BUSINESS PROCESSES: BEHAVIOR PREDICTION AND CAPTURING REASONS FOR EVOLUTION

Sharmila Subramaniam, Vana Kalogeraki and Dimitrios Gunopulos
Department of Computer Science and Engineering
University of California, Riverside
900 University Ave, Riverside, CA 92521

Keywords: Workflow graph models, classification, workflow mining.

Abstract: Workflow systems are being used by business enterprises to improve the efficiency of their internal processes and enhance the services provided to their customers. Workflow models are the fundamental components of Workflow Management Systems used to define ordering, scheduling and other components of workflow tasks. Companies increasingly follow *flexible* workflow models in order to adapt to changes in business logic, making it more challenging to predict resource demands. In such a scenario, knowledge of *what lies ahead* i.e., the set of tasks that are going to be executed in the future, assists the process administration to take decisions pertaining to process management in advance. In this work, we propose a method to predict possible paths of a running instance For instances that deviate from the workflow model graph, we propose methods to determine the characteristics of the changes using classification rules.

1 INTRODUCTION

Workflow management systems are widely being used by many business companies to automate their processes, and refine and improve the services they offer to their customers. Workflow process models define the ordering, scheduling and dependencies of the workflow tasks, enabling process automation, maintenance, diagnosis etc. The efficiency of a business' internal and external processes plays a vital role in determining the competency of its services.

Today, workflows need to be flexible (Sadiq et al., 2005a; Sadiq et al., 2005b) in order to accommodate varying business process requirements and provide fast and reliable services. In addition, since workflow models are designed manually, they may not incorporate the complete set of business logics that drive the processes and, thus, often result in suboptimal service performance. Furthermore, frequent changes in business requirements trigger corresponding changes in the definition of the process workflow (van der Aalst et al., 2003; Casati et al., 1998). Due to these reasons, the process instances do not always follow the workflow graph model. Even in cases when the instances follow the graph model, the complexity of the workflows makes it very difficult to predict the future state of a running instance accurately.

In our work, we consider workflow systems that follow flexible models, as described in (van der Aalst et al., 2003; Sadiq et al., 2005a). We propose a method for predicting the behavior (i.e., determine the future tasks) of a long running process instance, by analyzing the data stored in the workflow log of the corresponding process. For the instances that do not follow the graph model, we identify the likely conditions for the deviations that happen. We use these conditions to change the workflow models, where appropriate.

Predicting the future of running instances helps in assessing their future resource requirements and in scheduling their execution efficiently. As a motivating example, consider the graph model shown in Figure 1. Let us assume that the administrator is required to schedule the upcoming tasks at the stage ST of execution. Let $R1$ be the resource requirement (*e.g.*, cpu, memory, disk) for the sub-process marked as $S1$; where the load on resource $R1$ is constrained by its maximum value $R1_{max}$. Suppose there are n instances waiting at stage ST and the administrator has to verify if all these instances can be scheduled to proceed, without overloading $R1$. Such a scenario is very frequent in any resource constrained business environment. A pessimistic approach would be to assume the worst case scenario and ensure that the sum

Subramaniam S., Kalogeraki V. and Gunopulos D. (2006).
BUSINESS PROCESSES: BEHAVIOR PREDICTION AND CAPTURING REASONS FOR EVOLUTION.
In *Proceedings of the Eighth International Conference on Enterprise Information Systems - ISAS*, pages 3-10
DOI: 10.5220/0002464900030010

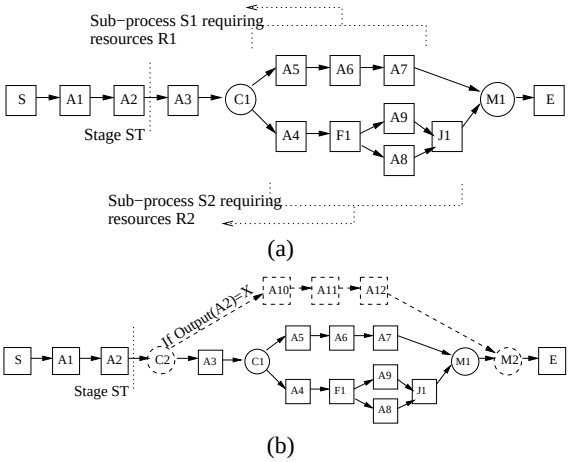


Figure 1: (a) $S1$ requires resources $R1$ and $S2$ requires resources $R2$. ST is the stage where the future activities are predicted. (b) Tasks $A10$, $A11$ and $A12$ are added to the process model when the frequency of the instances $S, A1, A2, A10, A11, A12, E$ is sufficiently high in the workflow log. The characteristic of this instance is determined as $Output(A2) = X$ and a choice node is added to capture it.

of the resource requirements of all the n instances is less than $R1_{max}$ at any point of time. This could be achieved by doing offline analysis of all the processes and their corresponding resource requirements. However, as stated, this is a pessimistic approach, and may end up under-utilizing the resources. Furthermore, it does not take into account the possibility that $S2$ could be chosen at $C1$. Our approach is to predict how many of the n instances are likely to follow the $S1$ path. Predicting the path will enable us to schedule the execution of the instances accordingly and, thus, achieve better resource management.

As stated earlier, in a flexible workflow environment, the instances in the workflow log or the predicted path of the current instance need not adhere to the workflow model graph. For example, some instances in the workflow log corresponding to the process shown in Figure 1 may have the following task execution order: $S, A1, A2, A10, A11, A12, E$. If the frequency of occurrence for such instances is found to be *high*, then the initial graph model has to be modified to adapt to this change. Characteristics of such instances, i.e., *executed if* $Output(A2) = X$, are identified and the model graph is modified accordingly.

For our work, we assume that an initial workflow graph model that corresponds to the process execution is given. We require that the workflow log contains the ordered set of events of each execution and the input/output data values of these events.

In this paper, we propose the following:

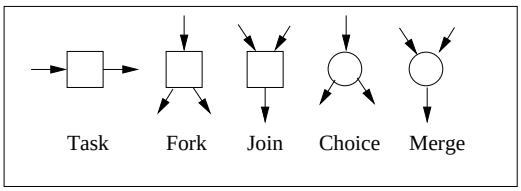


Figure 2: Workflow Graph Constructs.

- We denote the possible set of instances of a workflow model graph as *workflow instance types*. Given a running process, we propose to predict the behavior of this instance type. We apply a classification algorithm to the workflow log, considering the instance types as *classes*, and predict the instance type of the running process instance. The prediction is performed based on the output values of the set of activities executed so far.
- For evolving workflows, some of its instances do not adhere to the workflow graph. We propose to learn such changes over time and estimate their characteristics (i.e, the conditions under which the instances deviate from the workflow graph model). This can assist in the process re-engineering step where the process model is redesigned to adapt to the changes and thus improve its performance.

2 PRELIMINARIES

In this section, we give a brief overview of the workflow graph model used in this paper and give definitions for the *workflow log* and *workflow instance types*. We model the processes using workflow graph technique similar to that described in (Greco et al., 2005). This is homomorphic to the other models proposed in the literature (van der Aalst and van Hee, 1996; Georgakopoulos et al., 1995).

Definition (Workflow Graph G_P): *The control flow graph G_P of a process P is a tuple (A, S, E, CF) where A is the set of activities, $S \in A$ is the starting activity, $E \in A$ is the final activity, $CF \subseteq (A - E) \times (A - S)$ is the set of edges defining the control flow sequence between the activities. The activities can refer to a task in the process or any router nodes, choice or fork.*

Figure 2 illustrates the building blocks of the workflow model. Their descriptions are as follows:
Task Nodes (T) represent the individual tasks needed to accomplish the process. **Fork Nodes** represent the AND split. The fan-out paths that split from a fork node synchronize later using the Join node. The **Join**

Node waits until all the control flow in-transitions of the node are triggered, before proceeding with the next activity. **Choice Node** (or Decision Point) represent the XOR split, having mutually exclusive/ alternative paths out of it. **Merge Node** merges the exclusive paths out of the Choice node into one path. It is triggered when any one of the control flow intransitions is fired.

Input data container $Input(a)$ ($a \in A$) and output data container $Output(a)$ ($a \in T$) of a node a are defined as:

- $\forall a \in A$, $Input(a)$ is a set of data items V_i that a consumes, where $V_i \subseteq$ *Workflow Data*
- $\forall a \in T$, $Output(a)$ is a set of data items V_o that a writes to, where $V_o \subseteq$ *Workflow Data*

where *Workflow Data* is the set of data items for the process under consideration (Sadiq et al., 2004).

An *instance* of a workflow process is a single realization of the process. Collection of the instances produced by a process workflow is referred to as *workflow log*.

Definition (Workflow Log L): *For a process P, a workflow trace wt is a string $(T_i.V_o^j)^*$ representing the sequence of tasks and their outputs, where T_i is the identifier for task that was executed and V_o^j is the j^{th} value recorded(output) by the task.* Workflow Log *of process P, L_P is thus defined to be a set of traces wt, i.e. $L_P = \{wt\}$.*

Instances of a workflow graph model may process different sets of activities due to the presence of the choice nodes. All possible workflow instances of a process can be classified into workflow instance types as described in (Gruber, 2004).

Definition (Workflow Instance Types): *A workflow instance type refers to the workflow instances where at each choice node, the same successor node is chosen.*

Thus the workflow instances of a workflow instance type contain exactly the same set of activities. We denote the set of possible instance types of a graph model G as E_G. For the workflow graph model shown in Figure 1(a), following are the instance types: $(S, A_1, A_2, A_3, A_5, A_6, A_7, E)$ and $(S, A_1, A_2, A_3, A_4, (A_8)||(A_9), E)$. Note that more than one type of trace can be associated with an instance type. For example, (S, A_1, A_2, A_3, A_4, A_9, A_8, E) and (S, A_1, A_2, A_3, A_4, A_8, A_9, E) are the traces that belong to the second instance type above.

3 PROBLEM DEFINITION

In this paper, we consider the problem of predicting the future execution of an active workflow instance, which will enable efficient resource management and better understanding of the future requirements of the instance. Our approach is based on analyzing the workflow log and predicting the future execution in terms of the *instance types* (as defined by (Marjanovic and Orlowska, 1999)). Predicting the future of the active instance l_c in terms of the possible *instances* or the possible *instance types* implicitly takes into account the correlation between the tasks that are executed in sequence i.e., whenever A_5 is executed, it is followed by task A_6 in the process model shown in Figure 1. This will not be true if we predict the future as the set of tasks that will be executed in future.

The number of possible instances in a workflow is typically exponential and we seldom have sufficient data to train for all possible instances. Though theoretically it can be equal, the number of possible instance types is far less than the number of instances in most of the workflows in business enterprises.

Therefore, we propose to forecast the future of an active instance by predicting its possible *instance types*. In particular, the problems we consider in this paper are the following:

Problem 1 (Prediction of Paths) *Assume a process P, its control flow graph G_P and the set of process instances L_P. For the current instance l_c, and the current task T_i that is being executed as a part of l_c, our goal is to predict the set of tasks that are most likely to be executed toward completion of l_c.*

We address the problem by applying classification techniques to the workflow log - analyze traces with similar output values as in l_c and predict the future tasks. In our approach to solving the above problem, we classify the possible traces in L_P according to their *Instance Types*. We predict the future set of activities in terms of the instance type that the current process instance is likely to be in.

In current business enterprises, processes follow flexible workflows where the workflow graph models serve as a guidance for the process execution. This flexibility in workflow systems is necessary to allow refinement and accommodate process logics that were missed in the design phase. In such scenarios, instances can deviate from the process definition provided by the graph model. Such deviations could either be *exceptions*, i.e., a rare occurrences, or *evolutions* i.e., the instance adapting to a change in the process logic. In the latter case, it becomes necessary to periodically discover such instances and restructure the graph model accordingly.

When a set of instances deviates from the initial process definition, we determine whether it is an *exception*, or if it calls for a change in the graph model.

Problem 2 (Capturing Evolution of Models) *If a set of paths deviate from the workflow definition in G_P, find the conditions under which the deviation occurs and determine if it is an* evolution *of the workflow that will require adaptation of the graph model.*

If an instance from the workflow log does not match any of the instance types in E_{G_P} we consider that as a *deviation*. For deviating instances that occur frequently, we identify their characteristics in terms of classification rules. This serve as input to workflow graph restructuring. Essentially, we keep track of the *exceptions*, and if the frequency of an exceptions is high we assume that the workflow model has to be updated.

4 PREDICTION OF PATHS

In this section, we explain our method for predicting the instance types, for a given active instance l_c. We illustrate the problem with an example process model and explain the algorithm.

We consider the *Order Processing Workflow* model graph shown in Figure 3. The workflow illustrates the tasks involved in processing orders for items bought by customers. After verification of billing information of an order, the order is sent to inventory control system. If the item that is ordered is not in the inventory, it is shipped from suppliers A or B or C, with A being the highly preferred supplier, then B followed by C. To minimize the resources used for transportation of goods from a supplier, it is natural for the dealer (who deals with the supplier) to wait for as many orders as he could handle (for short, $Supplier_{Max}$), before contacting the supplier. However, waiting for the orders to arrive incurs delay in processing the previous orders. For example, let us consider the dealer who deals with the supplier B. Assuming equal probabilities for all choices at each choice node, only 25% of the orders arrive at supplier B, which implies considerable wait time before B_{Max} is reached.

In such situations, it might be of interest to determine in advance how many of the current orders are likely to arrive at B. The dealer can estimate the time taken to reach B_{max} and thus decide between sending the order list to the supplier and waiting for further orders. This illustrates one situation where predicting the future activities assists in improving process efficiency.

Let us consider the problem of estimating if a given order will be ordered at supplier B, for the current task $T_i = RBI$. IT is a set of instance types consisting of all the instance types corresponding to the workflow log. Thus, IT can contain some instance types that are not defined by the workflow (this is possible due to previous *exceptional* instances).

Let there be n traces in the corresponding workflow log L, each denoted by $l_1, l_2, l_3, ..., l_n$ and the current trace be denoted by l_c. For l_c, let the log so far consist of:

(S, LOG_AUTH.isEmployee = yes, LOG_AUTH.discountRate = 10%, ROI.orderNo = 3456, ROI.itemId = 56, ROI.itemCount = 2, RBI.isBillingAuth = yes)

We define sub-trace Pre(task, trace) of a trace as follows:

Pre(task, trace): For a task T_k executed during process instance l_r, we define $Pre(T_k, l_r)$ to be the set of tasks in trace l_r that are executed preceding task T_k.

We consider two sub-traces $Pre(T_k, l_r)$ and $Pre(T_k, l_{r'})$ to be symbolically equal i.e., $Pre(T_k, l_r) \simeq Pre(T_k, l_{r'})$, if either the sub-traces are identical or if for all tasks $a, b \in Pre(T_k, l'_r)$ such that a is executed before b in $Pre(T_k, l'_r)$ and b is executed before a in $Pre(T_k, l_r)$, a is executed parallel to b according to the graph G_P.

As the first step, we extract all the traces l_j from log L that consist of task T_i and where $Pre(T_i, l_j) \simeq Pre(T_i, l_c)$. For the example under consideration, let the following be two of the traces in L_P:

Trace a: (S, LOG_AUTH.isEmployee = yes, LOG_AUTH.discountRate = 15%, ROI.orderNo = 3401, ROI.itemId = 24, ROI.itemCount = 1, RBI.isBillingAuth = yes, SEND_INV.status = available, GET_INV.status = ready, PACK.status = ready, SEND.status = complete)

Trace b: (S, ROI.orderNo = 3413, ROI.itemId = 32, ROI.itemCount = 3, RBI.isBillingAuth = yes, SEND_INV.status = notAvailable, SEND_A.status = ready, GET_A.status = complete, PACK.status = ready, SEND.status = complete)

Thus, $Pre(PBI, l_c)$ will be symbolically equal to the sub-trace $Pre(T_i, a)$ but not the sub-trace $Pre(T_i, b)$.

From the set of traces l_m in L that consist of task *PBI* and where $Pre(PBI, l_c) \simeq Pre(PBI, l_m)$, we extract the sub-traces $Pre(PBI, l_m)$ and denote the set as L'. For each instance l_m in L', we add the instance type of l_m to the set of instance types IT, if it is not already present in IT. We extract those items in l_m corresponding to tasks in $Pre(PBI, l_m)$ and form the training set $TrainingSet$.

A sub-trace from Trace a with the information about its instance type (given below) is an example of a record in $TrainingSet$.

(S, LOG_AUTH.isEmployee = yes, LOG_AUTH.discountRate = 15%, ROI.orderNo = 3401, ROI.itemId = 24, ROI.itemCount = 1) : IT1.

where **IT1** is S, LOG_AUTH, ROI, RBI, SND_INV, GET_INV, PACK, SEND, E.

We apply a classification algorithm to the set of sub-traces from $TrainingSet$ with the instance types in IT as classes. The outputs of the tasks in $Pre(PBI, l_m)$ are considered as the classification attributes. In particular, we apply decision tree algorithm C4.5 (Quinlan, 1993) to generate classification rules, because it can handle missing items in the $TrainingSet$ gracefully. We apply the rules to the current instance l_c and forecast the instance types of l_c as the instance type with the highest accuracy. For example, the classification rules gener-

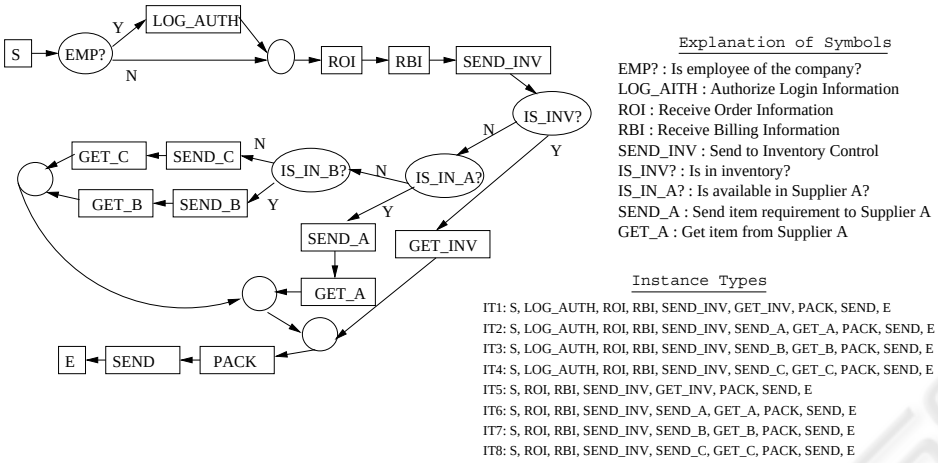


Figure 3: Example Process Model: Order Processing.

ated for the process model under consideration are: If $ROI.itemId < 50$ and $ROI.itemId > 10$ and $ROI.itemCount < 10$: IT1 (with accuracy=55%) If $ROI.itemId < 100$ and $ROI.itemCount > 50$: IT3 (with accuracy=67%)

Returning to the problem discussed in the previous section, we see that it can be solved by predicting the instance types of all the current orders, and estimating the number of orders that are likely to be in instance type IT. For example, suppose O is the set of orders that are likely to be in instance type $IT3$ or $IT7$, and the accuracy of each of the orders $o_c \in O$ is $acc(o_c)$. The dealer can wait for the orders, if $\sum_{o_c \in O} acc(o_c) > thr$ where thr is a user defined threshold value.

Algorithm in Figure 4 refers to the steps involved in training and predicting the instance types of a current instance l_c. IT, the set of instance types corresponding to the workflow log, is given as input to the algorithm. In steps 3 to 5 of the algorithm (a), the workflow log is scanned and new instance types, if any, are added to IT. In the algorithm, part (a) correspond to training the classifier. If prediction is performed frequently, after a particular stage of process execution, then the classifier need not be trained for every *prediction*. Instead, already created models can be used to predict the possible instance types (with steps in part (b)).

As seen above, we group possible instances of a workflow graph into instance types, and therefore we do not keep the information about the execution order of the tasks that are executed in parallel to each other. Thus, our method is suitable for applications (such as that explained in Section 1) where execution order of the tasks executed in parallel i.e., A_8 and A_9 in Figure 1, is insignificant.

Given: Process P, its workflow model graph G_P, workflow log L_P and the set of instance types IT corresponding to workflow log L_P

(a) Train the classifier for a task T_i and current trace l_c.
1. Let $L'_P \subseteq L_P$ be the set of traces that consists of task T_i and where the set of tasks executed before T_i is same as that in l_c, i.e, $L'_P = [l_m | l_m \in L_P \text{ and } Pre(T_i, l_m) \simeq Pre(T_i, l_c)]$.
2. **for** each trace l_m in L'_P
3. $\quad e_{l_m} \leftarrow$ *Instance Type of* l_m
4. $\quad$ **if** $e_{l_m} \notin IT$
5. $\quad\quad IT = IT \cup [e_{l_m}]$
6. $\quad l'_m \leftarrow$ Items in l_m that correspond to tasks in $Pre(T_i, l_m)$
7. $\quad TrainingSet \leftarrow TrainingSet \cup [l'_m]$
8. Apply a decision tree algorithm to $TrainingSet$ considering the elements in IT as $classes$.

(b) Predict the instance type of the current instance l_c.
1. Using the classification rules generated by (a), classify the current instance l_c into one of the instance types, with certain accuracy as determined by the classifier.
2. $e_{l_c} \leftarrow$ The instance type with highest accuracy for l_c
3. Return e_{l_c}

Figure 4: Algorithms for training the classifier and Predicting the instance type of a current instance.

5 CAPTURING EVOLUTION

When an instance of a process P deviates from its workflow graph G_P, i.e., if extra tasks are executed or some existing tasks are skipped, then the instance will not belong to any of the instance types in E_{G_P}. Such instances are possible due to two reasons (a) Changes in the process definition, (b) Business logics that were not captured initially by the workflow graph. The set of traces L_P are analyzed periodically to check for any such deviations.

In the example shown in Figure 3, let us assume that the company decided to purchase a new item with $itemId = X$ from a supplier D. Thus, the orders for item X do not follow the given workflow and are routed to supplier D. That is, if $ROI.itemId = X$ in l_c, then the predicted instance type of l_c is differ-

ent from the instance types in E_{G_P}. Let us refer to this new instance type as e_{new}. If the number of such instances is not significant (as determined by the modeler), then we consider it as an *exceptional* behavior. However, if the frequency of such instances is sufficiently high i.e., $\frac{frequency(e_{new})}{|L_P|} > threshold$, we extract the corresponding classification rule generated by the decision tree algorithm. For example,
If $ROI.itemid = X$ then $InstanceType = e_{new}$ **(Rule I)**
In such cases, the modeler can modify the existing workflow graph to capture the deviation. Figure 5 shows the process workflow model when the initial model is restructured to capture the above rule.

Algorithm in Figure 6 describes the steps involved in determining reasons for process evolutions. The classifier is retrained (step 1-2), and the rules corresponding to new instance types are analyzed to verify if they indicate process *evolution*. The workflow model is modified to adapt to the changes. The issues related to structural changes in graph models, and dynamic and smooth migration of running instances to the new graph model are discussed in depth in literature (Rinderle et al., 2004; van der Aalst and Basten, 2002; Casati et al., 1998). The technique we describe in this paper for determining the reasons for evolutions is orthogonal to how the graph model is modified, verified for correctness, etc.

When an initial workflow model G is modified to G', the set of instance types IT is updated as following:

E_1: The set consisting of those instance types in E_G that are valid for G'

E_2: The set consisting of those instance types in E_G that are *invalid* for G'

E_{new}: Set of new instance types in G'.

E_3 : Instance types that were exceptions *wrt* to G and are still exceptions *wrt* G'

Initially, $IT = E_1 \cup E_2 \cup E_{new} \cup E_3$. We update IT as $IT = IT - E_2$ and the set of instance types E' for G' as $E_1 + E_{new}$.

Thus, we make sure that no obsolete instance types are present in IT, and it is updated with the new instance types corresponding to the changes in the graph. When the workflow graph is modified, (a) the updated set of IT is used for predicting the paths, and (b) the traces in the workflow log, corresponding to instance types in E_2, are removed before building the models. We keep the traces corresponding to E_3 in the workflow log because there is a chance that these traces will become frequent later, requiring another change.

6 EXPERIMENTAL EVALUATION

In this section, we present our preliminary results for prediction of future paths in a workflow model. We generate the workflow graph models for our experiments using an incremental method described below. The graph generation procedure takes the following as inputs: number of nodes to be present in the graph and probabilities of adding a task node, a fork-join structure and a choice-merge structure at each iteration. Initially a simple sequence structure with a single task node encompassed between a start and an end node is generated. During further iterations of incremental additions, a random task node is chosen from the graph and is converted to either a sequence of task nodes or a fork-join structure or a choice-merge structure, with the given probability. The incremental graph generation assures that the generated model is free of structural conflicts (Aalst et al., 1994; Sadiq and Orlowska, 2000).

Given: Process P, its workflow model graph G_P, workflow log L_P and the set of instance types IT corresponding to workflow log L_P. For the instances that are not in E_{G_P}, find the corresponding classification rules
1. Apply a decision tree algorithm to L_P considering the elements in IT as $classes$.
2. $Rules \leftarrow$Classification Rules obtained from the decision tree
3. **for** each rule r in $Rules$
4. $e_{new} \leftarrow$ Instance type associated with r
5. **if** $e_{new} \notin E_{G_P}$
6. Calculate $support(e_{new})$
7. **if** $support(e_{new}) > threshold$
8. Modify G_P to accommodate rule r
9. Return modified graph model

Figure 6: Determining reasons for process evolution.

We compare the performance of our technique with an alternative simpler technique where we use the current partial execution of the process to predict, for each future task, if it will be executed or not. The fundamental difference with our approach is that in this technique each future task has to be predicted individually. To solve this **Task Correlation** problem, we can extend recent work on workflow mining ((Grigori et. al. 2001), (Subramaniam et. al. 2005)) which provide efficient techniques for predicting whether specific nodes in the workflow will be taken in the future. Essentially, in this approach we have to build and train separate classifiers to predict each task.

We present our initial results of comparing the accuracy and the speed of the two approaches, Task Correlation TC and Instance Type Correlation ITC. For our experiments, we considered a sample process model with 13 task nodes and 6 instance types as shown in Figure 7(a). We used the decision tree algorithm $C4.5$ for generating the classification rules. Figure 7(b) shows the *training time*, *query time* and the *accuracy* of the two methods. Training time is the time taken to build the models, and it is determined by the number of models to be trained and the number of attributes in the training data (in this case, it is proportional to the number of tasks that are executed before the point of prediction i). Figure 7(b) shows the total training time (for all the tasks and for all the models) for the two approaches. We observe from the values

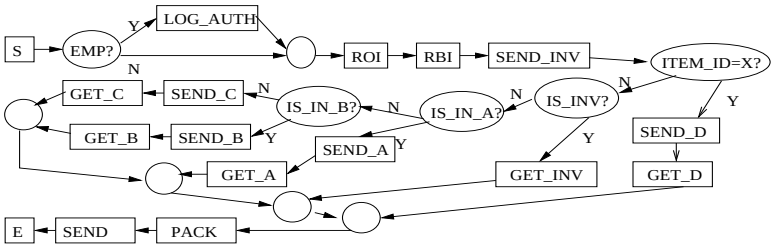
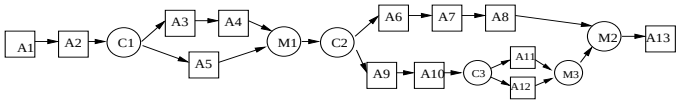


Figure 5: Restructured process model after capturing rule Rule I.



(a)

Total Training Time (secs)		Average Query Time (secs)		Average Accuracy	
TC	ITC	TC	ITC	TC	ITC
13.7995	3.5191	0.3338	0.070	56%	67%

(b)

Figure 7: (a) Process model graph used for the experiment (b) Comparison of the total training time, average query time and the average accuracy of the two approaches TC and ITC.

that ITC is much faster than TC with respect to the training time. This is due to the fact that the number of decision trees to be built for TC is more than that of ITC. For example, for task $A2$, the number of decision trees required to be constructed for TC methods is 11 (one for each of the future tasks, with *executed* and *not executed* as classes), whereas it is 1 for our method.

For ITC, query-time is the time taken for an active instance l_c to be classified into one of the instance types. For TC, it is the sum of the time taken to classify each of the future tasks as *executed* or *not executed*. In Figure 7(b), we compare the average query-time of the two approaches. The average query time for ITC is lesser than that of TC, because only few decision trees need to be built for each task, with method ITC. Furthermore, we observed that the values for TC ranges from 0 seconds to 83 seconds, with the average of 0.3338 seconds. Figure 7(b) also shows the average of the accuracy of the classification rules generated by the two methods. The accuracy of ITC and TC are comparable as seen from the figure.

7 RELATED WORK

Mining of the workflow logs has been applied in various phases of WfMS. A summary of the ongoing research in process mining is given in (van der Aalst et al., 2003). Methods of automating the process model construction through event data capture of the on going process were illustrated in literature (Cook and Wolf, 1995; Cook and Wolf, 1998). Another approach of process discovery proposed in (Agrawal et al., 1998) makes use of logs of past unstructured executions of the given process to construct the model. Event data analysis and process mining has been applied to model rediscovery in (Weijters and van der Aalst, 2002). In (Herbst and Karagiannis, 1998), the authors propose machine learning techniques to acquire and adapt workflow models by analyzing event data in the workflow log.

In (Grigori et al., 2001; Castellanos et al., 2005), the authors propose methods to predict the future behavior of a workflow instance in terms of pre-defined metrics, for example, resulting in exception or not etc. However, predicting possible paths of a running instance is not addressed in these works.

Workflow evolution has been discussed in many research papers (Casati et al., 1998; van der Aalst and Basten, 2002; Rinderle et al., 2004). These works discuss and propose frameworks for workflow schema modifications and how workflow instances can (dynamically) adapt a newly evolved workflow schema. All of them assume prior knowledge of the structural changes due in the evolving workflow. Issues related to process quality of flexible workflows is addressed in (Sadiq et al., 2005a). In (Rinderle et al., 2005), the authors propose a *case based reasoning* approach to learning process evolution. However, this method is useful only for processes where there is a user involvement at each level of the process - to interact with the CCBR system. Thus, it is only semi-automated and relies on the user input for extracting reasons for changes. Thus, none of them consider the problem of automatically determining the semantics for evolutions by analyzing the workflow log.

8 CONCLUSION

In this paper, we have proposed a method to predict future paths of a running instance of a workflow by analyzing past workflow logs. For those instances that deviate from the workflow models, we identify the possible conditions under which the deviations occur. Providing insights about what modifications are required to a workflow can be a significant input to the the workflow evolution.

REFERENCES

Aalst, W., Hee, K., and Houben, G. (1994). Modelling and analysing workflow using a Petri-net based approach. *Proceedings of the second Workshop on Computer-Supported Cooperative Work, Petri nets and related formalisms*, pages 31–50.

Agrawal, R., Gunopulos, D., and Leymann, F. (1998). Mining process models from workflow logs. *EDBT '98: Proc. of the 6th Intl. Conf. on Extending Database Tech.*, pages 469–483.

Casati, F., Ceri, S., Pernici, B., and Pozzi, G. (1998). Workflow evolution. *Data Knowl. Eng.*, 24(3):211–238.

Castellanos, M., Casati, F., Shan, M.-C., and Dayal, U. (2005). ibom: A platform for intelligent business operation management. *ICDE*, pages 1084–1095.

Cook, J. E. and Wolf, A. L. (1995). Automating process discovery through event-data analysis. *International Conference on Software Engineering*, pages 73–82.

Cook, J. E. and Wolf, A. L. (1998). Discovering models of software processes from event-based data. *TOSEM '98: ACM Transactions on Software Engineering and Methodology*, 7(3):215–249.

Georgakopoulos, D., Hornick, M. F., and Sheth, A. P. (1995). An overview of workflow management: From process modeling to workflow automation infrastructure. *Dist. and Parallel Databases*, 3(2):119–153.

Greco, G., Guzzo, A., and Pontieri, L. (2005). Mining hierarchies of models: From abstract views to concrete specifications. *Business Process Management*, pages 32–47.

Grigori, D., Casati, F., Dayal, U., and Shan, M.-C. (2001). Improving business process quality through exception understanding, prediction, and prevention. *VLDB '01*, pages 159–168.

Gruber, W. (2004). Modeling and transformation of workflows with temporal constraints. *Akademische Verlagsgesellschaft, Berlin*.

Herbst, J. and Karagiannis, D. (1998). Integrating machine learning and workflow management to support acquisition and adaptation of workflow models. *DEXA '98: Proc. of the 9th Intl. Workshop on Database and Expert Systems Appln.*, page 745.

Marjanovic, O. and Orlowska, M. E. (1999). On modeling and verification of temporal constraints in production workflows. *Knowledge and Info. Sys.*, 1(2):157–192.

Quinlan, J. R. (1993). *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Rinderle, S., Reichert, M., and Dadam, P. (2004). On dealing with structural conflicts between process type and instance changes. *Business Process Management*, pages 274–289.

Rinderle, S., Weber, B., Reichert, M., and Wild, W. (2005). Integrating process learning and process evolution - a semantics based approach. *Business Process Management*, pages 252–267.

Sadiq, S., Orlowska, M., Sadiq, W., and Foulger, C. (2004). Data flow and validation in workflow modelling. *CRPIT '04: Proc. of the 15ht conf. on Australasian database*, pages 207–214.

Sadiq, S. W., Orlowska, M. E., Lin, J. Y.-C., and Sadiq, W. (2005a). Quality of service in flexible workflows through process constraints. *ICEIS (3)*, pages 29–37.

Sadiq, S. W., Orlowska, M. E., and Sadiq, W. (2005b). Specification and validation of process constraints for flexible workflows. *Inf. Syst.*, 30(5):349–378.

Sadiq, W. and Orlowska, M. E. (2000). Analyzing process models using graph reduction techniques. *Inf. Syst.*, 25(2):117–134.

van der Aalst, W. M. P. and Basten, T. (2002). Inheritance of workflows: An approach to tackling problems related to change. *Theoretical Computer Science*, 270(1-2):125–203.

van der Aalst, W. M. P., van Dongena, B. F., Herbst, J., Marustera, L., Schimm, G., and Weijters, A. J. M. M. (2003). Workflow mining: A survey of issues and approaches. *Data and Knowledge Engineering, Volume 47, Issue 2 , November 2003*, pages 237–267. InternalNote: Submitted by: hr.

van der Aalst, W. M. P. and van Hee, K. M. (1996). Business process redesign: A petri-net-based approach. *Computers in Industry*, 29(1-2):15–26.

Weijters, A. J. M. M. and van der Aalst, W. M. P. (2002). Rediscovering workflow models from event-based data. *Proceedings of the Third International NAISO Symposium on Engineering of Intelligent Systems (EIS 2002)*, pages 65–65.