Process Mining to Discover the Global Process from its Fragments' Executions

Minh Khoi Nguyen, Hanh Nhi Tran and Ileana Ober
Institut de Recherche en Informatique de Toulouse, Toulouse, France

Keywords: Process Mining, Artifact-centric Process Modeling, Data-driven Process Engine.

Abstract: Process analysis to improve performance and detect anomalies is important for process management. However, such an analysis requires a global process model that is sometimes hard to get, especially for complex processes involving various teams. This is also an obstacle for *BAPE (Bottom-up Artifact-centric Process Environment)*, an artifact-centric process management environment that allows splitting a process into several fragments which are separately modeled and enacted by different actors. Thus, the knowledge about the whole process is distributed over the involved teams and an overview on the complete process is missing. This paper presents the integration of process mining into *BAPE* in order to construct the overall process model from the execution data of process fragments.

1 INTRODUCTION

Process Management Systems (PMSs) provide supports at the operational level to manage the execution of processes. A PMS relies on explicit process models to coordinate and synchronize process activities at enactment time. Mainstream PMSs embody a top-down approach that models a process as a whole to enable its enactment. In contrast to the conventional PMSs, in our previous work, we proposed a bottom-up approach that splits a process into several fragments which are separately modeled and enacted by process actors who play different roles in the process (Hajmoosaei et al., 2017).

We have developed BAPE (Bottom-up Artifact-centric Process Environment) for modeling and enacting process fragments. *BAPE* use a data-driven process engine to construct progressively, at enactment time, the global process from different process fragments. Thus, to execute a process, *BAPE* does not require the model of the whole process, but accepts a partially-defined model, i.e. one with some missing fragments. Although this feature is an advantage of *BAPE*, it also makes process analyzing harder in *BAPE* due to the lack of an overall process model.

Process Mining (van der Aalst, 2016) is a promising technique for diagnosing processes by mining event logs to extract knowledge. Since its emergence, process mining has gained important achievements in providing techniques and tools for discovering different process aspects (van der Aalst et al., 2004; van der Aalst et al., 2017; Peña and Bayona-Oré, 2018; Kim et al., 2021) and has been applied in numerous domains (Rovani et al., 2015; Jokonowo et al., 2018; Valencia-Parra et al., 2019; dos Santos Garcia et al., 2019).

The work presented in this paper aims at improving *BAPE* by using process mining to discover the global process from the execution traces of process fragments. Integrating process mining into *BAPE* has two benefits: (1) enabling the analysis of the whole process in order to detect anomalies; (2) generating process models of the missing fragments to enact and control them inside *BAPE* in the future.

The remainder of this paper is organized as follows. Section 2 presents the current environment *BAPE* with an example of a partially-defined process. Section 3 introduces the process mining steps to exploit the execution data of process fragments for discovering the global process model and process fragments. Finally, section 4 concludes the paper.

2 BAPE

BAPE is a process management environment that adopts an artifact-centric and data-driven approach to model and enact processes. To better explain the main characteristics of *BAPE*, we present first a running example, then illustrate the propositions of *BAPE* on this

Nguyen, M., Tran, H. and Ober, I.
Process Mining to Discover the Global Process from its Fragments' Executions.
DOI: 10.5220/0011044800003176
In *Proceedings of the 17th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2022)*, pages 363-370
ISBN: 978-989-758-568-5; ISSN: 2184-4895

example.

2.1 Illustration Example

We take an extract of a process given by our industrial partners as the running example for this paper. Figure 1 shows the process *Modify Test-bench Wiring* which is executed to reconfigure the wiring system of a test-bench due to an evolution of the system under test. The process concerns several expertise domains that can be geographically dispersed. Each role, i.e. *Analyst*, *Electrical Designer*, *Instrumentation Team*, *Supplier*, *Wiring Team* and *Bench Coordinator*, performs specifics tasks to produce the required artifacts in order to configure the test-bench.

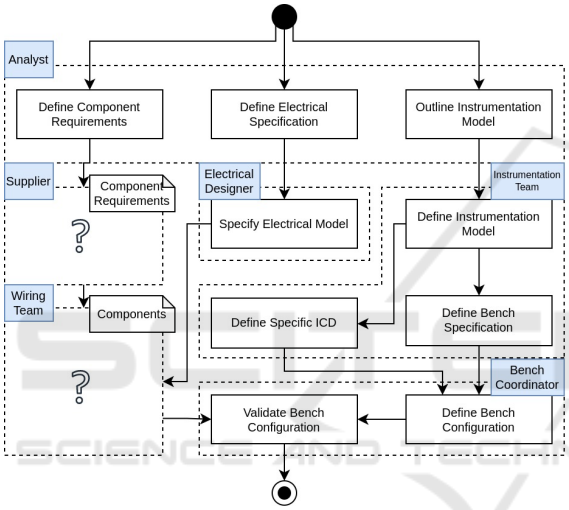

Figure 1: Process of *Modify Test-bench Wiring*.

The process starts when the *Analyst* receives the change requirement. Afterwards, he performs the tasks *Define Component Requirement*, *Define Electrical Specification* and *Outline Instrumentation Model* to specify the required modifications or acquisitions on *wiring*, *electrical* and *instrumentation* components of the test-bench. These modifications are realized in the next steps respectively by the *Wiring Team*, *Electrical Designer* and *Instrumentation Team*. After installing and wiring all required components in the test-bench, process finishes when the *Bench Coordinator* validates the configured test-bench.

2.2 Fragmented Process Model

In *BAPE*, a *Process* is comprised of several process fragments. A *Process Fragment* encapsulates the activities performed by a specific *Role*. An *Activity* describes a work composed of enactable tasks corresponding to different sub-objectives. A *Task*, *Mandatory* or *Optional* represents an undivided work performed by a process actor to put an *Artifact*, i.e. a tangible work product, into a specific *State*. To avoid being prescriptive about the way enacting processes, *BAPE* does not define explicitly the work-sequence relations between tasks. However, if two tasks work on the same artifact, the artifact's *demandedState* in the tasks' preconditions and post-conditions define the order to execute these two tasks.

Figures 2 illustrates a fragment of the process in Figure 1 corresponding to the role *Analyst*. The process fragment includes an activity *Detail Change Requirement* composed of three tasks. The two *mandatory tasks Define Component Requirements* and *Define Electrical Specification* are always realized but the *optional task Outline Instrumentation Model* will be executed only if the option *instrumentation* is activated. The mandatory task *Define Electrical Specification* requires two input artifacts *Wiring Change Demand* and *FICD* in states *defined*. The later artifact, produced by the *Instrumentation Team*, is required only if the *instrumentation* option is activated.

Process Modify Test-bench Wiring **options** instrumentation, interface
ProcessFragment Analyst
Activity Detail Change Requirement **options** instrumentation
MandatoryTask Define Component Requirements
Artifact Wiring Change Demand(**in**, defined)
Artifact Component Requirements(**out**, defined, **collection**)
MandatoryTask Define Electrical Specification
Artifact Wiring Change Demand(**in**, defined)
Artifact FICD(**in**, defined, **toFinish**) **ifOption** instrumentation
Artifact Electrical Specification(**out**, defined)
OptionalTask Outline Inst. Model **ifOption** instrumentation
Artifact Wiring Change Demand(**in**, defined)
Artifact FICD(**out**, outlined)

Figure 2: Process Fragment of Analyst.

A process actor can use the process modeling language of *BAPE* to model the process fragments of the roles that he plays in a project. Each process fragment can be modeled independently of the other fragments but using the shared company assets.

BAPE does not require all the defined process fragments to execute the whole process because it coordinates the tasks in different fragments by the states of exchanged artifacts. For instance, in our running example, the process fragments of *Analyst*, *Electrical Designer*, *Instrumentation Team*, and *Bench Coordinator* are provided but those of *Supplier* and *Wiring Team* are not defined. Thus, when the *Modify Test-bench Wiring* process is executed, only the tasks of *Analyst*, *Electrical Designer*, *Instrumentation Team*, and *Bench Coordinator* are controlled by *BAPE*, but they can be synchronized with those of *Supplier* and *Wiring Team* thanks to the exchanged artifacts *Component Requirements* and *Component Specifications*.

2.3 Components of *BAPE*

Figure 3 shows the architecture of *BAPE* with the main components to control the execution of process fragments operated by different actors.

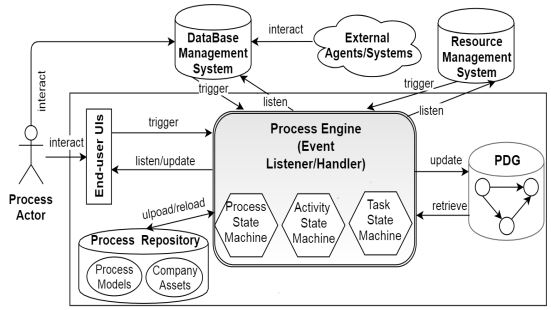

Figure 3: Architecture of *BAPE*.

1. *End-User UI:* This component provides interfaces that allow process actors to interact with the process environment. Process actors define their process fragments then store them into the *Process Repository* via the modeling interface. They use the enactment interface to manage their activities and tasks (e.g., create, start and complete tasks). Each user's action triggers a specific event that is handled by the process engine which decides if the required action is valid to be taken.
2. *Database Management System:* We suppose that resources (e.g., process actors, tools, etc.) and working artifacts are externally managed by each company proper tools. However, those external tools use a central Database Management System (DBMS) which is connected to *BAPE*. This connection makes the process environment being aware of any events changing the state of artifacts and resources in the external information systems of process actors.
3. *Process Dependency Graph (PDG):* In *BAPE*, the information of running processes is stored in a graph-based structure called PDG, which is composed of nodes representing process elements (i.e. tasks, activities, artifacts and actors) and arcs representing the relationships between these process elements (e.g. links between a task and its input and output artifacts). PDG is progressively updated whenever a task from any of process fragment is realized. In this way, at enactment time, PDG shows the current state of process elements and establishes the global view of the system. On one hand PDG is the information source for the process engine to coordinate and synchronize tasks, on the other hand PDG is the source to extract execution traces of running process fragments. Figure 4 shows an example of PDG containing elements of two running process fragments of the roles *Analyst* and *Instrumentation*. These teams enact their tasks separately but can be coordinated thanks to the shared artifact *FICD*.
4. *Process Engine:* This component is in charge of enacting and synchronizing the processes managed by *BAPE*. It listens to events triggered by process actors (when doing tasks in the process fragments controlled by *BAPE*) or by external systems (when updating elements of the process fragments that are not controlled by *BAPE*) and applies appropriate actions to handle these events.

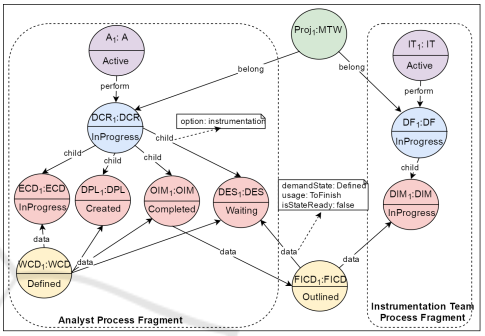

Figure 4: A snapshot of the PDG.

3 PROCESS MINING IN BAPE

We integrate process mining into *BAPE* for process discovery purpose, i.e. we analyze the execution data kept in an event log to build a process model. In the context of *BAPE*, getting an event log containing all information needed for mining the overall process is not obvious because *BAPE* can execute partially defined processes. Concretely, a process can be enacted with some defined process fragments which are managed inside *BAPE* and some missing fragments which are not controlled by *BAPE*. While the execution data of defined fragments are stored inside *BAPE*, those of missing fragments are located in process actors' own information systems. Therefore, it is necessary to extract the execution traces from both of these sources to build an event log that enables discovering the activities of different process fragments, even the missing ones.

The process mining steps for *BAPE* are shown in Figure 5. The first step *Event log pre-processing* (Section 3.1) is for collecting the execution data from two sources, *BAPE* event log and process actor's information systems, and building an event log in XES format[1]. Then the event log will be mined to extract

[1] http://xes-standard.org/start

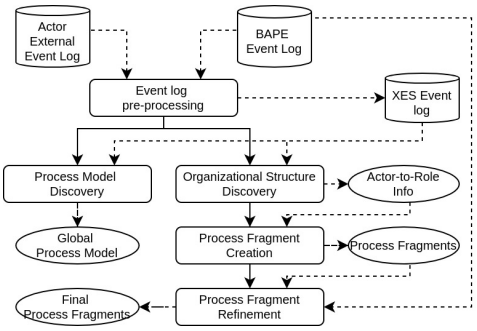

Figure 5: Process mining integration into *BAPE*.

knowledge about the global process on two aspects: (1) *Process Model Discovery* (Section 3.2) establishes the control flow between tasks executed in different fragments and (2) *Organizational structure Discovery* (Section 3.3) deduces the relationship between process actors and the tasks they performed. The two final steps presented in Section 3.4 are about creating process fragment models corresponding to the identified roles of process actors.

3.1 Event Log Pre-processing

As explained earlier, we use two sources of execution data of managed and unmanaged process fragments to build an activity-centric event log for process mining. Each event corresponds to the performing of a task in a process instance (called *case*). To allow discovering the control-flow between tasks as well as the resources used by tasks (process actors and artifacts needed for a task), each item in the even log must contain the requisite information of an event: timestamp, event name, actor relating to the event and artifacts used in that event.

Code 1 shows an event in the event log extracted from *BAPE* execution traces: actor *A1* performed the task *Define Component Requirements* using *Wiring Change Demand* artifact at 2021-10-12 11:04:54. It contains all necessary information, which is perfect for process mining algorithm.

```
<event>
<string key="org:resource" value="A1"/>
<date key="time:timestamp" value="2021-10-12T11:03:54.032"/>
<string key="concept:name" value="Define Component Requirements"/>
<list key="artifactlifecycle:moves">
<values>
string key="artifactlifecycle:model" value="Wiring Change Demand">
<string key="artifactlifecycle:instance" value="WCD1"/>
<string key="artifactlifecycle:transition" value="in"/>
</string></values></list></event>
```

Code 1: An event extracted from *BAPE* execution traces.

In contrast to *BAPE* event logs, it was more complicated with the execution data extracted directly from process actors' information systems of the missing fragments. In general, the transaction history in an information system is created to keep traces of manipulations on artifacts thus provides an artifact-centric event logs. For example, Table 1 shows a snapshot of the *Supplier* relational database in which there is no explicit task recorded, but the history of manipulations on *orders* and the relation between *orders* and *items*). Table 2 shows a working history of the *Wiring Team* which stores the changes made on managed components: the status, the person in charge and the timestamp of change.

Table 1: A snapshot of the *Supplier*'s information system.

orderId	order	personId	receive	release
o1	project1	s4	2022-01-20 12:34:56	2022-02-10 17:01:48
o2	project2	s3	2022-02-01 13:04:15	

itemId	name	price
i1	component1	300
i2	component2	200

orderId	itemId	quantity
o1	i1	2
o1	i2	3

Table 2: An artifact log extracted from the *Wiring Team*'s information system.

id	name	status	personId	timestamp
i1	component1	purchased	p1	2022-01-30 12:00:05
i2	component2	installed	p2	2022-02-05 16:40:38

In artifact-centric events logs, events of different instances of the same artifact type can be in the same case (divergence) or duplication of the same event can appear into different cases (convergence). For example, in the *Supplier* log, many component requirements from the *Analyst* (see Figure 1) could be the *items* of the same *order*, which raises the convergence issue. On the other hand, the log of *Wiring Team* might have the divergence problem, since the same installation step may appear multiple times for different components in a case. In order to deal with such complications, we had to apply some pre-processing on the logs obtained from the missing fragments that allow us to construct activity-centric logs more suitable for our process mining algorithm. Due to the limit of space, the details of this pre-processing is not presented here.

Another issue is that the available *BAPE* execution traces collected during the past project are insufficient in quantities for running a useful process mining. Thus, in addition to the real execution data, we simulated more execution traces of process fragments of *BAPE*. To ensure that the simulated event log can emulate the real one, we generated events diversifying in numerous factors: workflow, human resources, timing, etc. The constrains on work sequences between tasks in the generated events must be respected. For example, in the *Modify Test-bench Wiring* process, all the tasks of the role *Analyst* can be done in parallel while those of the *Wiring Team* can start only when the *Supplier* finishes their task. Consequently, we can generate simultaneous events of tasks done by *Analyst* but can not generate an event of a *Wiring*

Team task before an event of a *Supplier* task.

The simulated log was built to include some unexpected cases representing anomalies. In general, there are two popular anomalies situations during process enactment: (1) anomalies of the organization where an actor executes a task that is not assigned to his role; (2) anomalies of control-flows where an actor skips a task assigned to him or realized the task without reporting its enactment.

To generate the anomalies in the event log, we use two parameters: the parameter n for the number of anomalies occurring in a single trace and the parameter t for the type of anomalies appearing in the traces. Each time generating a new set of event log, the position and the number of these unexpected cases varies randomly. Each time a trace is built, a random number r is generated to control the opportunity that the trace has anomalies. If $r \leq t$, $(0 \leq t \leq 0.5)$, then we integrate in the trace an anomaly of type (1) (an actor does a task of another role). If $r \geq 1-t$, an anomaly of type (2) (an actor does not performed his task) will be added into the trace. To test the adaptation of our algorithm to multiple anomalies in the event log, we made some controls on the appearance rate of those anomalies by setting the parameters n and t with the appropriate values. We suggest setting the first parameters $n = 2$, which means at most 2 tasks in a trace will be the anomalies. Also, we choose to configure the second parameter $t = 0.05$. If we set a higher value for one of these two parameters, the unexpected cases are not the anomalies anymore but become normal and would emerge a new dependency in the process model.

We could collect 320 traces of the event logs from both *BAPE* system and the external information systems of the *Modify Test-bench Wiring* process's actors. We simulated an event log with 680 traces. Together, there are 1000 traces to perform the process mining for the running example.

3.2 Process Model Discovery

Algorithm 1 resumes the steps based on the heuristic-miner algorithm (Weijters et al., 2006) to discover the process model P from an event log L.

The final process model is presented as a heuristic net composed of tasks and the dependencies between them. Given two tasks a and b, a is said to be followed by b (denoted by $a \longrightarrow_L b$) if the dependency measure between a and b (denoted by $D(a,b)$) calculated from the log L is greater than a given *filter*.

The first step is to construct the dependency measure list $D_L = \{D(a,b)\}$ for each pair of tasks in the log

Algorithm 1: Process Model Discovery.

Input: $L, filter$
Output: P
1 Construct dependency measure list D_L
// Construct dependency list X_L
2 $X_L = \{\}$
3 **forall** $(a,b) \in D_L$ **do**
4 **if** $D(a,b) \geq filter$ **then**
5 $X_L = X_L \cup a \xrightarrow{\mathrm{D(a,b)}}_L b$
6 **end**
7 **end**
8 Build the process model P from X_L

L. For two tasks (a,b), $|a \Rightarrow_L b|$ is the value of the dependency relation between a and b and is computed using the equation 1 where $|a >_L b|$ is the number of times a is directly followed by b.

$$|a \Rightarrow_L b| = \begin{cases} \frac{|a>_L b| - |b>_L a|}{|a>_L b| + |b>_L a| + 1} & if\ a \neq b \\ \frac{|a>_L a|}{|a>_L a| + 1} & if\ a = b \end{cases} \quad (1)$$

$D(a,b)$ produces a value between -1 and 1. If $D(a,b)$ is closer to 1 then we deduce the high possibility that a is followed by b. In contrast, if $D(a,b)$ is closer to -1, there is a high possibility that b is followed by a. When $D(a,b)$ is around 0, there is nearly no intuitive relation between a and b.

A threshold $filter$ is used to exclude the dependencies caused by noises between two tasks. Only the dependencies $a \xrightarrow{\mathrm{D(a,b)}}_L b$ with $D(a,b) \geq filter$ are kept and then translated to the result process model P.

At this step, only work-sequences anomalies are detected because organizational anomalies (where an actor enacts an unassigned task) concerns the issue of human resources and does not affect the integrity of the process control flow.

Table 3: *Modify Test-bench Wiring* process's tasks.

Task name	Decoded
Define Component Requirements	a
Define Electrical Specification	b
Outline Instrumentation Model	c
Purchase Components	d
Specify Electrical Model	e
Define Instrumentation Model	f
Define Bench Specification	g
Define Specific ICD	h
Install Components	i
Wire Components	j
Define Bench Configuration	k
Validate Bench Configuration	l

To facilitate the presentation of the mining results, we show in Table 3 the decoded names of *Modify Test-bench Wiring* process's discovered tasks.

Table 4 gives an extract of dependency measures

Table 4: Extract of dependency measure between tasks in *Modify Testbench Wiring* with $n = 2, t = 0.05$

	c	g	i	j	l
c	0.000	0.667	-0.045	-0.197	0.000
g	-0.667	0.000	0.018	-0.211	0.750
h	0.750	0.082	-0.044	-0.044	0.857
i	0.045	-0.018	0.000	0.998	0.667
j	0.197	0.211	-0.998	0.000	0.998

between tasks in *Modify Testbench Wiring* process. $D(i, j)$ has the value 0.998, while $D(c, g)$ has the value 0.667. It means there is a higher chance that $i \rightarrow_L j$ but there might be a chance that $c \rightarrow_L g$ is a noise dependency caused by a process work-sequences anomaly. For example there might be a task between c and g but the actor skipped that intermediate task. We can use a threshold $filter$ to decide which dependencies should be kept in the final process model. For example, by setting $filter = 0.9$, we only keep high reliable dependencies as $i \rightarrow_L j$ and mask the noise dependencies with $D_(a,b) \leq 0.9$ and therefore eliminate $c \rightarrow_L g$ in the discovered process model (c.f. Figure 6).

3.3 Organizational Structure Discovery

The discovered global process model obtained in Section 3.2 focuses only on the control flows between tasks. To establish the performing relationship between actors and tasks, we need to execute an analysis to recognize the organizational structure of the examined process.

For this purpose, we use a resource-activity matrix $M(A,T)$, where A is the list of actors and T is the list of tasks. This matrix is built when calculating the mean number of times an actor performs a task per trace (van der Aalst et al., 2005). Based on this matrix, we can determine the similarity values among the tasks of some actors. By heuristic evaluation, these actors might share the same role.

For better recognizing the actor-to-role information, we decided to use the clustering algorithm DBSCAN (Ester et al., 1996). Assuming that each resource-activity value $m(a,t)$ in the matrix represents a position in an axis of a coordinate system, thus all the resource-activity values of an actor $m(a,T)$ determines a multi-dimensions point at a specific position in that coordinate system. When grouping near points together with the euclidean distance, the actors who perform the same tasks will be grouped into one role C_j. The number of points reflects the number of actors in the event log.

The analysis process is shown in Algorithm 2. Two parameters *minPoints* - the minimum number of actors required to form a role and *minDistance* - the minimum difference between the list of tasks done by 2 actors - should be considered rigorously to identify the correct role of actors.

Algorithm 2: Organizational Structure Discovery.

Input: $L =< A, T >, minPoints, minDistance$
Output: $PF(C_i)$
Compute resource-activity matrix $M(A,T)$ from L
Apply DBSCAN to $M(A,T)$ with $minPoints, minDistance$ to get $R\{C_i\}$
forall $t \in T$ **do**
  Find $a^* = arg\max_{a \in A} m(a,t)$
  $PF(C_i) = PF(C_i) \cup t$ if $a^* \in C_i$
end

Analyzing the resource-activity matrix helps detect organizational anomalies where actors perform tasks not belonging to their role. The resource-activity values of such anomalies are significantly lower than the regular circumstance when the actor realizes his proper tasks. By checking the value of $m(A,t)$, task t is identified as the task assigned to the role C_j of the actor who realizes task t the most frequently $PF(C_j)$. From the list of roles found in this step, we can reconstruct process fragments of identified roles.

Table 5 shows the resource-activity matrix of the *Modify Testbench Wiring* process. From this matrix we can deduce that *BC1* and *BC2* do task k and l; *WT2, WT1* and *WT3* do task i and j and so on with other actors. The values especially low around *0.002*, for example *T4* with task l or *A1* with task h, reveal possible organizational anomalies where an actor performs an unsigned task.

By setting $minPoints = 1, minDistance = 0.1$, from the Table 5 we establish the *performer relationship* between tasks and actors, then deduce 6 roles as shown in Table 6.

Table 5: Resource-activity matrix.

Actor	a	b	c	d	e	f	g	h	i	j	k	l
T4	0.000	0.000	0.000	0.000	0.000	**0.274**	**0.274**	**0.273**	0.000	0.000	0.000	0.002
A1	**0.516**	**0.513**	**0.526**	0.000	0.000	0.000	0.000	0.002	0.000	0.000	0.002	0.000
BC2	0.002	0.002	0.000	0.000	0.002	0.000	0.002	0.002	0.000	0.000	**0.478**	**0.477**
A2	**0.483**	**0.484**	**0.484**	0.000	0.000	0.000	0.000	0.000	0.000	0.002	0.000	0.000
BC1	0.000	0.002	0.000	0.000	0.002	0.000	0.000	0.002	0.000	0.000	**0.520**	**0.518**
ED1	0.000	0.000	0.000	0.002	**0.998**	0.000	0.000	0.002	0.000	0.002	0.000	0.000
WT2	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	**0.343**	**0.344**	0.000	0.002
WT1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	**0.336**	**0.335**	0.000	0.000
WT3	0.000	0.002	0.000	0.000	0.000	0.000	0.000	0.000	**0.319**	**0.318**	0.000	0.002
T1	0.000	0.000	0.000	0.000	0.000	**0.277**	**0.275**	**0.274**	0.002	0.000	0.000	0.000
S1	0.000	0.000	0.000	0.999	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
T2	0.000	0.000	0.000	0.000	0.000	**0.230**	**0.231**	**0.229**	0.002	0.000	0.000	0.002
T3	0.000	0.000	0.000	0.000	0.000	**0.220**	**0.219**	**0.220**	0.000	0.000	0.002	0.002

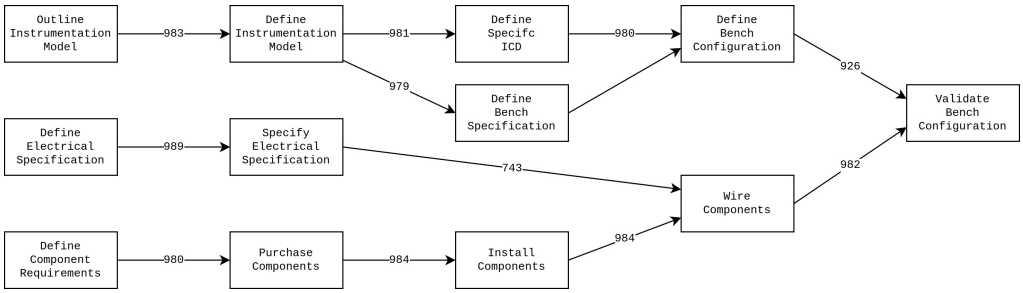

Figure 6: Process Model discover with $filter = 0.9$.

Table 6: Clusters found for actors and tasks.

Cluster	Actors	Tasks
Role 1	T1, T2, T3, T4	f, g, h
Role 2	A1, A2	a, b, c
Role 3	BC1, BC2	k, l
Role 4	ED1	e
Role 5	WT1, WT2, WT3	i, j
Role 6	S1	d

3.4 Process Fragment Creation

Based on the detected information about actor-role-task relations, the process fragments of each identified role can be generated under the format of *BAPE*. However, to complete these process fragments, we need to find out the exact role names because the identified roles are just numbered.

If the reconstructed process fragments controlled by *BAPE*, we can deduce the role names from the existing process fragment models in the process repository of *BAPE*. For the discovered fragments whose process models were missing and thus were not controlled by *BAPE*, the renaming of roles could be done automatically if the role names were specified in process actors' execution data. If not, the renaming are done manually.

Figure 7 shows a part of the process fragment discovered for the *Role* 1. By comparing it with available process fragments, we found a match and could rename *Role* 1 to *Instrumentation Team*. In the same way, *Role* 2 is renamed to *Analyst*, *Role* 3 to *Bench Coordinator* and *Role* 4 to *Electrical Design*.

Figure 8 and 9 display respectively the discovered process fragments for *Role* 5 and *Role* 6. Because there are no matched process fragments for *Role* 5 and *Role* 6 in *BAPE*, we analyzed the exchanged artifact between these fragments and the other fragments in the process to deduce the name of the missing roles. For example, when searching the exchanged artifact *Component Requirements* (for fragment of *Role 6*) among known process fragments, we found that *Component Requirements* is the output artifact of *Define Component Requirements* task. Recall the Figure 1, we can deduce that *Role 6* is the *Supplier*. Similarly, applying the same approach, we clarified that *Role 5* is the *Wiring Team*. Finally, we discovered the missing fragments and rebuilt the complete *Modify Testbench Wiring* process model.

```
<ProcessFragments name="Role 1">
  <ProcessFragment name="Modify Testbench Wiring" version="">
    <Activity name="Define Specific ICD">
      <MandatoryTask name="Define Specific ICD">…
      </MandatoryTask>
    </Activity>
    <Activity name="Define Bench Specification">
      <MandatoryTask name="Define Bench Specification">…
      </MandatoryTask>
    </Activity>
```

Figure 7: Discovered process fragment of Role 1.

```
<ProcessFragments name="Role 5">
  <ProcessFragment name="Modify Testbench Wiring" version="">
    <Activity name="Install Components">
      <MandatoryTask name="Install Components">
        <InputArtifact name="Components" usage="ToStart"/>
        <OutputArtifact name="Installed Components" usage="ToFinish"/>
    <Activity name="Wire Components">
      <MandatoryTask name="Wire Components">
        <InputArtifact name="Components" usage="ToStart"/>
        <OutputArtifact name="Testbench Wired" usage="ToFinish"/>
</ProcessFragments>
```

Figure 8: Discovered process fragment of Role 5.

```
<ProcessFragments name="Role 6">
  <ProcessFragment name="Modify Testbench Wiring" version="">
    <Activity name="Purchase Components">
      <MandatoryTask name="Purchase Components">
        <InputArtifact name="Component Requirements" usage="ToStart"/>
        <OutputArtifact name="Components" usage="ToFinish"/>
</ProcessFragments>
```

Figure 9: Discovered process fragment of Role 6.

4 CONCLUSION

We have developed a new functionality for *BAPE* to discover, from the execution traces of process actors, the global process model as well the missing process fragments. The overall process model enables *BAPE* to do more analysis, to detect the anomalies concerning the missing tasks or the mismatched role-task. The discovered models of missing process fragments could enable the process actors to use *BAPE* in the future to manage their processes.

The validation of this work however was limited by the lack of real execution data that could help

to obtain more interesting insights. Another drawback of this work is the omission if multiple-instances case. One challenge of process mining for artifact-centric processes is handling with many-to-many relationships between artifacts, which make it difficult to identify a unique process instance notion to group related events. Many works have addressed this problem, from the modeling viewpoint (van der Aalst et al., 2001; Cohn and Hull, 2009; Lohmann, 2013), or from the process mining algorithm angle (Schuster et al., 2021; Li et al., 2017; Li et al., 2018). *BAPE* adopts a hybrid approach of artifact-centric and activity-centric, thus does not face exactly the difficulty of the pure artifact-centric process approach. Therefore we could still apply the traditional miner algorithms when integrating process mining into *BAPE* with the assumption that artifacts are manipulated by a tasks of the same process instance. However, such an assumption is not always true, especially for execution data extracted directly from process actor's information systems. There are still more rooms for improvement of our works, especially for exploring the possibility of multiple-instances related to a case and for investigating other types of anomalies.

REFERENCES

Cohn, D. and Hull, R. (2009). Business artifacts: A data-centric approach to modeling business operations and processes. *IEEE Data Eng. Bull.*, 32:3–9.

dos Santos Garcia, C., Meincheim, A., Faria Junior, E. R., Dallagassa, M. R., Sato, D. M. V., Carvalho, D. R., Santos, E. A. P., and Scalabrin, E. E. (2019). Process mining techniques and applications – a systematic mapping study. *Expert Systems with Applications*, 133:260–295.

Ester, M., Kriegel, H.-P., Sander, J., and Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, page 226–231. AAAI Press.

Hajmoosaei, M., Tran, H. N., and Percebois, C. (2017). A user-centric process management for system and software engineering projects. In *7th International Conference on Industrial Engineering and Systems Management (IESM 2017)*, pages 123–128, Saarbrücken, Germany.

Jokonowo, B., Claes, J., Sarno, R., and Rochimah, S. (2018). Process mining in supply chains: A systematic literature review. *International Journal of Electrical and Computer Engineering*, 8:4626–4636.

Kim, K.-S., Pham, D.-L., and Kim, K. P. (2021). p-algorithm: A sicn-oriented process mining framework. *IEEE Access*, 9:139852–139875.

Li, G., de Carvalho, R. M., and van der Aalst, W. M. P. (2017). Automatic discovery of object-centric behavioral constraint models. In Abramowicz, W., editor, *Business Information Systems*, pages 43–58, Cham. Springer International Publishing.

Li, G., de Murillas, E. G. L., de Carvalho, R. M., and van der Aalst, W. M. P. (2018). Extracting object-centric event logs to support process mining on databases. In Mendling, J. and Mouratidis, H., editors, *Information Systems in the Big Data Era*, pages 182–199, Cham. Springer International Publishing.

Lohmann, N. (2013). Compliance by design for artifact-centric business processes. *Information Systems*, 38(4):606–618. Special section on BPM 2011 conference.

Peña, M. R. and Bayona-Oré, S. (2018). Process mining and automatic process discovery. In *2018 7th International Conference On Software Process Improvement (CIMPS)*, pages 41–46.

Rovani, M., Maggi, F. M., de Leoni, M., and van der Aalst, W. M. (2015). Declarative process mining in healthcare. *Expert Systems with Applications*, 42(23):9236–9251.

Schuster, D., van Zelst, S. J., and van der Aalst, W. M. P. (2021). Cortado—an interactive tool for data-driven process discovery and modeling. *Lecture Notes in Computer Science*, page 465–475.

Valencia-Parra, A., Ramos-Gutiérrez, B., Varela-Vaca, A., Gómez-López, M., and Bernal, A. (2019). Enabling process mining in aircraft manufactures: Extracting event logs and discovering processes from complex data. *CEUR Workshop Proceedings*, 2428:166–177.

van der Aalst, W., Barthelmess, P., Ellis, C., and Wainer, J. (2001). Proclets: A framework for lightweight interacting workflow processes. *International Journal of Cooperative Information Systems*, 10:443–481.

van der Aalst, W., Reijers, H., and Song, M. (2005). Discovering social networks from event logs. *Computer Supported Cooperative Work*, 14:549–593.

van der Aalst, W., Weijters, T., and Maruster, L. (2004). Workflow mining: discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142.

van der Aalst, W. M. P. (2016). *Process Mining: Data Science in Action*. Springer, Heidelberg, 2 edition.

van der Aalst, W. M. P., Bolt, A., and van Zelst, S. J. (2017). Rapidprom: Mine your processes and not just your data.

Weijters, A., van der Aalst, W., and Medeiros, A. (2006). *Process Mining with the Heuristics Miner-algorithm*, volume 166.