PROCESS MINING FOR JOB NETS IN INTEGRATED COMPLEX COMPUTER SYSTEMS

Shinji Kikuchi, Yasuhide Matsumoto, Motomitsu Adachi
Fujitsu Laboratories Limited, Japan

Shingo Moritomo
Fujitsu Limited, Japan

Keywords: Process mining, Batch job, Job net, Integrated system, Behavior analysis.

Abstract: Batch jobs, such as shell scripts, programs and command lines, are used to process large amounts of data in large scale enterprise systems, such as supply chain management (SCM) systems. These batch jobs are connected and cascaded via certain signals or files so as to process various kinds of data in the proper order. Such connected batch jobs are called “job nets”. In many cases, it is difficult to understand the execution order of batch jobs in a job net because of the complexity of their relationships or because of lack of information. However, without understanding the behavior of batch jobs, we cannot achieve reliable system management. In this paper, we propose a method to derive a job net model representing the execution order of the job net from its logs (execution results) by using a process mining technique. Improving on the Heuristic Miner algorithm, we developed an analysis method which takes into account the concurrency of batch job executions in large scale systems. We evaluated our analysis method by a conformance check method using actual job net logs obtained from a large scale SCM system. The results show that our approach can accurately and appropriately estimate the execution order of jobs in a job net.

1 INTRODUCTION

There are many cases where enterprise information systems are constructed not by developing them “from scratch”, but by connecting a large number of smaller systems. For example, many supply chain management (SCM) systems have been built by interconnecting individual systems processing different data such as production, sales & marketing, and logistics. These individual systems have usually been built at different times by different vendors based on different policies. Therefore, integrating and interconnecting different types of system can result in a more complex system than one that has been purpose built from the beginning. As a result, it becomes very difficult to manage these integrated systems such that their subsystems can not only process their own data propery but also work consistently with the other subsystems.

In this kind of integrated system, batch processes called “jobs”, such as batch files, shell scripts and commands, play important roles. These jobs are executed in order to handle large amounts of data, such as accounting or inventory checks, in contrast with transaction processes which handle each request from users as soon as it arrives. These jobs can be scheduled and invoked by job net management functions such as SystemWalker Operation Manager (Fujitsu, 2008) which control jobs and invoke them at a certain time such as overnight or at the end of the month. After a job finishes, it can invoke another job and hand over its processing results via files or signals output from the previous job. By invoking a job from another job runnning on a different server or subsystem, we can choreograph some subsystems to process their common data in the proper order, as described in Figure 1. Therefore, we can say that these batch jobs play important roles in bridging the gap between subsystems and connecting them so that the whole system can process data properly. We call a set of batch jobs concatenated and executed in a defined order a “job net”.

Kikuchi S., Matsumoto Y., Adachi M. and Moritomo S. (2010).
PROCESS MINING FOR JOB NETS IN INTEGRATED COMPLEX COMPUTER SYSTEMS.
In *Proceedings of the 12th International Conference on Enterprise Information Systems - Information Systems Analysis and Specification*, pages 5-12
DOI: 10.5220/0002853500050012

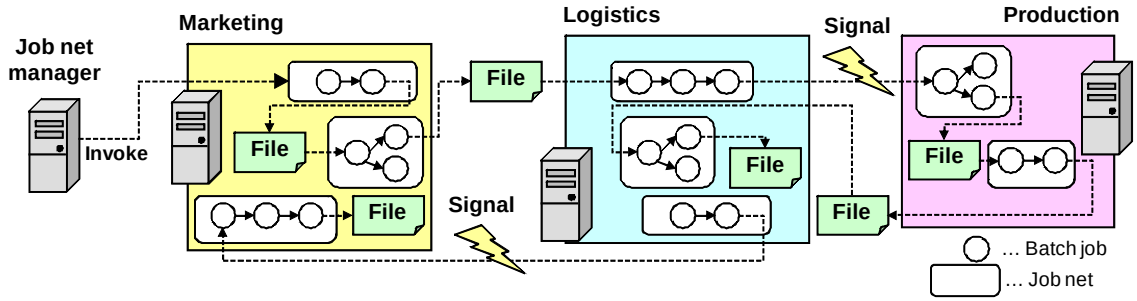

Figure 1: Job nets connecting different systems via files and signals.

It is, however, extremely difficult to understand the behavior and the execution orders of the jobs in these kinds of "tangled" job nets, because the clues to solving the problem are scattered everywhere. For example, even if job scheduling information is stored in several job net manager systems for the invocation of jobs, they might be managed by the administrator of a subsystem or by each individual department. Because of this "silo" management, access to this kind of information from outside the department might be prohibited. In addition, in many cases, the information regarding the triggers (files and signals) invoking the jobs is embedded in the job's script or the program itself. Deriving the information regarding the triggers from program code analysis is practically impossible. For these reasons, it is difficult to understand the behavior of interrelated batch jobs. This problem can worsen in the case of the integration of larger systems such as M&A. However, without understanding the behavior of job nets, we cannot achieve reliable service management, such as predicting the finishing time of jobs or determining which job was the root cause when the execution of jobs are delayed. Therefore, there is a strong need for a technique for understanding the behavior of job nets.

Against this background, we developed an analysis method to derive a model of job nets representing their execution order from the job net log recording their execution results by using a process mining technique. In this method, we improve the Heuristic Miner process mining algorithm by taking into account the concurrent execution of jobs. We then applied our method to job net logs derived from an actual SCM system and evaluated the accuracy of our approach by a conformance check method.

The rest of this paper is organized as follows. First, in Section 2, we survey related work. Next, Section 3 explains our job net mining algorithm in detail. We then show how it works through a case study in Section 4 using an actual set of log data and evaluate its performance. Following this, Section 5 concludes the paper and outlines future challenges.

2 RELATED WORK

One of the most important major techniques for deriving the behavioral characteristics of systems is the process mining approach (van der Aalst, 2007). Process mining is a method of extracting the information about a process from its execution results (event logs) in order to construct a process model that can represent the behavior of systems or processes. The process model can be represented by some state transition systems such as the Markov model or Petri Net. Various algorithms for process mining have been proposed so far, such as the alpha-algorithm (van der Aalst, 2004) and genetic algorithm (van der Aalst, 2005). These algorithms are intended for application to the analysis of business processes usually executed by human beings and consisting of less than a dozen events. The computational time for these algorithms therefore tends to increase rapidly with the number of events per process. While this does not matter when the process consists of only a small number of events, we can not apply these methods directly for job net analysis since the job nets in large scale systems can consist of hundreds of jobs.

Computational time for the Heuristics Miner algorithm (Weijters, 2006) is relatively small because of its simplicity and straightforwardness. It is, however, possible that this simple algorithm cannot achieve sufficient accuracy in job net analysis for large systems where we have to take into account the possibility that many jobs are executed concurrently. There is therefore a strong need for an algorithm that is specialized for job net mining so as to achieve both short computational time and sufficient accuracy.

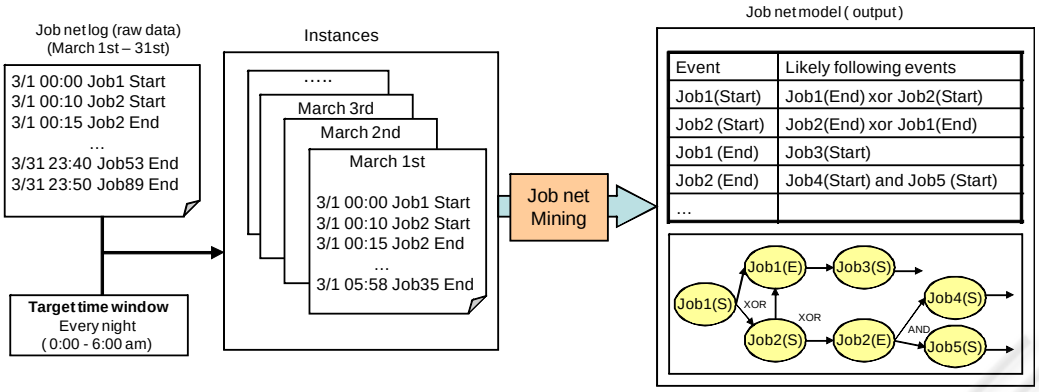

Figure 2: Input and output data for job net mining.

3 JOB NET MINING METHOD

In this section we explain our job net mining method in detail. First, we define its data structure. Next, we explain our mining algorithm based on the Heuristic Miner algorithm with some improvements for taking concurrency in batch job execution into account. Then, we explain how the accuracy of our mining method can be evaluated through a conformance check approach.

3.1 Data Structure

Figure 2 summarizes the input and output data for our approach. As explained in Section 1, in many cases we cannot obtain or determine the location of the information defining the schedules or relationships of the batch jobs. Therefore, we assume here that we can obtain only the job net event logs which are output as the execution results of these jobs. This kind of log is relatively easy to obtain, since it is usually created so that the administrators of job nets can diagnose their behavior after a problem has occurred. We also assume that the start time and end time of each job is recorded in the job net logs. For simplicity, we assume here that the granularity of the timestamp is 1 second and each job is executed no more than once per day. In our analysis, we define the time window (e.g. overnight, from 0:00 am to 6:00 am) on which we focus attention. Then we extract the data within the time window to be used for our analysis. We refer to the sequence of log data for a job net executed in the time window on a particular day as an instance of the day.

The output from our method is a job net model representing the common patterns of orders of events emerging in many instances. Here we assume that each event is either the beginning or the finishing of a job recorded in the logs. The model contains order relations between each preceding event and a set of (likely) following events. It can be represented by tables or directed graphs as shown in the right hand part of Figure 2. If a preceding event has more than two possible following events, we should determine those branches as either an AND-fork or an XOR-fork. The AND-fork means that all of the following events will occur after the preceding event, while the XOR-fork means that only one of the following events will occur after the preceding event.

3.2 Mining Algorithm

Since a large number of batch jobs may be executed simultaneously in large scale systems consisting of many servers, our analysis has to take the concurrency in job net mining into account in order that sufficient accuracy is achieved. We therefore developed an algorithm consisting of the following three steps. First, we determine the set of jobs which are likely to start at the same time from timestamps recorded in the log. Next, we derive the order of events using the Heuristic Miner algorithm. Finally, we modify the Heuristic Miner results using the information regarding concurrent jobs derived in the first step. The details of these steps are as follows.

Step 1: Concurrent Job Detection from Timestamp

In the first step, we determine the set of jobs which start at almost the same time for reasons such as the preceding job triggering several following jobs, or jobs happening to be scheduled to start at the same time by different administrators. We use the

following evaluation functions to determine whether jobs J_i and J_k are likely to start at the same time.

$$p(J_i, J_k) \equiv \frac{N(|\, S(J_k) - S(J_i)\,| < \tau_p)}{N(J_i)} \quad (1)$$

$N(J_i)$ represents the number of instances including execution of job J_i. $N(|\, S(J_i) - S(J_k)\,| < \tau_p)$ represents the number of instances in which the difference between the start times of J_i and J_k is smaller than the threshold τ_p sec. We can say that J_i and J_k tend to start at the same time if $p(J_i, J_k)$ is close to 1.

Using equation (1), we define the set $c(J_i)$ of jobs which are likely to start at the same time as job J_i.

$$c(J_i) \equiv \{J_k \mid p(J_i, J_k) > \tau_c\} \quad (2)$$

This means that if $p(J_i, J_k)$ is larger than τ_c, J_k is included in $c(J_i)$.

Step 2: Event order Analysis by Heuristics Miner

Heuristics Miner (Weijters, 2006) is a process mining algorithm which derives patterns in the order of events from event logs independent of the events' timestamps. This method determines the existence of consecutive order relations between events using the following function $e_i \Rightarrow_W e_k$.

$$e_i \Rightarrow_W e_k \equiv \left(\frac{|\, e_i >_W e_k\,| - |\, e_k >_W e_i\,|}{|\, e_i >_W e_k\,| + |\, e_k >_W e_i\,| + 1} \right) \quad (3)$$

Function $|\, e_i >_W e_k\,|$ represents a count of the instances in which event e_i's next event was e_k. Here, we take into account only the order of events, independent of timestamps. We consider that there is an order relation between event e_i and e_k when the function $e_i \Rightarrow_W e_k$ is over a given threshold. In our analysis, we adopt an all-activities-connected-heuristic that derives at least one preceding event for each event. Here, we define two thresholds: (1) Dependency threshold τ_D and (2) Relative to best threshold τ_R. If $(e_i \Rightarrow_W e_k) > \tau_D$, we conclude that there is an order relation between event e_i and e_k. If event e_k does not have any preceding event e_i such that $(e_i \Rightarrow e_k) > \tau_D$, we select an event e_x such that $(e_x \Rightarrow_W e_k) \geq (e_y \Rightarrow_W e_k)$ for any other event e_y. We then consider that there is an order relation between event e_i and e_k if $(e_i \Rightarrow_W e_k) \geq (e_x \Rightarrow_W e_k) - \tau_R$.

Next, we use the following function $e_i \Rightarrow_W e_j \wedge e_k$ to determine whether the order relations $e_i \Rightarrow_W e_k$ and $e_i \Rightarrow_W e_j$ from the same event e_i represent an AND-branch or XOR-branch.

$$e_i \Rightarrow_W e_j \wedge e_k \equiv \left(\frac{|\, e_j >_W e_k\,| + |\, e_k >_W e_j\,|}{|\, e_i >_W e_j\,| + |\, e_i >_W e_k\,| + 1} \right) \quad (4)$$

If the value of this function is larger than threshold τ_A, we assume that the two relations are AND-branches, meaning that both following events will eventually occur after the preceding event e_i. Otherwise, we conclude that they are XOR-branches, i.e. that only one of the following events will occur after the preceding event.

In our analysis, we assume each event to be either the start event or end event of a job. In the remainder of this paper, we denote the job J_i's start event and end event by e_i^S and e_i^E respectively.

Step 3: Adjustment for Concurrency

After determining the sets of concurrent jobs in Step 1 and the jobs' order relations in Step 2, we adjust the results of the latter by those of the former's in Step 3.

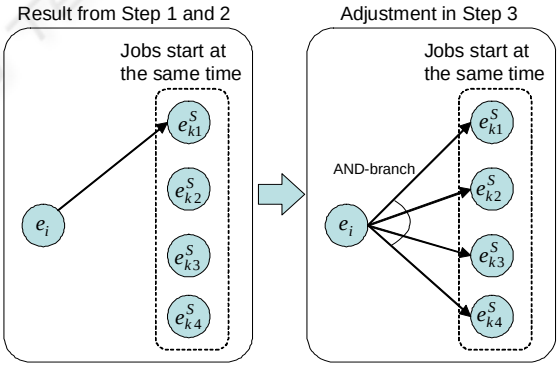

Figure 3: Adjustment in Step 3.

Figure 3 shows the general concept of the adjustment. Here we suppose that Step 1 determined that the set of jobs J_{k1}, J_{k2}, J_{k3} and J_{k4} start at the same time. The corresponding start events of these jobs are represented in the dotted rectangle by e_{k1}^S, e_{k2}^S, e_{k3}^S, and e_{k4}^S respectively. We also suppose that the order relation from a preceding

event e_i to the start event of some of these jobs (e.g. J_{k1}) is determined by Step 2, as shown by the arrow in the left hand part of Figure 3. In such cases, while any jobs in the set of following jobs (J_{k1}, J_{k2}, J_{k3} and J_{k4}) can start after the preceding event e_i, the relations between event e_i and the following events other than e_{j1} are not correctly detected by the Heuristics Miner algorithm. It is difficult for Heuristics Miner to correctly determine such concurrencies because the occurrence of several events at almost the same time can be recorded in their logs in random order.

In order to solve this problem, we adjusted the model derived in Step 2 using the results of Step 1 as follows.

(1) Select a relation $e_i \Rightarrow_W e_k^S$ and a set of jobs $c(J_k)$ which start at the same time as the start event e_k^S of job J_k.
(2) Establish the order relations from the preceding event e_i to the start event of the jobs in $c(J_k)$.
(3) Designate the relations thus established as AND-branches

The result of this adjustment can be seen on the right hand side of Figure 3. By performing this adjustment in our model construction, we can take into account the concurrent job information which may be overlooked by the Heuristic Miner algorithm.

3.3 Conformance Check

In order to evaluate the accuracy of our mining algorithm, described in Section 3.2, we use a conformance check (Rozinat 2005, 2008) which evaluates how well process models derived by a process mining algorithm express the patterns emerging in event logs by "replaying" the instances of the logs on the obtained models and detecting inconsistencies between the model and the logs.

The general concept of the conformance check is shown in Figure 4. First, we prepare a process model derived from a process mining algorithm. We also prepare instances of logs for evaluation of their conformance with the model. Next, we replay on the model, one by one, the events recorded in the instances. In this replay, we predict the candidates for the next events following each preceding event by referring the process model. For example, in the

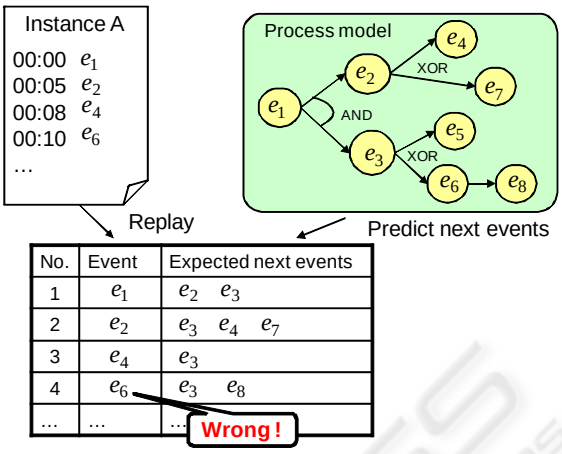

Figure 4: Conformance Check.

case shown in Figure 4, after the first event e_1 occurs in instance A, we predict that the next event will be either e_2 or e_3, because these events are the following events for e_1 in the process model. Likewise, after the second event e_2 occurs, we predict one of the events e_3, e_4, or e_7 will be the third event. Here event e_3 still remains as one of the expected next events since the links $e_1 \rightarrow e_2$ and $e_1 \rightarrow e_3$ are AND-branches meaning that both e_2 and e_3 can occur after the preceding event e_1.

Next, we check whether or not each prediction is correct. We conclude that the model conforms to the instance if the *i*-th event recorded in the instance is included in the *(i-1)*-th expected next events predicted by the model. In Figure 4, while the first three events (e_1, e_2 and e_4) are predicted correctly, the occurrence of the fourth event e_6 is not predicted by the model, because it is not included in the third set of expected next events. If the number of such events, those not expected by model, is small, we can conclude that the model fits well with the given instance. This fitness can be evaluated by the following "fitness" function which is simplified from the original functions (Rozinat 2005, 2008) so that it suits the conditions in our job net analysis.

$$f = 1 - \frac{\sum_{i=1}^{k} m_i}{\sum_{i=1}^{k} n_i} \quad (5)$$

In this fitness function, k represents the number of instances used for the evaluation, n_i is the number of events recorded in the i-th instance and m_i is the number of events which are not predicted correctly

by the given process model. A value of the function close to 1 indicates that the model fits well with the given instances.

All the same, if we include all the events of the instance in the set of expected next events, we can always achieve a high value for the fitness function. This, however, would be meaningless because it does not narrow down the set of possible next events. Therefore, the smaller number of expected next events derived from a model, the closer that model appropriately represents the structure of the process, and the more valuable it is. To evaluate this characteristic, we use the following "appropriateness" function, which has also been tailored to our purpose.

$$a = \frac{\sum_{i=1}^{k} n_i (M - x_i)}{(M-1) \cdot \sum_{i=1}^{k} n_i} \quad (6)$$

M is the number of events emerging in the model and x_i represents the average number of expected next events in the replay of the i-th instance. If the model can always narrow down the expected next events to just one event, the value of the appropriateness function is 1.

When checking the conformance of the model with the instances, we evaluate both the fitness and appropriateness functions.

4 EXPERIMENT

4.1 Setup

We evaluated our approach using the following setup. First, we collected job net log data from an actual SCM system. This system was created by interconnecting 18 servers fulfilling different roles such as marketing, production management, and logistics. Of these 18 servers, we picked out the data recorded in the five main servers, on which many of the job nets are executed. For evaluation, we prepared the two sets of data specified in Table 1: Log A is data obtained overnight on weekdays in June and Log B is data obtained for the same days and times in July. Each job's start/end timestamp is recorded in the data. In order to evaluate whether our approach is able to predict the order of job executions correctly, we constructed the job net model from Log A and separately checked its conformance with Log A and with Log B. In addition, in order to evaluate the effectiveness of our mining algorithm, we compared the results of our approach (using all of the steps 1, 2 and 3 in Section 3.2) with the Heuristic Miner Algorithm (using Step 2 only). For the thresholds, we used $\tau_p = 1$ (sec), $\tau_c = 0.5$, $\tau_D = 0.8$, $\tau_R = 0.1$, and $\tau_A = 0.1$.

Table 1: Log data used for experiments.

	Duration	Number of days	Time window	Number of jobs (avg.)
Log A	2009 June 1st – 30th (Weekday only)	21	00:00am – 06:00am	1018
Log B	2009 July 1st – 31st (Weekday only)	23	00:00am – 06:00am	1027

4.2 Results

We implemented our algorithm in Java and executed the experiments described in the previous subsection by using a PC with Windows XP Professional Edition, 4.3GHz CPU, and 1GB memory. The job net mining task in each experiment finished within 10 minutes. Since 3,356 individual jobs were recorded in Log A, the number of events (job start and end events) in the job net models constructed in each experiment was 6712. Figure 5 shows a part of the derived model drawn by Graphviz (Gansner, 2000) with the arrow attributes (AND or XOR) omitted for simplicity.

Table 2 summarizes the results of the experiments. Comparing the numbers of unexpected events in Heuristic Miner (Case 1 and 2) with the numbers in our approach (Case 3 and 4), it can be seen that the

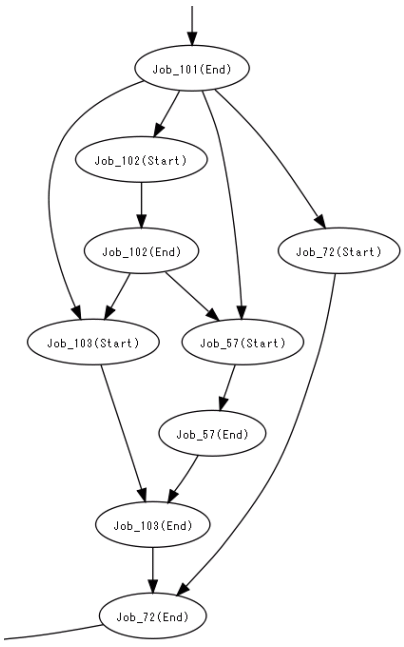

Figure 5: Job net model (part).

Table 2: Experimental results.

Case	Algorithm	Data for model	Data for check	Number of events (avg.)	Number of unexpected events (avg.)	Number of expected next events (avg.)	Fitness	Appropriateness
1	Heuristic Miner	Log A (June)	Log A (June)	2035.2	246.4	64.3	**0.879**	**0.991**
2	Heuristic Miner	Log A (June)	Log B (July)	2054.7	273.5	64.8	**0.867**	**0.991**
3	Proposed algorithm	Log A (June)	Log A (June)	2035.2	93.7	75.2	**0.954**	**0.989**
4	Proposed algorithm	Log A (June)	Log B (July)	2054.7	122.1	76.4	**0.941**	**0.989**

latter are much smaller than the former. This results in a higher value of fitness parameter for our approach than for Heuristic Miner. Furthermore, the numbers of expected next events and the appropriateness values in both algorithms are almost the same. Therefore, we can conclude that a more precise model can be constructed through our approach than through the Heuristic Miner algorithm alone, without having much impact on the appropriateness parameters.

In addition, the difference between the results produced by the same algorithm (Case 3 and 4) is quite small. Therefore, we can conclude that our algorithm is able to predict the behavior of the job nets in July using the model constructed from the logs recorded in June with the same precision as in the case where the log data used for model construction and for conformance checking are the same.

5 CONCLUSIONS

We proposed a job net mining method to derive the execution order of job nets from their logs. In this method, we identify the set of jobs executed at the same time. Using this information, we then modify the job net model derived by the Heuristic Miner algorithm. Through conformance checking using the log data of job nets executed in an actual SCM system, we confirmed that our method enables construction of a job net model that represents the order relations between jobs more accurately and appropriately than that obtained through Heuristics Miner alone.

We are now considering the following work for the future. First, we plan to develop methods for the concise visualization of the structure and characteristics of job nets. Since it is difficult for system administrators (humans) to understand the relationships between over 1000 events in a single directed graph, we need a method of extracting the important part of the model or abstracting its structure in order to make it understandable.

Next, using the proposed approach, we plan to develop a method of predicting the finishing times of job nets. Since one of the biggest concerns many administrators of job nets have is whether or not the job nets will finish within the deadline, this function will be able to help them manage their job nets more efficiently.

Finally, we plan to develop a method for analyzing the model derived by our approach. For example, when failures or delays occur in job net execution, the job representing the root cause can be detected by backtracking through the order relations in the derived model. In addition, by measuring the execution durations of jobs, the critical path, taking a large amount of time to finish, can be detected. This information is useful for reorganizing job nets so as to reduce their execution times. By these analysis techniques, we will be able to improve reliability in the management of large scale integrated complex computer systems.

ACKNOWLEDGEMENTS

We would like to thank Masaru Ito for his help in collecting job net data and for giving us much useful advice.

REFERENCES

Fujitsu, 2008, SystemWalker Operation Manager v13.3, http://www.fujitsu.com/global/services/software/systemwalker/products/operationmgr/

Van der Aalst, W. M. P., Reijers, H. A., Weijters, A. J. M. M., van Dongen, B. F., Alves de Medeiros, A. K., Song, M., and H. M. W. Verbeek, 2007, Business Process Mining: An Industrial Application, *Information Systems*, 32(5):713-732.

Van der Aalst, W. M. P., Weijters, A. J. M. M., and Maruster, L., 2004, Workflow Mining: Discovering Process Models from Event Logs, *IEEE Transactions on Knowledge and Data Engineering*, Vol.16, No.9.

Van der Aalst, W. M. P., Alves de Medeiros, A. K., Weijters, A. J. M. M., 2005, Genetic process mining, *Proceedings of the 26th international conference on*

applications and theory of Petri nets. Lecture notes in computer science, Vol. 3536. Springer.

Wen L., van der Aalst, W. M. P., Wang, J. and Sun, J., 2007, Mining process models with non-free-choice constructs, *Data Mining and Knowledge Discovery*, 15(2):145-180.

Weijters, A. J. M. M., van der Aalst, W. M. P., and Alves de Medeiros, A. K., 2006, Process Mining with the Heuristics Miner-algorithm, *BETA Working Paper Series*, WP 166, Eindhoven University of Technology.

Rozinat, A., and van der Aalst, W. M. P., 2008, Conformance Checking of Processes Based on Monitoring Real Behavior, *Information Systems*, Vol.33, No.1, pp.64-95.

Rozinat, A., and van der Aalst, W. M. P., 2005, Conformance Testing: Measuring the Fit and Appropriateness of Event Logs and Process Models, *Proceedings of First International Workshop on Business Process Intelligence (BPI'05)*, pp.1-12.

Gansner, E., North, S., 2000, An open graph visualization system and its applications to software engineering, *Software – Practice & Experience*, Vol.30, No.11, pp.1203-1233.