is a nested loop if there exists another loop N such that all the tasks in N are also members of M and both M and N do not share the same initial task. If M and N share the same initial task, we consider M as a crossing loop.

Crossing loop. Crossing loops are also classified as Arbitrary Loops (Van der Aalst et al., 2003). In Figure 4, an example of a crossing loop is depicted with two loops N and M crossing each other at T_a and T_k .

Definition 3: Crossing loop. Loop N and M in a workflow wf are crossing loops if (1) both N and M share at least one task and (2) in each of N and M there exists at least one task of wf which is a member of only one but not both of the loops N and M . Both loops may begin at the same task, i.e. both flows from T_k and T_b may point back to either T_i or T_a in Figure 4. In this case, the loop whose last task ends earlier is considered as loop N .

In this chapter we define workflow specifications in XML Schema based on YAWL (Van der Aalst & Ter Hofstede, 2005). A sample XML schema is shown in Figure 5. The XML code defines the duration of each of the workflow specifications and the resource usage for each of them. It also defines the loops within the workflow specifications.

CRITICAL PATHS IN WORKFLOWS

In Figure 6, we describe the calculation of the execution time for the control flow patterns defined in Li and Yang (2005). The resulting execution

Figure 4. Crossing loop

time is then used for calculating the critical path of the workflow. A critical path in a workflow specification can be defined as a series of tasks from the first task to the last with the longest execution time. Critical paths can be used to determine the hard deadline of the workflow after resolving any conflict in resource usage. In this research, we apply a breadth-first-search algorithm to identify all

the possible paths of a workflow specification. We assume that the workflow is well-formed and free from structural errors. The breadth-first-search algorithm recursively traverses the workflow and generates a list of all possible paths. The total execution time of a path can be calculated from the sum of the execution times of each task within the path. In case there are loops defined in

Figure 5. Workflow schema defined using XML schema

```

<?xml version="1.0" encoding="UTF-8"?>
<workflowSchema>
  <resource size="4"/>
  <workflow name="wf1">
    <task name="T11" duration="5"
    resource="2">
      <flowsInto task="T12"/>
      <join type="or"/>
      <split type="and"/>
    </task>
    ...
    <lastTask name="T16"/>
    <firstTask name="T11"/>
    <loop name="wf1_n"
    maxExecution="3">
      <member name="T11"/>
      <member name="T12"/>
      <member name="T13"
      loopTo="T11"/>
    </loop>
  </workflow>
  <workflow name="wf2">
    ...
  </workflow>
</workflowSchema>

```

Figure 6. Execution time for different control patterns (Adapted from Li and Yang (2005))

the workflow, the algorithm traverses the loops in all possible numbers of iterations and treats each of them as a different path. The algorithm for calculating all possible paths within a workflow is described in Algorithm 1 below.

The path(s) with the longest execution time are then selected as the critical path(s) of the workflow. In addition, the shortest path(s) in the workflow are also recorded. Later in this chapter, we will make use of the longest and shortest paths for calculating the Hard Deadline (*HD*) and Earliest Completion Time (*ECT*) of the workflow.

PREDICTING TEMPORAL EXCEPTIONS

The algorithm for predicting temporal exceptions is divided into two phases (see Figure 7). The first phase can be performed during the design time and second phase is performed during run time. These phases are shown in the upper and lower areas of Figure 7, respectively.

Design Time: During the design time, each workflow specification is defined with corresponding temporal and resource allocation pa-

Algorithm 1. Generation of all possible paths within a workflow

Input: Workflow specification <i>wf</i> Output: List of all possible paths for <i>wf</i>
1. Initialize <i>pathlist</i> as an empty list of task lists 2. Initialize <i>path</i> as an empty list of tasks 3. Start with the first task <i>T</i> of <i>wf</i> 4. Call Algorithm 1-(a) with <i>T</i> , <i>path</i> , <i>pathlist</i> as parameters
Algorithm 1-(a) – Recursive procedure that adds a task to a path
Input: Task <i>T</i> , List of tasks <i>path</i> , list of paths <i>pathlist</i> Output: Updated list of paths <i>pathlist</i>
1. Add <i>T</i> to <i>path</i> 2. If <i>T</i> has an outgoing flow to another task (i.e., <i>T</i> is not the end task of the workflow) a) For each next task <i>T'</i> from <i>T</i> , call Algorithm 1-(a) with <i>T'</i> , <i>path</i> , <i>pathlist</i> as parameters Else if <i>path</i> is not in <i>pathlist</i> , add <i>path</i> to <i>pathlist</i>

Figure 7. Steps for predicting temporal exceptions

rameters. Specifically, every task in the workflows is configured with parameters such as maximum allowable execution time and the number of required resources. These parameters should be defined to reflect the actual constraints of the business process. Next, Algorithm 1 is used to

derive all the possible paths from these workflow specifications. The generated paths are then stored in a list for further processing.

Run Time: Exception prediction is performed during run time (i.e. during workflow execution). Suppose that workflow specifications defined in the previous phase are deployed in a workflow management system (or in a workflow engine) for execution. During the deployment, a number of workflow instances could be generated by the system. These workflow instances may execute in parallel and any resources defined within the workflows could be shared by the tasks from these instances. For the purpose of simplification, we define our algorithms for two concurrent workflow instances. These algorithms can be extended or generalized for cases including more than two workflow instances.

Let p be the current time point during the execution. To predict exception at time p , all paths generated from the previous phase and the detailed information of the ongoing workflow instances at time p are input into four algorithms (algorithms II, III, IV, and V).

First, Algorithm 3 is used to estimate the end times of two ongoing concurrent workflow instances for worst case scenarios. Such estimation can be done by calculating/projecting the longest possible execution path of each workflow instance beyond current time point p . First, we identify a longest path from the path lists which contain the identical sequence of tasks from the paths of each ongoing workflow instance. Note that the path lists are generated during design time based on Algorithm 1. The identified two longest paths can be considered as the sequence of tasks which can be executed during the worst case scenario.

However, when the two ongoing workflow instances progress (according to the identified longest paths in the worst case scenarios), these instances may need to share limited resources and therefore any conflicts resulting from the tasks within these workflow instances need

Algorithm 2. Generation of Execution Trace from two workflow instances

Input: Workflow instances $wi1, wi2$ Output: Conflict free Execution Trace
1. For each occurrence of task $t_{(i)}$ in the workflow instance, where i represents the i -th occurrence of task t in the workflow instance, check if any other task t' has overlapping executing time and resource conflict. If it does, put t' into a list of tasks called $dep(t_{(i)})$. For each task $t_{(i)}$, each task in $dep(t_{(i)})$ is said to have a "resource conflict" with $t_{(i)}$. 2. For each task t' in $dep(t_{(i)})$, modify $[ST(t'), ET(t')]$ as $[MAX\{ST(t'), ST(t_{(i)})+D(t_{(i)})\}, MAX\{ET(t'), ET(t_{(i)})+D(t_{(i)})\}]$. Modify the ST/ET values of tasks t'' where all tasks t'' are reachable by t' accordingly.

to be resolved. In Algorithm 2, we resolve the resource conflicts between any two workflow instances by interleaving and ordering of tasks from each instance. The result of Algorithm 2 is called an *Execution Trace* which represents the conflict-free ordering of tasks from two workflow instances. By using Algorithm 2, we can resolve any conflicts among two longest paths identified from the previous step. The end time of the resulting execution trace is called the *Hard Deadline* (HD- shown in algorithm 3) which is the latest time by which both ongoing workflow instances must complete their execution without causing an exception. HD can be also considered as the outermost boundary of the acceptable time limit for both workflow instances.

Next, Algorithm 4 is used to estimate the end times of two ongoing concurrent workflow instances for best case scenarios. In contrast to Algorithm 3, such estimation can be done by calculating/projecting the shortest possible execution path of each workflow instance beyond cur-

rent time point p . First, we identify a shortest path from the path lists which contain the identical sequence of tasks from the paths of each ongoing workflow instance. The path lists are generated during design time based on Algorithm 1. The identified two shortest paths represent the sequence of tasks which can be executed during the best case scenario.

However, when the two ongoing workflow instances progress according to the identified shortest paths in the best case scenario, these instances may need to share limited resources and therefore any conflicts resulting from the tasks within these workflow instances need to be resolved. By using Algorithm 2, we can resolve any conflicts among two shortest paths identified from the previous step. The end time of the resulting execution trace is called the *Earliest Completion Time (ECT)* which is the earliest time by which both ongoing workflow instances can complete their execution.

Algorithm 3. Calculation of hard deadline (HD)

Input: Workflow specifications $wf1, wf2$; Workflow instances $wi1, wi2$; time point p Output: Hard Deadline (HD) for the workflow instances
1. For workflow specification $wf1$, call Algorithm 1 to generate the list of all possible paths for $wf1$. From the list, identify the longest path $wi1'$ that contains the same sequence of tasks from $wi1$. 2. Set maximum allowable duration for each task in $wi1'$ based on workflow specification $wf1$. 3. For workflow specification $wf2$, call Algorithm 1 to generate the list of all possible paths for $wf2$. From the list, identify the longest path $wi2'$ that contains the same sequence of tasks from $wi2$. 4. Set maximum allowable duration for each task in $wi2'$ based on workflow specification $wf2$. 5. Call Algorithm 2 with $wi1', wi2'$ and time point p as input to generate the execution trace. 6. The end time of the execution trace is the Hard Deadline (HD) of the workflow. 7. If there is more than one longest path (with the same path cost) from either $wf1$ or $wf2$, repeat steps 1 to 6 for each longest path to calculate HD and choose the largest one as the final result.

Algorithm 4. Calculation of earliest completion time (ECT)

Input: Workflow specifications $wf1, wf2$; Workflow instances $wi1, wi2$; time point p
Output: Earliest Completion Time (ECT) of the workflow instances
1. For workflow specification $wf1$, call Algorithm 1 to generate the list of all possible paths for $wf1$. From the list, identify the shortest path $wi1'$ that contains the same sequence of tasks from $wi1$.   2. Update the duration of tasks from $wi1'$ with the actual execution time (actual duration is used to reflect the real execution progress of the workflow instance) recorded in $wi1$. In this step, only tasks before and up to time point p will be updated.   3. Set maximum allowable duration for all remaining unexecuted tasks in $wi1'$ based on workflow specification $wf1$.   4. For workflow specification $wf2$, call Algorithm 1 to generate the list of all possible paths for $wf2$. From the list, identify the shortest path $wi2'$ that contains the same sequence of tasks from $wi2$.   5. Update the duration of tasks from $wi2'$ with the actual execution time recorded in $wi2$. In this step, only tasks before and up to time point p will be updated.   6. Set maximum allowable duration for all remaining unexecuted tasks in $wi2'$ based on workflow specification $wf2$.   7. Call Algorithm 2 with $wi1', wi2'$ and time point p as input to generate the execution trace for the two workflow instances.   8. The end time of the resulting execution trace is the Earliest Completion Time (ECT) of the workflow.   9. If there is more than one shortest path (with the same path cost) from either $wf1$ or $wf2$, repeat steps 1 to 8 for each shortest path to calculate ECT and choose the smallest one as the final result.

Algorithm 5. Predicting exception

Input: Workflow specifications $wf1, wf2$; workflow instances $wi1, wi2$; time point p
Output: Either exception is predicted or cannot be determined
1. Call Algorithm 3 with $wf1, wf2, wi1, wi2$, and p as input to calculate HD.   2. Call Algorithm 4 with $wf1, wf2, wi1, wi2$, and p as input to calculate ECT.   3. If $ECT > HD$, then ongoing workflow instances are predicted to have an exception. Otherwise an exception cannot be determined.

Based on Algorithms 3 and 4, we can derive the Earliest Completion Time (ECT) from the shortest paths and Hard Deadline (HD) from the longest paths. HD is the latest time by which both workflow instances must complete their execution while ECT is the earliest possible time for completion. By comparing ECT and HD we can predict whether there will be an exception or not in the future. Algorithm 5 shows the steps for exception prediction.

EXAMPLE

In this section we outline a sample exception prediction for two workflow instances from specifications $wf1$ and $wf2$ (see Figure 8 and Table 1, respectively).

In $wf1$, loops m and n form a crossing loop. In $wf2$ loops m and n form a nested loop.

For the purpose of illustration, in this example, we assume that each loop can iterate for at most one time. Suppose that $wf2$ starts later than $wf1$ and both workflows are required to share 4 units of identical resources. The durations and resource usages for each of the tasks in both workflows are specified during the design time and are shown in Table 1. $D(T)$ represents the maximum allowable execution time and $R(T)$ represents the required number of resources for a task T .

First, we apply Algorithm 1 to find all the possible paths for the two workflow specifications (see Figure 9). Figures 10 (a) and (b) show the list of all possible paths generated for $wf1$ and $wf2$, respectively.

Suppose two workflow instances $wi1$ and $wi2$ are executing at time point p (see Figure 11) based on workflow specifications $wf1$ and $wf2$. Our aim is to perform Algorithm 5 at different time points to predict potential exceptions for both workflow instances (Figures 11 (a) and (b)). In this example,

two prediction attempts are made at $p = 20$ and $p = 40$.

At time point = 20, $wf1$ has completed $T11 \rightarrow T12 \rightarrow T14 \rightarrow T15 \rightarrow T16$ (the last one was partially executed), and $wf2$ has executed $T21 \rightarrow T22 \rightarrow T25 \rightarrow T26 \rightarrow T27$. First we apply Algorithm 3 to find their longest paths. The longest path for $wf1$ that starts with $T11 \rightarrow T12 \rightarrow T14 \rightarrow T15 \rightarrow T16$ is $T11 \rightarrow T12 \rightarrow T14 \rightarrow T15 \rightarrow T16 \rightarrow T12 \rightarrow T13 \rightarrow T11 \rightarrow T12 \rightarrow T13 \rightarrow T15 \rightarrow T16$; whereas the longest path of $wf2$ that starts with $T21 \rightarrow T22 \rightarrow T25 \rightarrow T26 \rightarrow T27$ is $T21 \rightarrow T22 \rightarrow T25 \rightarrow T26 \rightarrow T27 \rightarrow T28 \rightarrow T21 \rightarrow T22 \rightarrow T23 \rightarrow T24 \rightarrow T23 \rightarrow T24 \rightarrow T27 \rightarrow T28$ (see Figure 12 (a)). These two longest paths are fed into Algorithm 3 and a Hard Deadline HD (65)

is calculated from the resulting conflict-free execution trace.

Next we apply Algorithm 4 to find their shortest paths. For $wf1$, the shortest path that starts with $T11 \rightarrow T12 \rightarrow T14 \rightarrow T15 \rightarrow T16$ is $T11 \rightarrow T12 \rightarrow T14 \rightarrow T15 \rightarrow T16$. For $wf2$, the shortest path that starts with $T21 \rightarrow T22 \rightarrow T25 \rightarrow T26 \rightarrow T27$ is $T21 \rightarrow T22 \rightarrow T25 \rightarrow T26 \rightarrow T27 \rightarrow T28$ (see Figure 12 (b)). Note that $T16$ has not finished execution at $p = 20$. These two shortest paths are fed into Algorithm 4 and the Earliest Completion Time $ECT(27)$ is calculated from the resulting conflict-free execution trace. Since $ECT < HD$, at time point $p = 20$, we cannot determine whether an exception will occur in the future.

Figure 8. Workflow specifications $wf1$ and $wf2$

Table 1. Durations and resource usage for the tasks in both workflows

Task T	Duration $D(T)$	Resource Usage $R(T)$
T11	5	2
T12	2	1
T13	5	4
T14	3	2
T15	1	2
T16	5	1
T21	2	1
T22	3	2
T23	5	3
T24	4	3
T25	2	1
T26	2	1
T27	1	1
T28	6	4

Since exception can not be predicted at $p = 20$, we make another attempt at prediction at $p = 40$ (see Figure 11 (b)). When $p = 40$, $wf1$ has executed $T11 \rightarrow T12 \rightarrow T14 \rightarrow T15 \rightarrow T16 \rightarrow T12 \rightarrow T14 \rightarrow T15 \rightarrow T16$ (the last one was partially executed), and $wf2$ has executed $T21 \rightarrow T22 \rightarrow T25 \rightarrow T26 \rightarrow T27 \rightarrow T28 \rightarrow T21 \rightarrow T22 \rightarrow T23$ (the last one was also partially executed). First we apply Algorithm 3 to find their longest paths. The longest path for $wf1$ that starts with $T11 \rightarrow T12 \rightarrow T14 \rightarrow T15 \rightarrow T16 \rightarrow T12 \rightarrow T14 \rightarrow T15 \rightarrow T16$ is $T11 \rightarrow T12 \rightarrow T14 \rightarrow T15 \rightarrow T16 \rightarrow T12 \rightarrow T14 \rightarrow T15 \rightarrow T16$; whereas the longest path of $wf2$ that starts with $T21 \rightarrow T22 \rightarrow T25 \rightarrow T26 \rightarrow T27 \rightarrow T28 \rightarrow T21 \rightarrow T22 \rightarrow T23$ is $T21 \rightarrow T22 \rightarrow T25 \rightarrow T26 \rightarrow T27 \rightarrow T28 \rightarrow T21 \rightarrow T22 \rightarrow T23 \rightarrow T24 \rightarrow T23 \rightarrow T24 \rightarrow T27 \rightarrow T28$ (see Figure 12 (c)). Notice that the two longest paths are different from those at $p = 20$ since both workflow instances have progressed up to time point 40. These two longest paths are fed into Algorithm 3 and a Hard Deadline HD

(53) is calculated from the resulting conflict-free execution trace.

Next we apply Algorithm 4 to find their shortest paths. For $wf1$, the shortest path that starts with $T11 \rightarrow T12 \rightarrow T14 \rightarrow T15 \rightarrow T16 \rightarrow T12 \rightarrow T14 \rightarrow T15 \rightarrow T16$ is $T11 \rightarrow T12 \rightarrow T14 \rightarrow T15 \rightarrow T16 \rightarrow T12 \rightarrow T14 \rightarrow T15 \rightarrow T16$. For $wf2$, the shortest path that starts with $T21 \rightarrow T22 \rightarrow T25 \rightarrow T26 \rightarrow T27 \rightarrow T28 \rightarrow T21 \rightarrow T22 \rightarrow T23$ is $T21 \rightarrow T22 \rightarrow T25 \rightarrow T26 \rightarrow T27 \rightarrow T28 \rightarrow T21 \rightarrow T22 \rightarrow T23 \rightarrow T24 \rightarrow T27 \rightarrow T28$ (see Figure 12 (d)). These two shortest paths are fed into Algorithm 4 and the Earliest Completion Time ECT (55) is calculated from the resulting conflict-free execution trace. Since $ECT > HD$ we can predict that an exception may occur in the future. In the next section we will perform an experiment to show how different prediction time points can affect the prediction accuracy.

EXPERIMENT

To validate our algorithm we perform a simulation experiment by generating a set of workflow instances using Algorithm 6. The workflow instances are generated based on randomized execution times for each task.

Algorithm 7 shows how we validate our prediction result. In this algorithm, we check if an instance will have an actual exception or not, and compare it with the predicted result. If they match, we say that the prediction is correct.

Random Task Duration

In our experiment, we use Algorithm 6 to generate 6000 workflow instances (3000 for each workflow specification $wf1$ and $wf2$). Exception prediction is then performed for each pair of workflow instances at every 20 time units (except when $p=0$ and $p \geq$ end time of these instances). Figure 13 shows the accuracy of the prediction algorithm after running Algorithm 7. The x-axis shows dif-

Figure 9. Applying algorithm 1 to find all possible paths

ferent time points (p) that we have chosen, and the y-axis indicates the accuracy of the prediction result. Since the latest end time of the generated instances is 138, we perform prediction at $p = 20, 40, 60, 80, 100$ and 120 respectively. The result shows that the accuracy of the proposed method reaches approximately 60% when $p = 60$. We can observe that the prediction accuracy increases with the time point. This makes sense since the longer the workflow instance executes, the more information we can get from the instance for predicting exceptions.

In addition, we further conduct six simulation experiments to analyze the influence of execution time of tasks on the prediction accuracy. In the

following six scenarios we use various randomization procedures to generate the execution time of the tasks.

1. Early “long-task” scenario
2. Late “long-task” scenario
3. High resource usage “long-task” scenario
4. Low resource usage “long-task” scenario
5. Long duration “long-task” scenario
6. Short duration “long-task” scenario

For each scenario, we generate 6000 instances (3000 instances for each workflow specification $w/1$ and $w/2$) and perform exception prediction at various time points for each pair of workflow

Figure 10. (a) List of all possible paths for wf1; (b) List of all possible paths for wf2

Figure 11. (a) Workflow instance at time point = 20; (b) Workflow instance at time point = 40

Algorithm 6. Generating workflow instances for simulation

Input: Workflow specification $wf1$, $wf2$
Output: Workflow instances $wi1$ and $wi2$ with random durations for tasks
1. Extract temporal and resource parameters from workflow specifications $wf1$ and $wf2$.   2. Create workflow instances $wi1$ and $wi2$ corresponding to $wf1$ and $wf2$. Set starting time of $wi1$ to 0, and randomize the starting time of $wi2$.   3. Continue the execution of $wi1$ and $wi2$ according to the control flows defined in the respective workflow specifications. During the execution, interleave and resolve any conflicts for tasks in $wi1$ and $wi2$ whenever necessary. Randomize the execution time of each task in $wi1$ and $wi2$.   4. For any loop within the workflow, a random number (which is less than the maximum number of allowable iterations) is chosen as the number of iterations.

Algorithm 7. Validating the accuracy of prediction

Input: Workflow specification $wf1$, $wf2$; Workflow instances $wi1$, $wi2$; time point p
Output: Validity of the predicted result compared with the actual result
1. For Workflow instances $wi1$, $wi2$, pick a time point p and calculate the Hard Deadline (HD) using Algorithm 3.   2. Retrieve the actual end time $aet-wi1$ and $aet-wi2$ of $wi1$, $wi2$ from the audit trail.   3. Set Actual End Time (AET) = $\max(aet-wi1, aet-wi2)$.   4. If $AET > HD$, then this workflow instance has an exception.   5. Perform the exception prediction using Algorithm 5, and record whether this workflow instance is predicted to have an exception or not.   6. Compare results from steps (4) and (5). If the actual result is the same as the predicted result, then we say the prediction is correct.

instances. Table 2 shows the statistics about these scenarios. The total number of prediction points (attempts) varies from one scenario to another since some simulated workflow instances have shorter end times.

In the following sections, we summarize the experiment results from these scenarios.

Early “Long-Task” Scenario

In this scenario, certain tasks are randomly selected to have longer execution times (compared to the

predefined maximum allowable duration). Selection is done for those tasks which are executed before $p=30$. Figure 14 shows the accuracy of the proposed prediction algorithm. The result shows that nearly 50% accuracy is achieved at an early stage ($p=40$) compared to the result from Figure 13. It also shows that the performance of the proposed algorithm increases when tasks exceed their maximum allowable duration in the early phase of the workflow execution.

Figure 12. (a) Longest paths when $p = 20$, (b) Shortest paths when $p = 20$; (c) Longest paths when $p = 40$, (d) Shortest paths when $p = 40$

Figure 13. Prediction accuracy for workflow instances with randomized task duration

Table 2. Statistics on different scenarios

Scenario	No. of instances	No. of prediction points (attempts)	No. of actual exceptions	Max. end time of workflow instances
1	6000	5914	640	102
2	6000	5492	552	100
3	6000	7233	2542	137
4	6000	6815	1314	121
5	6000	7892	3138	148
6	6000	5977	801	97

Late “Long-Task” Scenario

In this scenario, certain tasks are randomly selected to have longer execution times (compared to the predefined maximum allowable duration) at the later phase of the workflow. Selection is done for those tasks which are executed after $p=50$. Figure 15 shows the accuracy of the prediction algorithm.

The result reveals the lower accuracy in prediction compared to the previous scenario. The maximum accuracy only reaches 60% when $p=80$. The low accuracy result is caused by the late scheduling of tasks which have longer execution time in the simulated workflow instances. Due to the delayed execution of “long-task” at later

stages of the workflow, the prediction algorithm has less information about the future execution situation. As a result, the accuracy of prediction is affected. It is also interesting to see that 20% accuracy in prediction is achieved at $p=40$ even though “long-tasks” are introduced at a later stage of the workflow (when $p>50$).

High Resource Usage “Long-Task” Scenario

In this scenario, certain tasks with resource usage of more than 2 units are randomly selected and assigned with longer execution times (compared to the predefined maximum allowable duration).

Figure 14. Prediction accuracy on early “long-task” scenario

Figure 15. Prediction accuracy on late “long-task” scenario

Figure 16 shows the accuracy of the prediction algorithm. In concurrent workflows, tasks with a high resource demand are more likely to affect other competing tasks since any delay in these tasks can severely reduce the level of available shared resources in the workflow for an extended

period. Such a situation can significantly increase the chance of resulting in an exception.

Our simulation result also confirms that a high number of actual exceptions is generated in this scenario (see Table 2). However, the prediction accuracy is similar to the experiment result from

Figure 16. Prediction accuracy on high resource usage “long-task” scenario

Figure 13. It shows that an increase in the number of actual exceptions does not affect the accuracy of the prediction algorithm.

Low Resource Usage “Long-Task” Scenario

In this scenario, certain tasks with resource usage less than 3 units are randomly selected and assigned with longer execution times (compared to the predefined maximum allowable duration). In contrast to the previous scenario, Figure 17 shows the increase in the accuracy of the prediction algorithm. Our simulation result also shows that a low number of actual exceptions is generated in this scenario (see Table 2). This is in line with our expectation since any delays in tasks with low resource usage have a lower chance of generating an exception.

Long Duration “Long-Task” Scenario

In this scenario, tasks with predefined execution time greater than 3 units are randomly selected and assigned with longer execution times (compared

to the predefined maximum allowable duration). Figure 18 shows the accuracy of the prediction algorithm. The result is similar to that of the random task duration experiment (see Figure 13).

Short Duration “Long-Task” Scenario

In this scenario, during the simulation, tasks with execution time less than 4 units are randomly selected and assigned with longer execution times (compared to the predefined maximum allowable duration). Figure 19 shows the accuracy of the prediction algorithm. In this scenario, short tasks are favored for creating a delay and as a result, a significantly lower number of actual exceptions is generated (see Table 2). The maximum end time of the workflow instances is also shorter compared to other scenarios. The result also shows that the prediction accuracy is similar to that of the low resource usage “long-task” scenario.

In summary, our experiment results show that the proposed algorithm is able to predict more accurately at $p=80$ (at slightly over half-way to completion). The experiment result also reveals

Figure 17. Prediction accuracy on low resource usage “long-task” scenario

Figure 18. Prediction accuracy on long duration “long-task” scenario

that the execution of long duration tasks and their positions within the workflow instances have a significant impact on the accuracy of the prediction algorithm. The result shows that the proposed algorithm is more effective in predicting excep-

tions for instances where long duration tasks are scheduled (or executed) at the early phase of the workflow.

Figure 19. Prediction accuracy on short duration “long-task” scenario

RELATED WORK

Critical paths are widely used in predicting as well as in handling exceptions. For detecting artifact anomalies in temporal structured workflows, Hsu and Wang (2011) unfold loops into a decision structure with three branches. The first branch represents the case when there is no iteration. The second branch represents the case when there is a single iteration. The last branch represents the case for maximal iterations. The resulting decision structure is then used for the analysis of structure and temporal relationships between artifact operations to reveal any anomalies buried in the workflow. In the algorithm for handling temporal exception (Xie, Yu, & Kuang, 2009), the longest and shortest time to complete is used to detect potential exceptions at the process instantiation or during run time. When a potential exception is detected, the slack time of remaining activities is reduced so that the process can complete before the overall deadline. Son and Kim (2001) also propose a method for determining the minimum number of servers for the activities within the critical path of a given workflow specification.

Their approach maximizes the number of workflow instances that satisfy the deadline and hence improve the performance of time-constrained workflow processing.

The M/M/1 queuing network based approach for identifying the critical path of a workflow model is detailed by Son et al. (2005). In their approach, loops are transformed into sequence control constructs for finding the longest execution paths. The main difference between their approach and ours is that in their approach, each activity is considered as an independent M/M/1 queuing system and the average execution time of a workflow instance in an activity is derived from the sum of the average servicing time and the average waiting time of the activity, whereas in our approach, an exhaustive search (breadth-first search) is used to identify the critical path based on the maximum allowable execution time of activities. In addition, the waiting time at the queue of the activity is implicitly taken into account when the conflict-free execution trace is derived from the resulting critical paths.

For preventing temporal violations in scientific workflows, Liu et al. (2010a) proposed an estima-

tion method based on the probability distribution model of the critical path. Based on the probability distribution, the workflow execution time is estimated for detecting potential temporal violations. Once a potential violation is detected, a genetic algorithm based local rescheduling strategy is used to speed up the execution of subsequent activities. An ant colony optimization based approach was also proposed by Liu et al. (2010b) for rescheduling workflows.

For predicting temporal exceptions in concurrent workflows, our approach further extends the algorithm given in Li and Yang (2005) by taking into account different kinds of loops which are identified by Van der Aalst et al. (2003). A critical path based algorithm is then used to calculate the duration of tasks in nested loops with arbitrary depth. In resource sharing, our approach allows sharing of identical resources whereas the approach proposed in Li and Yang (2005) only considers sharing of single unique resources by concurrent workflows.

Besides temporal exceptions, algorithms for predicting other types of exceptions are also proposed in a number of research reports. For instance, Yuan, Ding, and Sun (2008) propose a Support Vector Machine (SVM) to forecast potential exceptions in a workflow. In their approach, the state and operation instances of the workflow captured in a Label Petri Net are used for training the SVM.

A knowledge-based approach is also proposed by Klein and Dellarocas (2000) to determine when a workflow is going to fail. In their approach, rules are created for detecting exceptions during the workflow definition stage. The approach proposed mainly focuses on prediction of exceptions at the design time of the workflow. However, in our approach, audit trail information of the workflow instance is used to dynamically generate the execution trace which is then used for exception prediction. Therefore our approach produces a

prediction model that reflects the real situation more accurately.

A dynamic approach for detecting exceptions during workflow execution is proposed in Kammer, Bolcer, Taylor, Hitomi, and Bergman (2000). In their approach, historical information is compared with the current workflow status and feedback data is captured during the execution of a workflow. The result of the comparison is then used for detecting exceptions. In our approach, we focus on iteration control flow patterns and the XML Schema of YAWL (Van der Aalst & Ter Hofstede, 2005) is extended for modeling different kinds of loop. A prototype exception prediction system was also developed based on the YAWL workflow management system.

Modeling errors in workflow specifications can often lead to exceptions. An algorithm for validating a workflow model represented in the form of a graph is proposed in Lu, Bernstein, and Lewis (2006). The algorithm identifies the pre-conditions and post-conditions of a workflow and checks if the input and output edges of the workflow follow a valid pattern or not. In contrast, our approach does not validate the correctness of the workflow specification during design time. Instead, our algorithm utilizes predefined temporal constraints and run time information for predicting exceptions.

Exception prediction based on data warehousing and mining techniques was proposed in Grigori, Casati, Dayal, and Shan (2001). In their approach, audit trail data from a workflow management system is extracted and stored in a relational database. The audit trail data is then used in generating rules and decision trees for exception prediction in future workflow instances. The main difference between our approach and the approach proposed by Grigori et al. (2001) is that our approach allows prediction of exceptions in concurrent workflows which are required to share identical resources.

CONCLUSION

In this chapter, we describe a critical path based approach for predicting temporal exceptions in resource constrained concurrent workflows. These workflows form a highly dynamic element of a larger digital business ecosystem. During design time, we use a brute-force approach to find the paths with maximum and minimum path length. During run time, we create conflict-free execution traces for concurrent workflow instances to resolve both time and resource constraints. The conflict-free execution traces are calculated based on the critical paths as well as the shortest paths of the workflow. These execution traces are then used for predicting exceptions at selected time points. The accuracy of the proposed prediction algorithm is analyzed based on a number of simulation scenarios.

The proposed exception prediction algorithm was developed based on YAWL (Van der Aalst & Ter Hofstede, 2005) Editor 1.4 and Java 5.0. YAWL Editor is an open source tool written in Java and allows modeling of workflow specifications with different workflow patterns (Van der Aalst et al., 2003). It also supports the export of workflow specifications to XML formats.

The main contributions of our research work are two-fold; from the theoretical standpoint, we contribute to the calculation of critical paths for concurrent workflows which contain iterative control-flow patterns. The proposed critical path based approach can be effectively used to calculate the hard deadlines of the workflows. In addition, the proposed approach takes into account identical resources and real time audit trail data for predicting temporal exceptions in multiple stages. From the practical standpoint, our research opens the door to the further development of mechanisms for deriving temporal constraints and predicting exceptions for workflows which are designed with complex control-flow patterns (Van der Aalst et al., 2003).

ACKNOWLEDGMENT

This research was funded by the University of Macau under grant RG074/09-10S/SYW/FST “Resource Assignment in Business Process Simulation, Performance Monitoring During Process Enactment, and Change Management for Process Improvement.”

REFERENCES

- Boley, H., & Chang, E. (2007). Digital ecosystems: Principles and semantics. In *Proceedings of the Inaugural IEEE International Conference on Digital Ecosystems and Technologies* (pp. 398-403). Cairns, Australia.
- Casati, F. (1998). *Models, semantics, and formal methods for the design of workflows and their exceptions*. Unpublished doctoral dissertation, Politecnico di Milano.
- Grigori, D., Casati, F., Dayal, U., & Shan, M. C. (2001). Improving business process quality through exception understanding, prediction, and prevention. In *Proceedings of the 27th International Conference on Very Large Data Bases* (pp. 159-168). San Francisco, CA, USA.
- Hsu, H. J., & Wang, F. J. (2011). Detecting artifact anomalies in temporal structured workflow as reusable assets. In *Proceedings of the 35th IEEE Annual Computer Software and Applications Conference Workshops* (pp. 362-367). Munich, Germany.
- Kammer, P. J., Bolcer, G. A., Taylor, R. N., Hitomi, A. S., & Bergman, M. (2000). Techniques for supporting dynamic and adaptive workflow. *Computer Supported Cooperative Work*, 9(3-4), 269–292. doi:10.1023/A:1008747109146

- Klein, M., & Dellarocas, C. (2000). A knowledge-based approach to handling exceptions in workflow systems. *Journal of Computer Supported Collaborative Work*, 9(3-4), 399–412. doi:10.1023/A:1008759413689
- Li, H., & Yang, Y. (2005). Dynamic checking of temporal constraints for concurrent workflows. *Electronic Commerce Research and Applications*, 4(2), 124–142. doi:10.1016/j.elerap.2004.09.003
- Liu, X., Chen, J., Wu, Z., Ni, Z., Yuan, D., & Yang, Y. (2010b). Handling recoverable temporal violations in scientific workflow systems: A workflow rescheduling based strategy. In *Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud, and Grid Computing* (pp. 534–537). Melbourne, Australia.
- Liu, X., Yang, Y., Jiang, Y., & Chen, J. (2010a). (Accepted for publication). Preventing temporal violations in scientific workflows: Where and how. *IEEE Transactions on Software Engineering*.
- Lu, S., Bernstein, A., & Lewis, P. (2006). Automatic workflow verification and generation. *Theoretical Computer Science*, 353(1-3), 71–92. doi:10.1016/j.tcs.2005.10.035
- Son, J. H., Kim, J. S., & Kim, M. H. (2005). Extracting the workflow critical path from the extended well-formed workflow schema. *Journal of Computer and System Sciences*, 70(1), 86–106. doi:10.1016/j.jcss.2004.07.001
- Son, J. H., & Kim, M. H. (2001). Improving the performance of time-constrained workflow processing. *Journal of Systems and Software*, 58(3), 211–219. doi:10.1016/S0164-1212(01)00039-5
- Van der Aalst, W. M. P., & Ter Hofstede, A. H. M. (2005). YAWL: Yet another workflow language. *Information Systems*, 30(4), 245–275. doi:10.1016/j.is.2004.02.002
- Van der Aalst, W. M. P., Ter Hofstede, A. H. M., Kiepuszewski, B., & Barros, A. P. (2003). Workflow patterns. *Distributed and Parallel Databases*, 14(1), 5–51. doi:10.1023/A:1022883727209
- Xie, T., Yu, Y., & Kuang, G. (2009). A time exception handling algorithm of temporal workflow. In *Proceedings of the 2009 IEEE International Symposium on Parallel and Distributed Processing with Applications* (pp. 641–646). Chengdu, China.
- Yuan, H. T., Ding, B., & Sun, Z. X. (2008). Workflow exception forecasting method based on SVM theory. In *Proceedings of the 2008 International Symposium on computational Intelligence and Design* (pp. 81–86). Hefei, China.