Business Monitoring Framework for Process Discovery with Real-Life Logs

Mari Abe and Michiharu Kudo

IBM Research – Tokyo
5-6-52 Toyosu, Koto-ku, Tokyo, Japan
{maria, kudo}@jp.ibm.com

Abstract. Business analysis with processes extracted from real-life system logs has recently become important for improving business performance. Since business users desire to see the current situations of business with visualized process models from various perspective, we need an analysis platform that supports changes of viewpoint. We have developed a runtime monitoring framework for log analysis. Our framework can simultaneously extract process instances and derive appropriate metrics in a single pass through the logs. We tested our proposed framework with a real-life system log. The results for twenty days of data show synthesized process models along with an analysis axis. They were synthesized from the metric-annotated process instances generated by our framework.

1 Introduction

Process mining plays an important role in the business analysis of real-life logs generated from enterprise applications. The actual situation of an enterprise is visualized and analyzed based on extracted workflows to detect best practices. When consulting on the actual business situations of enterprises, analysis methods that include defining metrics for the analysis axes, extracting workflows based on the metrics, and analyzing them is practically a norm.

Our previous work describes business process analysis and associated client engagements [1, 2] where we presented a technical approach with practical application scenarios. The most iterative and time-consuming work was the determination of metrics to discover processes that customers recognize the real processes of their business. Then they desire to see the discovered processes from different view points of the metrics as new analysis axes. For example, after customers recognize the processes of their business, they desire to compare the weekday processes with the weekend processes to find the causes of delays. Another example is customers want to compare purchasing processes for various product types with a duration within 10 minutes against over 10 minutes for the marketing strategies.

To change analysis axes according to the changes of the customer's requirements, we need to analyze the logs to extract the workflows again. It is an issue on the analysis method with process discovery that the analysis cycle is time-consuming and it causes belated results since a module of reading log and

extracting workflow must be reimplemented according to the metrics changed. This implies a need for a process discovery technology that can quickly extract workflows while reducing the time spent reading the logs so that the extracted results can be used even as the requirements of the customers are changing.

In this paper, we introduce a monitoring framework for process discovery. It simultaneously extracts the process instances and metrics of a single pass through the logs. We define the abstraction level of the monitoring context based on inclusion relationships of the correlation key definitions and present the monitoring algorithm based on the abstraction level. Instances of monitoring contexts are linked at runtime and that allows us to build process models. We tested our proposed approach with real-life logs and showed the process models that were synthesized with different values of an analysis axis.

2 Related Work

There are many proposed approaches for data warehousing that can analyze process logs [3, 4]. Aalst [5] proposes "process cubes" that enable analyzing process logs by storing the results into data cubes that allow domain experts to execute online analytical processing (OLAP) operations such as drill-down, roll-up, and other operations for understanding the executed processes. We deal with real-life event logs from existing applications, so we are not limited to logs that are generated from particular workflow engines. Therefore, each event that we analyze is not always directly linked to a particular activity. An example of logs includes access history of external pages outside of the business flow. These logs can still be using as important as process logs if the domain experts want to see the causes of business performance degradation. Our advantage over these approaches is we expand the universe of inputs of the event base to event logs of legacy applications by leveraging our reverse engineering approach with business monitoring technology.

Liu et al. [6] propose complex event processing systems that support OLAP operations for multi-dimensional pattern analysis over event streams. Their solution improves computational efficiency for multi-dimensional event patterns by sharing the results among queries using a unified query plan. The difference in their approach is pattern-based event filtering versus our model-based filtering for event streams. However, optimization methods should also be beneficial for our method for efficient handling of events in real time.

Schiefer et al. [7] propose a solution for managing the performance data of business processes. Their system is intended for process-driven decision support and continuously improving business processes. It is important to drive decision support by analyzing both the process logs and other logs linked to processes.

The concepts and architectures of real-time business monitoring have been proposed for and applied to real customers [8, 9]. The framework itself is a generic approach for an enterprise to improve its capabilities of sensing and responding to business situations. In our previous work [10], a model-driven mechanism of creating business monitoring applications was proposed. However, there are

difficulties with such a purely top-down approach to build applications for enterprises that already have legacy applications for process management. We are now working on a novel approach for enhancing monitoring applications for agile business analysis with process discovery that can lead to a process model reflecting the actual behavior of users as recorded in the system logs. This reengineering approach helps manage the lifecycle of business processes from the top-down to bottom-up and bridges the gap between business monitoring and process discovery.

3 Process Discovery from Real-life Logs

line #	Event type	User ID	Timestamp	Host	Request URL	Parameters
1	request	user1	2013-08-20T08:30:33	sample.server.com	calc_premium	
2	request	user2	2013-08-20T08:30:33	sample.server.com	logon	
3	response	user1	2013-08-20T08:30:34	sample.server.com		pageid="calc_premium_page"...
4	request	user3	2013-08-20T08:30:35	sample.server.com	logon	
5	response	user2	2013-08-20T08:30:35	sample.server.com		pageid="logon_page"...
6	request	user1	2013-08-20T08:30:48	sample.server.com	submit_condition	sourcepageid="calc_premium_page", birthday="1965/01/01", gender="male"
7	response	user1	2013-08-20T08:30:49	sample.server.com		pageid="insurance_option_page", msg="input options next"
8	response	user3	2013-08-20T08:30:50	sample.server.com		pageid="logon_page"...
9	request	user1	2013-08-20T08:31:50	sample.server.com	submit_product	sourcepageid="insurance_option_page, product="life insurance", premium="$50", period="10 years"
10	response	user1	2013-08-20T08:31:51	sample.server.com		pageid="simulation_result_page", msg="need physical check-up"
11	request	user1	2013_12-20T08:32:00	sample.server.com	save	sourcepageid="simulation_result_page"

Fig. 1. Example of input logs for insurance application

Most real-life system logs are generated for system diagnoses including problem determination when there are abnormal situations such as server failures. The logs were not originally designed for reuse to create any secondary value. The characteristics of such logs make process discovery difficult to apply to real-life systems and our prior work [1, 2, 11] has attempted to address these problems. Fig. 1 is an extract from an example log for an insurance application that does premium calculations. Before discovering a process in the log, we must determine which parameters are correlated so we can extract tasks (activities) and process instances. In this example, a pair of parameters "pageid" for transaction type "response" and "sourcepageid" for "request" can be correlated to associate with the task. The column "user ID" can be used as a correlation key to extract a process. The semantics of these logs should be given by domain experts for our system of process discovery before the analysis.

To determine the metrics, we should select a column of the logs or some parameters that will eventually become our metrics or sources of metrics. In this example, the duration of each task can be derived using subtraction with

the timestamps of the response and request correlated by "pageid" and "sourcepageid" e.g. the duration is 14 seconds for "calc_premium_page" as calculated from the timestamps of Lines 3 and 6. Lines 1, 2, 4, and 5 are ignored if the users focus on the process of premium calculations. Similar metrics can be derived from other parameters. For example, "birthday" and "gender" are input to the insurance application on the "calc_premium_page" (Line 6) and "msg" is output to the "insurance_option_page" (Line 7). Process execution flow of the example is shown in Fig 2. The duration for each page is the metric for each task (14, 61, and 9 seconds). The input data and output data are also metrics for each task.

Monitoring business metrics in real time requires an event monitor runtime and event subscriber called a *monitoring context* [8–10, 12]. We propose monitoring contexts and a runtime that allow extracting metrics and process instances simultaneously in a single pass through the logs described in the following subsections. Our approach offers two improvements over the existing approaches: (1) Associations of parent-child of the instances of the monitoring contexts are determined dynamically based on inclusion relationships of correlation key definitions and (2) The lifecycle of the instances of the monitoring contexts of the parents and children can be handled independently

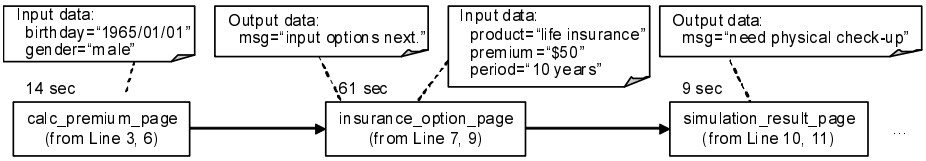

Fig. 2. A process instance for the insurance application of "user1"

3.1 Definition of Monitoring Context

We define "monitoring context" here for monitoring the events and calculating the metrics. Let E be an event sequence $\{e_1, \cdots e_n\}$. An event e is a tuple $e = \langle type_e, A \rangle$ where $type_e$ is a type of the event and A is a set of *attributes* for the event. An *attribute* is a tuple $a = \langle type_a, name, value \rangle$ where $type_a$ is a type of the attribute, *name* is a name of the attribute and *value* is its value.

Let X be a set of *variables*, including array variables. Let F be a set of *functions* for computing the values of X. Each value of variable x is derived from a function $f : t_1 \times t_2 \cdots \times t_n \rightarrow type_x$, where $type_x$ is a type for x. A *metric* is a specific *variable* that domain experts define with f and that will be used to query a process model in our method. A monitoring context mc is a tuple $mc = \langle E_i, K, E_o, C_e, X, F \rangle$ and an instance of mc is identified by a unique identifier. Each element of mc is defined as follows:

- E_i is a set of types of events occurred in E that can be monitored (inbound events). E_i does not necessarily include all of the types of events occurring in E. The *attributes* of an event are mapped to X with F.
- K is a set of variables derived from a set of mappings of E_i to uniquely identify an instance of mc (correlation keys). If there is no mc instance correlated with an inbound event and the event can be mapped to K, then an instance of mc is created and starts monitoring.
- E_o is a type for an event that can be generated from mc (outbound events). A generated event can be an input of other mcs.
- C_e is a set of conditions c that determines whether or not monitoring should be terminated. Each c is a specific f where the type of the variable derived from f is boolean. If all of the cs of an mc are set to $true$, then an instance of E_o is emitted to other mcs and monitoring is terminated.

We define the *abstraction level of the monitoring contexts* based on the relationships of the correlation keys. The mc that has K is denoted by mc_K. *An abstraction level of a monitoring context* $L(mc)$ is a positive integer and satisfies $L(mc_{K_0}) > L(mc_{K_1})$ if $K_1 \supset K_0$. It is a necessary condition for a monitoring context to generate hierarchical structures. Consider a process instance for the insurance application of "user1" (Fig. 2). The correlation key definitions of this task are "pageID" and "userID" while that of the process-instance is "userID". A set of correlation key definitions of task K_t and of process instance K_{pi} satisfy $K_t \supset K_{pi}$ where $userID, pageID \in K_t$, and $pageID \in K_{pi}$. These constraints also imply $L(mc_{K_t}) < L(mc_{K_{pi}})$.

3.2 Algorithm of Monitoring Framework

A monitoring context manager mgr manages the lifecycle of the instances of mc. It has a list of mcs for each abstraction level and knows how to serialize the mc instances when the instances terminate. The function "PROCESSEVENT" of mgr is the main flow of the event processing. We show the algorithm of "PROCESSEVENT" in Algorithm 1, which simply calls the "DOCORRELATION" and "DOEVENTPROCESSING" functions.

The mgr gets the correlating mc instance list by calling a function "GETCORRELATINGMC" in Line 6. In Lines 16 to 25 if this function, mgr gets the list of instances of mcs that is a higher level of abstraction than the event source. If the event source is a component for reading logs, then the list includes the lowest level of the instances of mcs, such as the instances of mc for mining tasks. If there are no instances of mcs listed, then mgr tries to instantiate mc from the event. A newly instantiated mc is registered in the mgr based on its abstraction level (from Lines 7 to 15). The next mgr has to do is to process the event. In the function DOEVENTPROCESSING from Lines 26 to 30, mgr simply calls processEvent of mc in the correlated list to update the variables of an instance of mc. If all of the variables of the instance of mc are set and the terminal condition C_e is true, then the instance emits an outbound event that includes the metrics. Then the instance recursively calls PROCESSEVENT of mgr.

Algorithm 1 event processing flow of *mgr*

```
1: function processEvent(e)                                   ▷ e is an object of BusinessEvent
2:     correlating_list ← doCorrelation(e)
3:     doEventProcessing(correlationg_list, e)
4: end function
5: function doCorrelation(e)
6:     list ←getCorrelatingMC(e)
7:     if list = empty then
8:         instantiate MonitoringContext m from e
9:         if m ≠ null then
10:            register m based on abstraction level
11:            list.add(m)
12:        end if
13:    end if
14:    return list
15: end function
16: function getCorrelatingMC(e)
17:    m0 ← e.getEventSource()
18:    list0 ← getList(m0)                    ▷ get monitoring contexts by abstraction level of m0
19:    for m in list0 do
20:        if m.correlate(e) then
21:            list.add(m)
22:        end if
23:    end for
24:    return list
25: end function
26: function doEventProcessing(correlating_list, e)
27:    for m in correlating_list do
28:        m.processEvent(e)
29:    end for
30: end function
```

4 Experiment with Real-life Logs

Table 1. Summary of the test data of the real-life logs and the results of the experiment

Test data				Result of experiment	
Period of logs	**# of lines of logs**	**# of task metrics**	**# of proc_inst metrics**	**# of instances of *mc* for task**	**# of instances of *mc* for proc_inst**
20 days	685,318	4	15	260,568	25,781

We tested our proposed framework on real-life system logs and verified the usefulness of the proposed approach. Table 1 shows some statistics for our experiment. We tested the logs from 20 successive days of an application server. The number of lines in the logs was 685,318.

The metrics for each task included four metrics, the task durations, the names of products, the numbers of help pages accessed, and the status of the forms created (either new or update). The metrics for a process_instance included fifteen metrics such as a list of pages, counts of help pages accessed, the status of process started (either new or update) derived from *mc* for task, the status of process termination (either save or cancel), and so on. The number of instances of *mc* for the task was 260,568 and the total of process_instance was 25,781. There were instances for a task that were not linked with any instances for process_instance

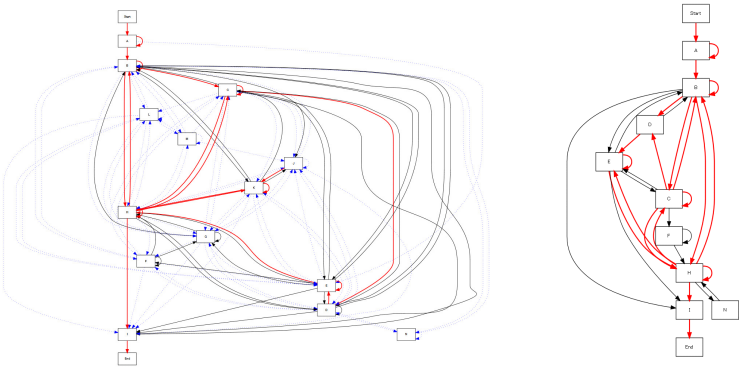

Fig. 3. The generated process models with different values of analysis axis

because they did not match the conditions for process start. Our framework serialized the results into an MXML[13] file with metric annotations and an XSLT [14] file to extract the process instances.

Fig. 3 shows synthesized process models generated from our process discovery tool [2]. Each box indicates a page access as a task and the process flow starts from top to bottom. The help page are not included in the models as tasks because it is not a part of the business flow, but they appeared in the original logs. The left side is a process model in which the number of help page access is less than three. The right side is a model in which the number of help page access equals or is more than three. The difference is that users on the left are struggling with the process while users on the right are not. This result become one of data sources for domain experts to verify whether or not the help pages are effective.

5 Conclusion

In this paper, we proposed a monitoring framework for process discovery that simultaneously extracted the process instances and metrics of a single pass through the logs. We defined the abstraction level of the monitoring context based on inclusion relationships of the correlation key definitions and presented the monitoring algorithm based on the abstraction level. Instances of monitoring contexts were linked at runtime and that allows us to build process models. With the results, users could get process models from different metrics without reading huge log again. We tested our proposed framework with a real-life system log of twenty days and the results become one of data sources for domain experts to verify whether or not their system is used effectively.

References

1. Kudo, M.: Operational Work Pattern Discovery Based On Human Behavior Analysis. In: Service Research and Innovation Institute Global Conference. (2014)
2. Kudo, M., Ishida, A., Sato, N.: Businesss Process Discovery by using Process Skeletonization. In: International Symposium on Data-Driven Process Discovery and Analysis. (2013)
3. Kueng, P., Wettstein, T., List, B.: A Holistic Process Performance Analysis Through a Performance Data Warehouse. In: Proceedings of the Seventh Americas Conference on Information Systems (AMCIS'2001). (2001) 349–356
4. Mansmann, S., Neumuth, T., Scholl, M.H.: OLAP Technology for Business Process Intelligence: Challenges and Solutions. In: Proceedings of DaWaK 2007). (2007) 111–122
5. W.M.P. van der Aalst: Process Cubes: Slicing, Dicing, Rolling Up and Drilling Down Event Data for Process Mining. In: Asia Pacific Conference on Business Process Management (AP-BPM 2013). Volume 159 of Lecture Notes in Business Information Processing., Springer-Verlag (2013) 1–22 Berlin.
6. Liu, M., Rundensteiner, E.A., Greenfield, K.: E-Cube: Multi-Dimensional Event Sequence Analysis Using Hierarchical Pattern Query Sharing. In: Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data (SIGMOD'11). (2011) 889–900
7. Schiefer, J., Jeng, J., Kapoor, S., Chowdhary, P.: Process Information Factory:A Data Management Approach for Enhancing Business Process Intelligence. In: Proceedings of the IEEE International Conference on E-Commerce Technology (CEC '04). (2004) 162–169
8. Liu, R., Vaculin, R., Shan, Z., Nigam, A., Wu, F.: Business Artifact-Centric Modeling for Real-Time Performance Monitoring. In: Proceedings of the International Conference on Business Process Management (BPM2011). (2011) 265–280
9. Chowdhary, P., Bhaskaran, K., Caswell, N., Chang, H., Chao, T., Chen, S., Dikun, M., Lei, H., Jeng, J., Kapoor, S., Lang, C., Mihaila, G., Stanoi, I., Zeng, L.: Model Driven Development for Business Performance Management. IBM Systems Journal **45** (2006) 735–749
10. Abe, M., Jeng, J., Koyanagi, T.: Authoring Tool for Business Performance Monitoring and Control. In: Proceedings of IEEE International Conference on Service-Oriented Computing and Applications (SOCA 2007). (2007)
11. Kudo, M., Nogayama, T., Ishida, A., Abe, M.: Business Process Analysis and Real-world Application Scenarios. In: International Symposium on Data-Driven Process Discovery and Analysis. (2013)
12. Momm, C., Gebhart, M., Abeck, S.: A Model-Driven Approach for Monitoring Business Performance in Web Service Compositions. In: Fourth International Conference on Internet and Web Applications and Services. (2009) 343–350
13. Process Mining Group, Math and CS department, Eindhoven University of Technology.: Mining eXtensible Markup Language (MXML). `http://www.processmining.org/logs/mxml` (2003)
14. W3C Recommendation: XSL Transformations (XSLT) Version 2.0. `http://www.w3.org/TR/xslt20/` (2007)