DPMine: modeling and process mining tool

Sergey Shershakov
International Laboratory
of Process-Aware Information Systems (PAIS Lab)
National Research University Higher School of Economics
Moscow 101000, Russia
Email: sshershakov@hse.ru

***Abstract*—Volume of the data information system operate has been rapidly increasing. Data logs have long been known, as they are a useful tool to solve a range of tasks. The amount of information that is written to a log during a specified length of time leads to the so-called problem of "big data". Process-aware information systems (PAIS) allow developing models of processes interaction, been monitoring accuracy of their performance and correctness of interaction with each other. Studying logs of PAIS in order to extract knowledge about the processes and construct their models has to do with the process mining discipline. There are available developed tools for process mining, both on a commercial and on a free basis. We are proposing a concept of a new DPMine tool for building a model of multistage process mining from individual processing units connected to each other in a processing graph. The resulting model is executed (simulated) by making an incremental process from the beginning to the end.**

***Keywords:* Process Mining, Model, Tool, Distributing, Workflow, Processes, PAIS**

I. Introduction

As a consequence of information systems (IS) development, volume of the data they operate has rapidly increased. This applies both to the data entered into the system in different ways (automatic, semi-automatic and manual) and the data obtained as a result of some processing which is output by the system to various types of media. In the latter data type one can distinguish a special subclass — the so-called *data log* representing a trace left by some IS and storing information about a set of partial states of the system at different times of its work. These are so-called time logs, which include most of the logs.

Data logs have long been known, as they are a useful tool to solve a range of tasks, including diagnosing errors, documenting a sequence of accessing different nodes of the system, maintaining important system information, etc.

The amount of information that is written to a log during a specified length of time can be quite substantial, making it virtually impossible for a user to manually analyze the user log, which leads to the so-called problem of "big data" [1]. For studying large data represented by some orderly way (e.g. DB), different data-oriented techniques such as machine learning, data mining, etc. are involved.

Of particular interest are process-aware information systems, PAIS, the basic concept of which is the *process* (e.g., a business process or a workflow). These systems allow developing models of processes interaction, monitoring accuracy of their performance and correctness of their interaction with each other. These include systems such as BPMS (Business Process Management Systems), CRM (Customer Relationship Management), ERP (Enterprise Resource Planning), etc. As in the case with many other IS such systems can produce large logs containing information on interaction of processes in time.

Study of such logs is of great interest from many points of view. For example, this may be a study for obtaining a pattern of processes interaction which reflects the real situation in a subject field in the form of a mathematical and/or graphical model (Petri nets, UML activity diagrams, BPMN, etc.); the so-called *process discovery* problem. The opposite problem is studying compliance of processes execution with an available (developed manually or automatically obtained) model, i.e. *conformance checking* problem. In general, process discovery and conformance checking are related problems. Also, with this model, one can make some adjustments that are designed to show conceptual changes in regard to the subject area, or can perform permanent *enhancement*.

Studying logs of process-aware information systems in order to extract knowledge about the processes and construct their models, as well as studying such models has to do with the *process mining* discipline, which is relevant to data mining, machine learning, process modeling and model-based analysis. Is a relatively young discipline, its goals and objectives are postulated in the *Process Mining Manifesto* [2] supported by more than half a hundred organizations and a large number of experts from various fields [3].

The *event log* is the starting point for process mining research. Typically, such a log is a sequence of *events* united by a common *case* (i.e., a *process instance*). Each event is related to a certain activity which represents a well-defined step in the process. A set of events relating to the same case makes a so-called trace representing an imprint of processes interaction in a particular case.

There is a variety of data mining techniques but most of them employ the following types of information (*resource*): device or person that initiates and then executes an activity, event data elements and event timestamp.

One of the most frequently used forms of Process Mining models representation is *Petri nets* [4]. Petri nets are used both for internal representation of a model by different algorithms [5], [6] and for visualization. Other models, such as heuristic nets and C-nets, are reduced to Petri nets.

To date, there are available developed tools for process mining, both on a commercial and on a free basis. Examples of these tools include *ProM*, which is probably the most widely used process mining tool. ProM is a framework, the possibilities of which are enhanced by plugins, which are several hundreds in number.

Yet for all its power, ProM has a significant architectural constraint: use of the algorithms that are implemented by numerous plugins, can only be done in a discrete mode with direct participation of the user. In other words, one cannot build a chain of processing operations conducting a multistage procedure of log extraction, cannot analyze, transform, and save (or visualize) data derived from such processing.

This paper proposes a concept of a new *DPMine* tool for building a model of multistage process mining from individual processing units (so called *modular blocks*) connected to each other in a *processing graph*. The resulting model is executed (simulated) by making an incremental process from the beginning to the end and producing intermediate and final results.

In the work, by real-world examples we will show differences in approaches to process mining used in tools like ProM and the new DPMine instrument.

The rest of this paper is organized as follows. Section II describes the modular concept and the basic components of DPMine tool. Section III discusses some specific DPMine blocks as applied to some tasks. Certain aspects of practical implementation of the tool are presented in Section IV. Finally, Section V concludes with an analysis of the work done and a look at the future.

II. Modular model concept

Consider the following problem. Suppose there is a log represented in the computer as an XML-file of a specified format. This log should be processed, for this it should be loaded to an appropriate tool that must support this format. An internal representation of the log is formed after downloading the file (partially or fully).

Suppose one wants to create a model as a Petri net, using some of the implemented algorithms, such as conventional x-miner, by executing which the desired Petri net will be formed.

Next, assume that one needs to build a skeleton graph from this Petri net and then to convert it to a passage graph, and to save the results to a file.

To do this using ProM, one should follow these steps:

1) Run importing a log file (e.g. in XES format).
2) In Actions mode select as Input object “Anonymous log imported from ‘file-name.ext’”.
3) In the list of actions choose the right one for Input object such as “Mine for a Petri Net using Alpha-algorithm”.
4) Run the action (Petri net construction).
5) Set the resulting net as Input Object.
6) ...

By consistently performing steps 2–4 for each type of input object and appropriate action resulting in an output object[1], one can solve the task given above step by step.

Suppose now that after having completed the sequence of actions, the user discovers that at the second stage, while producing a Petri net from the original log, it would have been desirable to use other options of the miner algorithm. This means that all the results obtained after having processed this net become useless and only “obstruct” the resources list. Another example: if a similar sequence of processing operations has to be applied to ten input logs. Or a hundred. Or during mining of the original log one needs to get two models of Petri net, then to compare their similarity.

Obviously, for a flexible solution of such problems a fundamentally different approach to multiprocessing is required, and we proposed it in a tool called DPMine[2].

Program allows creating a *model of multistep process mining* as a graphical diagram (*processing model*) consisting of *modular blocks*. An example of such a scheme is shown in Fig. 1.

This model defines processing sequence for the initial log (*input object*) present in the form of a file (for example, in XML format). It is then consistently submitted to the mining module, the result in the form of Petri net is replicated by a splitter module and sent in three copies respectively to skeleton module construction, visualization module and the module for saving the Petri net model to a file in one of the available formats.

Blocks are executed sequentially, each block begins its work as it receives input data to its *input port* from the *output port* of the previous block. Depending on block type, for starting block’s execution all the data (from all the ports) or partial data may be needed. The modules located “on the right” of the splitter module (Fig. 1) can be executed in parallel, and this can be quasi-parallel in the case of a single processor in a multitasking operating system or “real” parallel if they are run in different threads on a multiprocessor system. There is another version of “true” parallelism: use of special modules for distributed processing that employ as nodes remote computers, which are identified, for example, by IP addresses.

In case of a change in parameters (or input data) of a particular module which is “on the left” in the chain of consecutive executions, the interim data developed “on its right” are announced invalid (invalidation procedure) and are automatically rebuilt (if the model settings are set so). At this, the user does not have to think about how, in what order and at what point he or she needs to perform such a rebuilding: once the scheme is developed, it will then run as many times as it is required to produce a final result satisfying the user.

[1]Actually ProM allows manually setting type of output object before step 3, thus narrowing the range of possible actions.

[2]Letter ‘D’ in the title has many meanings, among which are such as ‘Distinguished’, ‘Direct’ and ‘Distributed’ (as it will be shown below, one of the key features of the new tool is structural distribution of the original objects for analysis, including that on the set of computers (nodes) for splitting task).

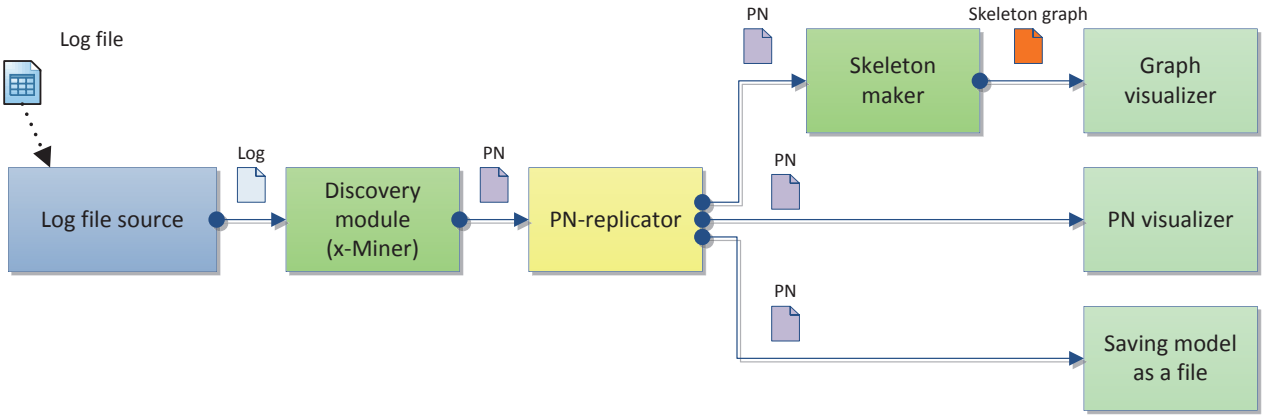

Figure 1. Example of a sequential processing of a log specified in a file

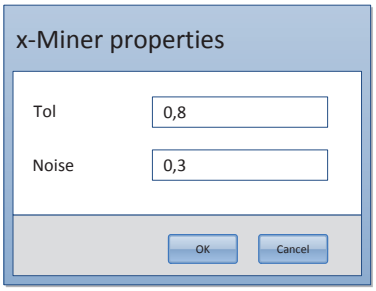

Figure 2. Example of a dialog box for setting parameters of x-miner module

A *processing model* is stored to a file, which allows creating a library of processing models for multiple use for different sets of input data. In addition to information about modules and relationships among them, a processing model stores *modules parameters*, that is a set of characteristics that allow customizing behavior of the modules, for which it is provided. Examples of these parameters can be tolerance params, names of input, output and intermediate files, display options for the modules with a graphical user interface, and so on (see Fig. 2).

As with many other tools, DPMine is developed in a modular fashion that allows extending its functionality through plugins. This way it is possible to connect a large number of third-party modular blocks and significantly expand opportunities for construction of the processing model. Of course, for the entire model to function properly regardless of what modular block is included in its composition, relevant software, implementing their functionality, should be developed in accordance with certain requirements, some of which are covered in the next section.

III. Some peculiarities of modular blocks

Connection of modular blocks between each other is made by means of *ports* that can be either *input* or *output* and correspond to the data types they support. Thus, a logical connection (displayed as directed arrows) in the diagram can be achieved only between output (connector start) and the input (connector end) ports, provided that both ports support the same data type (Fig. 4).

All modular blocks available in DPMine are divided into three groups: *source blocks*, *action blocks* and *render blocks*. They differ in what place in the data processing chain they can occupy.

Consider the purpose of each type of blocks closer.

A. Source blocks

Source blocks have only output ports and, as their name implies, are the source of data to process (Fig. 5). The most obvious example of this block is a log file source that loads a log from the file. Another example would be a block that retrieves a log from some IS, e.g. databases.

Semantically, the source block is a (multi-valued) function with no parameters[3] that returns results which can be used by other functions.

B. Render blocks

Render blocks act as consumers of data obtained as a result of processing and have only input ports. By analogy with the source blocks one of the purposes of render blocks is to store results in a file or export to an IS, for example, to a database (Fig. 6).

Along with that, an equally important feature that the render block provides is results visualization. As soon as DPMine is a modeling tool with a user interface, it means that it must support the graphical representation of models in the form of charts, diagrams, graphs, reports, etc. The framework itself does not restrict users to a few possibilities for visualization of results, but instead offers to plug in visualization modules as render-function blocks.

It is a natural question how to proceed in the development of a model when it is necessary both to visualize results and to save them to a file. Here come to rescue special types of blocks — replicator blocks discussed in Section III-E.

[3]Parameters in this context refers to input and output ports. We should not confuse them with blocks settings, which we shall call *properties*.

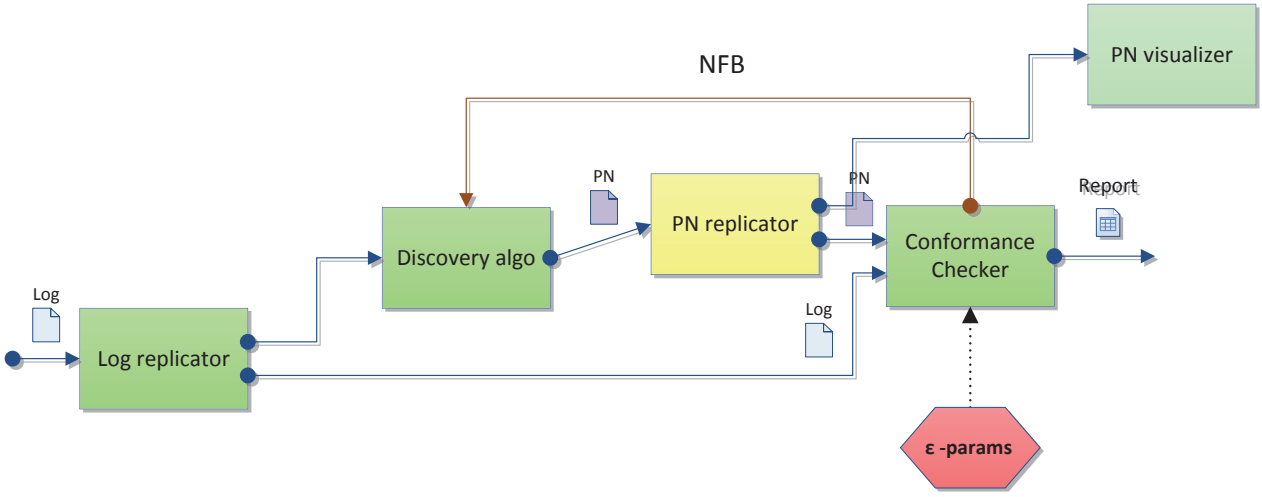

Figure 3. Example of a multistage Process Mining problem solving

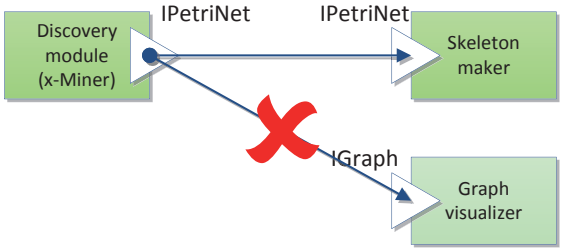

Figure 4. Example of acceptable ports and ports not acceptable for connection (denoted by triangles) which support different types of data

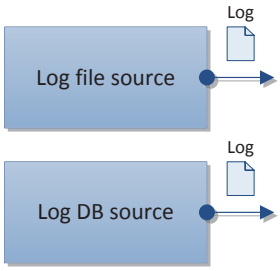

Figure 5. Examples of action blocks

Semantically, the render block is a function of no return values that has results produced by other functions as input parameters.

C. Action blocks

Action blocks are the largest group of blocks having both input and output ports. Semantically, the action block is a function that receives parameters, performs their processing and return them for passing to following functions.

Consider some of the action blocks classes.

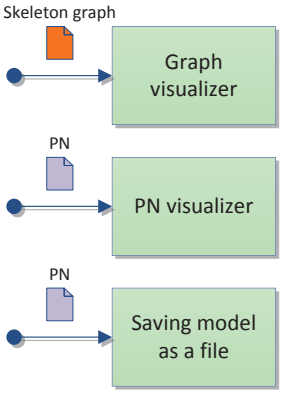

Figure 6. Examples of render blocks

D. Action blocks for solving PM tasks

The main purpose of action blocks for solving PM tasks is execution of serial conversion of input data into outputs for solving a particular PM problem. These are block miners for solving Process Discovery task, comparator blocks for solving Conformance Checking, blocks with feedback for Enhancement task (Fig. 7), etc.

There is a variety of blocks in this class that transfer computational load of their tasks to a cloud or any other system of distributed computing grids, etc. (Fig. 8).

Most of the ProM plugins can be attributed to this very class of action blocks.

E. Control action blocks

A separate class includes action blocks not operating data transformation as such but organizing the structure of their execution model. Their purpose in a way is ideologically close to program control command in programming languages. Previously there was cited an example of a problem when the mine-resulting Petri net has to be visualized by a display block

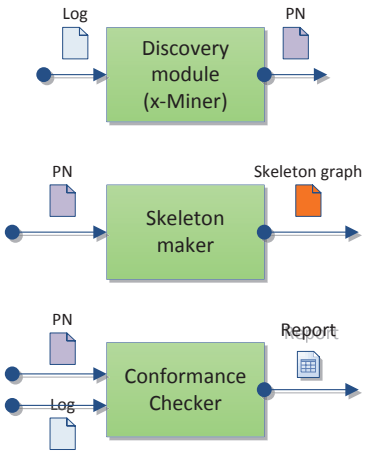

Figure 7. Examples of actions blocks for solving PM tasks

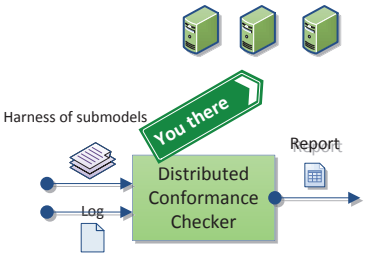

Figure 8. Action block which implements the Conformance Checking task by using distributed calculating nodes

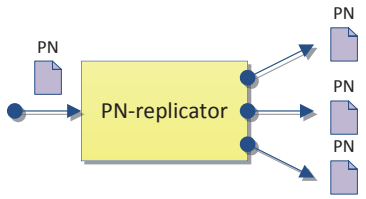

Figure 9. Replicator block performs replication of the inbound PN to multiple outputs

and simultaneously stored (in the form of a model with a set of vertices, transitions, labels, and other support elements) in an external file or database. Here comes to the aid the so-called replicator block (Fig. 9) performing the multiplication of incoming network into multiple outputs. Then one of these outputs can be connected to a visualizer block, another — to a file saving block.

F. Action blocks for distributed calculations

Another major challenge being actively studied in PM is the problem of distributing calculations ([6], [7], [8]). It follows from the computational complexity of the algorithms used in Process Discovery and Conformance Checking, which is manifested in medium and large sets of input data (logs and models), commonly found in experience. It is shown that decomposition of source data (logs section and selection of

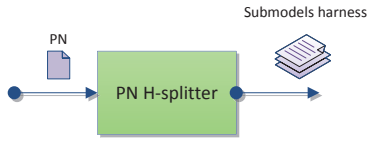

Figure 10. Splitter block with output port in a harness of Petri nets

submodels out of a large model) can significantly reduce the computational load, which stems from the logarithmic complexity of these algorithms.

One can identify three types of distribution: replication, horizontal partitioning of event logs, and vertical partitioning of event logs.

It has been written about replication above, now consider how h-partitioning (h-splitting) and v-partitioning (v-splitting) blocks can be implemented. Based on some heuristic assumptions, the original log is divided into a number of sublogs, which is generally not known in advance. Therefore use of "one sublog — one output port" can be uncomfortable. The answer to this situation is introduction of a special type of data port: *logs harness* and *models harness* (Fig. 10). Of course, to work with a harness, an action block must have an appropriate input port, as in the case with Conformance checking block (Fig. 8). However, one can implement a special demultiplexer block that accepts a harness to input port and its individual elements to outputs ports, and issuing its output ports individual elements of the input harness.

By combining the approaches suggested above, one can build a complex branching model with feedback and get exactly the functionality that is required in each case (Fig. 3).

IV. Practical implementation

The last question that to be treated in this paper relates to practical aspects of implementing DPMine tool.

By the time of writing this paper two parallel development works codenamed DPMine/C and DPMine/J have been going on. As the names might suggest, the first fork of the tool is developed mainly in C++ language using Qt 4.8 library as a framework for building GUI, which allows making it cross-platform. Plugin support is performed using dynamic link libraries, for OS Microsoft Windows these are DLLs.

DPMine/J edition is developed in Java, and the main purpose of this line is to attempt to utilize numerous ProM plugins by designing corresponding wrapper classes to use them with DPMine/J framework interfaces. One of GUI options being considered is to utilize IDE Eclipse with a custom plugin implementing the functionality of DPMine tool.

Because at the moment DPMine project is a pure research, probably at some point later a decision will be taken to focus only on one of the forks — either DPMine/C or DPMine/J. Probably the main arguments in favor of the first or second decision will be to what extent it will be possible to utilize ProM plugins and at which cost.

V. Conclusion

In this paper DPMine tool concept was discussed. It builds up research models in process mining area. Despite the fact that works on the tool are at the beginning, some of the tasks have already found their realization. Another part is subject to change in the process of working on it.

Among the challenges for the future the following task can be identified, namely developing a declarative language of the underlying graphical model that allows describing complex models.

Acknowledgment

The study was implemented in the framework of the Basic Research Program at the National Research University Higher School of Economics (HSE) in 2013.

The author would like to thank Prof. Irina A. Lomazova for her vital encouragement and support.

References

[1] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, and A. H. Byers, "Big data: The next frontier for innovation, competition, and productivity," Tech. Rep., 2011.

[2] IEEE Task Force on Process Mining, "Process Mining Manifesto," in *BPM 2011 Workshops*, ser. Lecture Notes in Business Information Processing, F. Daniel, S. Dustdar, and K. Barkaoui, Eds., vol. 99. Springer-Verlag, Berlin, 2011, pp. 169–194.

[3] W. M. P. van der Aalst, *Process Mining - Discovery, Conformance and Enhancement of Business Processes*. Springer, 2011.

[4] C. A. Petri and W. Reisig, "Petri net," *Scholarpedia*, vol. 3, no. 4, p. 6477, 2008.

[5] J. Carmona, J. Cortadella, and M. Kishinevsky, "A region-based algorithm for discovering petri nets from event logs," in *BPM*, 2008, pp. 358–373.

[6] W. Aalst, "Decomposing Process Mining Problems Using Passages," in *Applications and Theory of Petri Nets 2012*, ser. Lecture Notes in Computer Science, S. Haddad and L. Pomello, Eds., vol. 7347. Springer-Verlag, Berlin, 2012, pp. 72–91.

[7] W. M. P. van der Aalst, "Distributed process discovery and conformance checking," in *FASE*, 2012, pp. 1–25.

[8] ——, "Decomposing petri nets for process mining. a generic approach," Department of Mathematics and Computer Science, Technische Universiteit Eindhoven, The Netherlands., Tech. Rep., 2012.