Critical Path Based Approach for Predicting Temporal Exceptions in Resource Constrained Concurrent Workflows

Iok-Fai Leong
Faculty of Science and Technology
University of Macau
henryifl@umac.mo

Yain-Whar Si
Faculty of Science and Technology
University of Macau
fstasp@umac.mo

Simon Fong
Faculty of Science and Technology
University of Macau
ccfong@umac.mo

Robert P. Biuk-Aghai
Faculty of Science and Technology
University of Macau
robertb@umac.mo

ABSTRACT
Departmental workflows within a digital business ecosystem are often executed concurrently and required to share limited number of resources. However, unexpected events from the business environment and delay in activities can cause temporal exceptions in these workflows. Predicting temporal exceptions is a complex task since a workflow can be implemented with various types of control flow patterns. In this paper, we describe a critical path based approach for predicting temporal exceptions in concurrent workflows which are required to share limited resources. Our approach allows predicting temporal exceptions in multiple attempts while workflows are being executed.

1. INTRODUCTION
In a highly dynamic business environment, simultaneously executing departmental workflows forms digital business ecosystems [10] which are designed to share limited resources. In this paper we address the temporal exception prediction problems of a digital business ecosystem involving concurrent workflows.

A workflow schema defines how a business process functions within an organization [1]. Based on these schemas, Workflow Management Systems (WfMS) allocate and dispatch work to users [8]. A workflow instance is an execution of a workflow schema. During the execution of a workflow instance, some events that are not defined in the workflow schema 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 a program. Exceptions in WfMS are also similar to that of programming languages. Two types of exceptions [2] 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.

iiWAS2009, December 14–16, 2009, Kuala Lumpur, Malaysia.

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 for inventory management process and a delivery process and may share the same human resources (e.g. a group of technicians) who are responsible for managing 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 arise from such delay as temporal exceptions.

In workflow management systems, control-flow patterns [7] are used to describe the order of tasks that make up a process and the relationship between them. In this paper, we focus on predicting temporal exception for workflows which include iteration patterns (loops). Iteration pattern refers to a recursive execution of one or more tasks within a workflow. Although iteration patterns are extensively used in workflow schemas, less attention has been devoted in understanding their implications to temporal exceptions. Specially, rapid changes in the number of iterations can cause deadline violations as well as conflicts in resource usage.

Our approach can be basically divided into two phases; preparation phase and prediction phase. In the preparation phase, temporal and resource constraints are calculated for each task within the workflow schema. In the prediction phase, an algorithm is used to predict potential deadline violations by taking into account constraints calculated from the preparation phase.

This paper is organized into 7 sections. A brief introduction to resource and temporal constraints in a workflow is given in Section 2. We describe our proposed solution in Section 3 and 4. Section 5 illustrates an example on temporal exception prediction. In Section 6, we briefly review recent work. In section 7, we summarize our ideas.

2. 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).

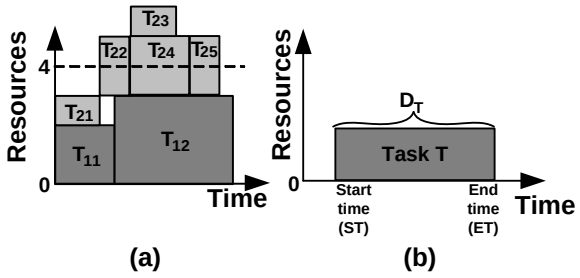


Figure 1. (a) Resource conflict in workflows (b) Temporal constraints of task T

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 right after another task, its start time will be the end time of its predecessor task. However, depending on the control flows [7] 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 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 [7], loops can be classified as structured loops, and arbitrary loops. In this paper, 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 [7]. An example of a structured loop is shown in .Figure 2

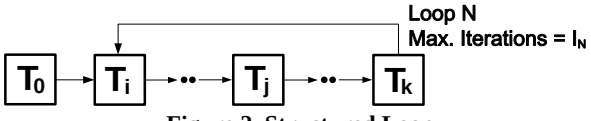


Figure 2. Structured Loop

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 [7]. For instance, in Figure 3, a loop *N* is nested within another loop *M*.

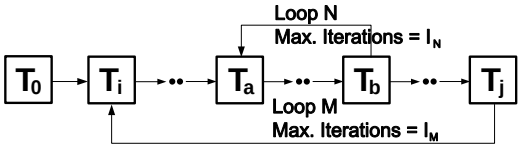


Figure 3. Nested Loop

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 [7]. In Figure 4, an example of crossing loop is depicted with two loops *N* and *M* crossing each other at T_a and T_k.

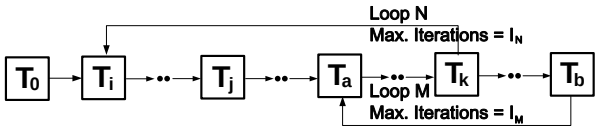


Figure 4. Crossing Loop

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) there exists at least one task in *wf* which is a member of only either one of the loops. 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*.

3. CRITICAL PATH IN WORKFLOWS

A critical path in a workflow schema can be defined 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 breadth-first-search algorithm to identify the critical path of a workflow schema. Specifically, the algorithm recursively traverses every possible path in the workflow. We assume that the workflow is well-formed and free from structural errors.

In Figure 5, we describe the calculation of the execution time for the control flow patterns defined in [8]. The resulted execution time is then used for calculating the critical path in the workflow. In case if there are loops defined in the workflow, for identifying critical paths, these loops are assumed to be executed up to their maximum number of iterations. Figure 6 shows a pseudo code for the algorithm.

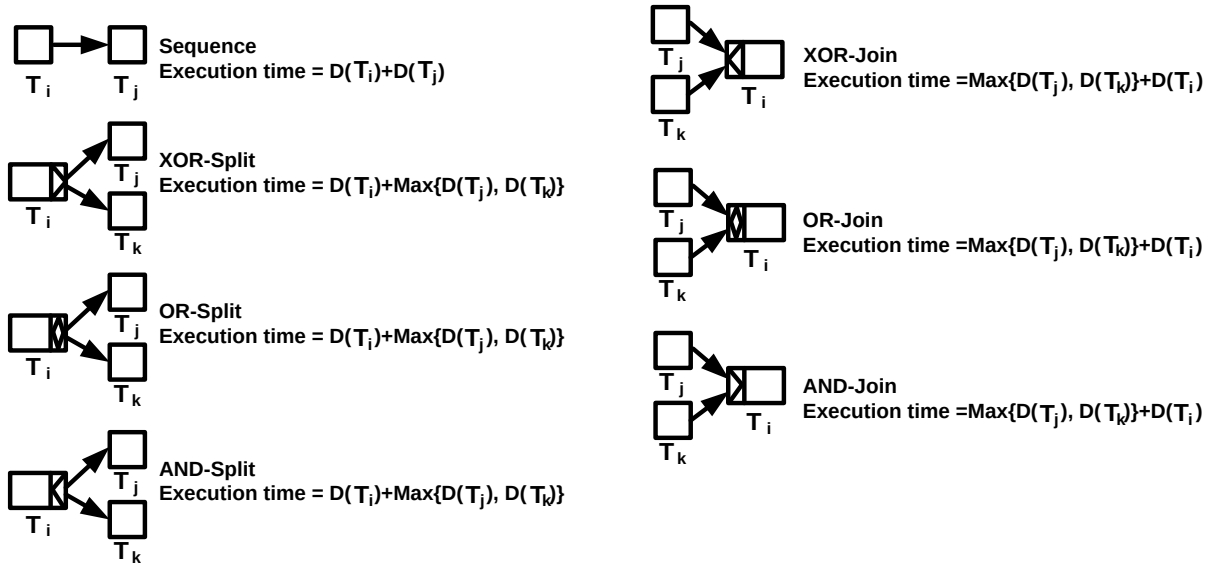


Figure 5. Execution time for different control patterns [8]

```
Initialize pathlist as an empty list of
task list
//function that returns a path in
//wf that starts with task T
Function findPath(T)
Begin
  Initialize path as an empty list of task
  Add T to path
  If T is not end task then
    For each task T' with flow from T
     Add path + result in findPath(T') to
    pathlist
  Else
    Add path to pathlist
  End if
End

Get first task T from wf
Call findPath(T)
For each path in pathlist
  Calculate the total duration of each task
Next
Longest path:=path with the longest
duration
```

Figure 6. Pseudo Code for calculating the critical path within a workflow

4. PREDICTING TEMPORAL EXCEPTIONS

The proposed prediction algorithm is separated into two steps as shown in Figure 7. The first step can be performed during design time (after workflow schemas are specified with maximum allowable execution time) and second step is performed during run time (when workflow instances are being executed simultaneously).

First, in the conflict detection and resolution step, the critical paths of the workflows are calculated. We then extend the algorithm from [8] to re-enact a conflict free *execution trace* based on the pre-defined workflow specification. In the prediction step, we use the generated execution trace from previous step to check possible deadline violations in run time for the two workflow instances which are executing in parallel.

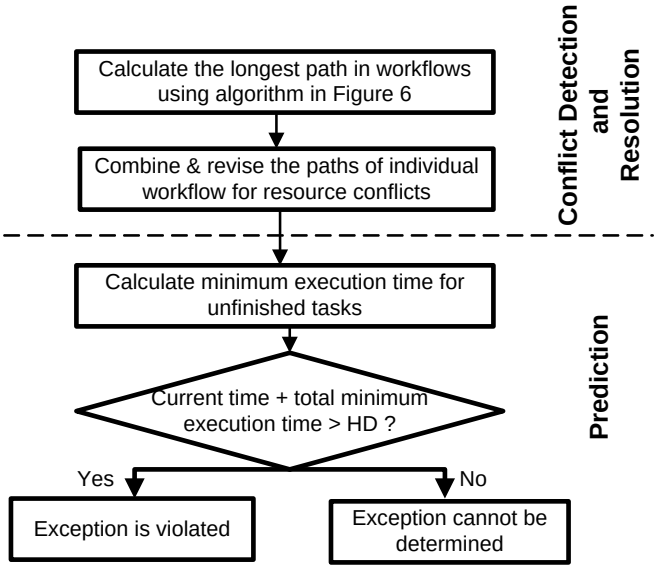


Figure 7. Steps for predicting temporal exceptions

Step I – Conflict Detection and Resolution (Design Time)

1. Calculate the critical paths based on the algorithm from Figure 6.
2. For each task T_i in *wf1* and *wf2*, we calculate $dep(T_i)$, which is a list of tasks which has resource conflicts with T_i. A task *t'* in *wf1* and *wf2* (where $t' \neq T_{i)}$ is a member of $dep(T_i)$ if

i) *resource usage of* T_i + *resource usage of* t' > *maximum number of available resources* and

ii) T_i and t' have overlapping execution time, i.e., $\exists \mu, ST(T_i) \le \mu \le ET(T_i), ST(t') \le \mu \le ET(t')$

3. If $dep(T_i)$ is not empty, then do the followings:
   i) For each task t' in $dep(T_i)$, if execution time of T_i and t' overlap, modify $[ST(t'), ET(t')]$ as $[MAX\{ST(t'), ST(T_i)+D(T_i)\}, MAX\{ET(t'), ET(T_i)+D(T_i)\}]$.
   ii) Modify the end times of subsequent tasks of t' using the new $ET(t')$ value, i.e., if t' is has a delay of n time units, each subsequent task of t' will be delayed by n time units
4. Generate an *execution trace* by taking into account revised *ET* values of each task.

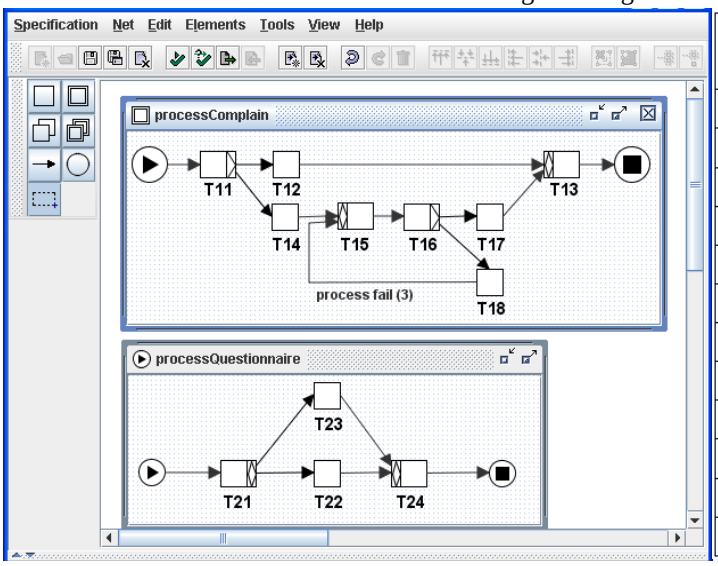
An execution trace will be created in step I, where tasks are only permitted to use the maximum allowable resources. The execution trace guarantees that the concurrent workflows will not exceed the resource usage during execution. The end time of the resulted conflict-free execution trace is the hard deadline (HD) of the workflow.

Step II – Prediction (Run Time)

5. Pick a time point *current time* for workflow instances *wi1* and *wi2*, which are instantiated from the workflow specification *wf1* and *wf2* respectively during run time.
6. Calculate the *total remaining execution time* required for all tasks including those which are currently being executed at *current time* and tasks which are planning to be executed after *current time*.
7. If (*current time* + *total remaining execution time* > *HD*), we can conclude that an exception is predicted, otherwise exception cannot be determined.

5. EXAMPLE

In this section, we outline a sample exception prediction for two workflow schemas for a complaint handling process and a questionnaire handling process (see Figure 8(a) and 8(b)).



(a)

Task *T*	Description	Duration *D(T)*	Resource usage *R(T)*
T11	Evaluate	3	2
T12	No processing	1	1
T13	Process complete	3	2
T14	Processing required	2	1
T15	Process complain	7	4
T16	Check processing	4	3
T17	Processing ok	2	1
T18	Processing not ok	1	1
T21	Prepare questionnaire	5	2
T22	Sent by post	8	3
T23	Sent by email	2	1
T24	Process complete	2	2

(b)

Figure 8. (a) Process complaint & questionnaire workflows
(b) Durations and resource usage for the tasks in both workflows

In the complaint handling workflow, the customer service staffs will evaluate the complaint to see if it needs processing or not. If it does, the staff will process the complaint by following a sequence of tasks. The staff will repeat part of the complaint handling process until the customer satisfies with the result (For the sake of illustration, we assume that the loop can only iterate for at most 3 times, as denoted in the blanket). If no processing is required, the workflow will directly execute the "No processing" task. In the process questionnaire handling workflow, the customer service staffs will send the questionnaires to the customers either by post or by email. In this example, we assume that the company has 4 staff members (resources) to be shared by both workflows. The durations and resource usages for each of the task in the workflows are specified during the design time and shown in Figure 8(b).

First, we apply the algorithm in Figure 6 to calculate the critical path for the workflows. As shown in Figure 9, the critical path from the process complaint workflow is *T11* → *T14* → *T15* → *T16* → *T18* → *T15* → *T16* → *T18* → *T15* → *T16* → *T17* → *T13* and the execution time is 45. The critical path for the process questionnaire workflow is *T21* → *T22* → *T24* and the execution time is 15. We then apply algorithm in Figure 7 to generate the conflict-free execution trace (see Figure 10(a)), which has a hard deadline (*HD*) of 53. The conflict-free execution trace from Figure 10(a) needs to be calculated only once during design time. The resulted execution trace can be used as a benchmark for predicting temporal exceptions in workflow instances during run time.

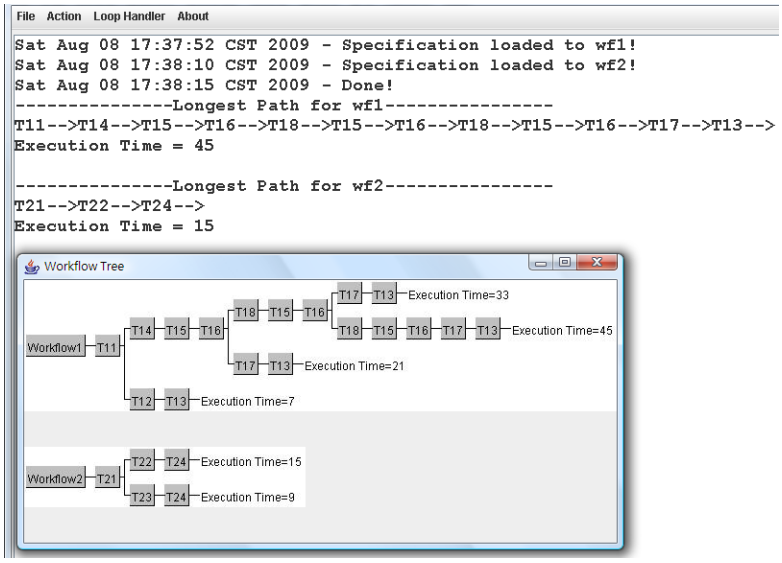


Figure 9. Tree generated by algorithm in Figure 6

Suppose that during the execution of both workflows, unexpected delay occurs at *T15* which extends its duration to 8 units. The execution trace resulted from these workflow instances is shown in Figure 10(b).

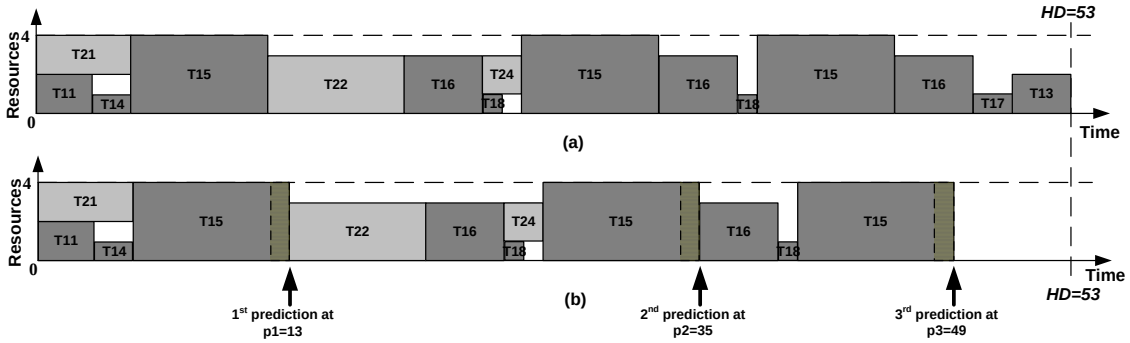


Figure 10. (a) Execution Trace generated for normal workflow instances
(b) Execution Trace when duration of T15 extends to 8 time units

Our aim is to check whether the workflow instances from Figure 10(b) will violate the previously derived hard deadline (*HD*) after all pending tasks have been executed. Prediction attempts are then made at several time points while both workflow instances proceed with their executions (see Figure 11). Figure 10(b) shows the execution trace for the both workflow instances.

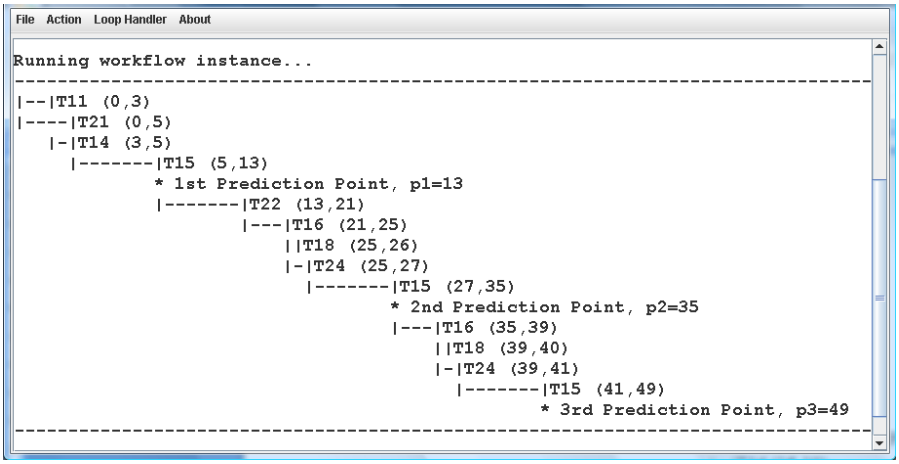


Figure 11. Predicting exception at 3 time points (stages)

1st Prediction Point, *p1 = 13*: When time = 13, only *{T11, T14, T21, T15}* are executed. Tasks *T13, T16, T17, T22* and *T24* have not been started and are waiting to be executed. We then calculate (a) the *total remaining execution* time to execute all remaining tasks and (b) *predicted end time* of the current workflow instances as follows:

Total remaining execution time

= D(T13)+D(T16)+D(T17)+D(T22)+D(T24)

= 13

predicted end time=current time + total remaining execution time

= 13 + 13 = 26

Since the predicted end time is less than *HD (26 < 53)*, deadline violation cannot be predicted.

2nd Prediction Point, *p2 = 35*: *Total remaining execution time* for the pending tasks *{T13, T16, T17} = 9* and the *predicted end time* of the workflow is *35 + 9 = 44*, which is still less than the *HD* and therefore exception cannot be determined.

3rd Prediction Point, *p3 = 49*: A final prediction attempt is made right after *T15* was executed for the fourth time. In this case, tasks waiting to be executed are *T13, T16* and *T17* and their *total remaining execution time = 9*. The *predicted end time* is therefore *49 + 9 = 58*, which exceeds the hard deadline of the workflow. Therefore we can conclude that there will be a deadline violation in future. From this example, we can see that prediction at the later stage of the workflow execution yields more accurate result. This makes sense in real world since prediction at later stage of workflow execution can exploit more accurate runtime information from the audit trail data.

6. RELATED WORK

In [11], Hyun Son et al. propose a method for determining the minimum number of servers for the activities within the critical path of a given workflow schema. Their approach maximizes the number of workflow instances that satisfy the deadline and hence improve the performance of time-constrained workflow processing.

In [1], Jin Hyun Son et al. propose a method for identifying the critical path for a given workflow schema based on queuing theory. In their approach, loops are transformed into sequence control constructs. However, to the best of authors' knowledge, no work has been reported on predicting exception in concurrent workflows which contain loops. In this paper, we use an exhaustive approach to find every possible path within a workflow for identifying the path with longest execution time. In addition, we further extend the temporal checking algorithm from [8] to allow prediction of unexpected temporal exceptions in concurrent workflows which are not only designed to share identical resources but also contain one or more loops.

M/M/1 queuing network based approach for identifying critical path of a workflow model is detailed in [1]. In their approach, loops are transformed into sequence control construct for finding the longest execution paths. The main difference between their approach and ours is that, in [1], 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 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 maximum allowable execution time of activities. In addition, the waiting time at the queue of the activity is implicitly taken into account when conflict-free execution trace is derived from the resulted critical paths.

In [5], a knowledge-based approach is used to determine when a workflow is going to fail. In their approach, rules are created for detecting exceptions during workflow definition stage. The approach proposed in [5] 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 [3]. In their approach, historical information is compared with current workflow status and feedback data 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 XML Schema of YAWL [9] is extended for modeling different kinds of loop. A prototype exception prediction system was also developed based on YAWL workflow management system.

Modeling errors in workflow schemas can often lead to exceptions. An algorithm for validating a workflow model represented in the form of a graph is proposed in [6]. The algorithm identifies the preconditions and post-conditions of a workflow and checks if the input and output edges of the workflow follows 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 exception.

Exception prediction based on data warehousing and mining techniques is proposed in [4]. In their approach, audit trail data from 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 [4] is that, our approach allows predicting of exceptions in concurrent workflows which are required to share identical resources.

For predicting temporal exceptions in concurrent workflows, our approach further extends the algorithm given in [8] by taking into account different kinds of loops which are identified in [7]. 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 [8] only considers sharing of single unique resources by concurrent workflows.

In [8], Li et al. addressed the problem of predicting unexpected exceptions in concurrent workflows by verifying temporal constraints. However, the approach described in [8] only considers single unique resources. In this paper, we address this issue by

taking into account identical resources which are shared by one or more concurrent workflows.

7. CONCLUSION

In this paper, 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 create a conflict free execution trace for concurrent workflows to resolve both time and resource constraints. The conflict free execution trace is calculated from the critical path of the workflow. During run time, we use the generated execution trace to predict exceptions in workflow instances which are in the progress of executing. The proposed exception prediction algorithm is developed based on YAWL [9] 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 [7]. It also supports exportation 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 algorithm effectively reduces the complexity in calculating hard deadlines for tasks within 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 [7].

8. ACKNOWLEDGEMENT

This research is funded by the University of Macau under grant RG056/06-07S/SYW/FST "Audit Trail Data Management, Intelligent Deadline Escalation, and Exception Prediction in Workflows".

9. REFERENCES

[1] Son, J.H., Kim, J.S., and Kim, M.H., 2005. Extracting the workflow critical path from the extended well-formed workflow schema, Journal of Computer and System Sciences, 70(2005), pp. 86-106.

[2] Casati, F., 1998. Models, semantics, and formal methods for the design of workflows and their exceptions, PhD Thesis.

[3] Kammer, P.J., Bolcer, G.A., Taylor, R.N., Hitomi, A.S., and Bergman, M., 2000. Techniques for supporting dynamic and adaptive workflow, Computer Supported Cooperative Work, 9(3-4), pp. 269 – 292.

[4] Grigori, D., Casati, F., Dayal, U., and Shan, M.C., 2001. Improving business process quality through exception understanding, predication, and prevention. In Proceedings of the 27th International Conference on Very Large Data Bases, San Francisco, CA, USA, pp. 159 – 168.

[5] Klein, M., and Dellarocas, C., 2000. A knowledge-based approach to handling exceptions in workflow systems, Journal of Computer Supported Collaborative Work, 9(3-4), pp. 399 – 412.

[6] Lua, S., Bernsteinb, A., and Lewis, P., 2006. Automatic workflow verification and generation, Theoretical Computer Science, 353(1-3), pp. 71 - 92.

[7] van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., and Barros, A.P., 2003. Workflow Patterns, Distributed and Parallel Databases, 14(3), pp. 5-51.

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

[9] van der Aalst, W.M.P., and ter Hofstede, A.H.M., 2005. YAWL: Yet Another Workflow Language, Information Systems, 30(4), pp. 245-275.

[10] Boley, H., and 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, February 21-23.

[11] Son, J.H., and Kim, M.H., 2001. Improving the Performance of Time-constrained Workflow Processing. Journal of Computer and System Sciences, 70(2001), pp. 211-219.