International Journal of Innovative
Computing, Information and Control
Volume **14**, Number **4**, August **2018**

ICIC International ©2018 ISSN 1349-4198
pp. **1285–1299**

# CONSTRUCTING CONTROL-FLOW PATTERNS CONTAINING INVISIBLE TASK AND NON-FREE CHOICE BASED ON DECLARATIVE MODEL

Kelly R. Sungkono and Riyanarto Sarno

Department of Informatics
Institut Teknologi Sepuluh Nopember
Jalan Raya ITS, Sukolilo, Surabaya 60111, Indonesia
kelsungkono@gmail.com; riyanarto@if.its.ac.id

Received December 2017; revised April 2018

Abstract. *Business processes can be modeled into declarative models and imperative models. The declarative model provides rules to describe the relation of flexible processes and the imperative model uses control-flow patterns to depict the relation of processes in detail. To correlate relations in both models, extended MINERful algorithm converts a declarative model into an imperative model in the form of Petri Net model. However, while forming an imperative model, extended MINERful cannot detect invisible task and invisible task in non-free choice relation. This research proposes a method to construct invisible task and invisible task in non-free choice relation by connecting discovered primary control-flow patterns that are constructed by combining rules in a declarative model. The primary control-flow patterns are patterns of sequence, XOR, AND and OR. All of those patterns are formed in linear temporal logic and a tree model as the form of an imperative model is composed of those patterns. The experiment shows that the proposed method can build an imperative model that contains invisible task and the invisible task of non-free choice based on a declarative model. This experiment verifies that the results of the proposed method have higher precision than results of the extended MINERful algorithm.*

**Keywords:** Declarative model, Imperative model, Linear temporal logic, Control-flow patterns

1. **Introduction.** A model established based on an event log helps analysts to describe and evaluate the business processes. Nowadays, analysts are facilitated with two kinds of models, such as declarative models and imperative models. The declarative model is a model that describes processes by several rules. The rules define relations of activities flexibly, so the declarative model can be easily arranged by analysts [1]. The imperative model uses control-flow patterns to define the relations of activities explicitly. The control-flow patterns depict the detail of the activities flows. Imperative models are widely utilized in many aspects, such as business [2-5], fraud [6,7], medical [8,9], and advertisement [10]. In order for a model to provide general information or detail information of processes, an algorithm that converts a declarative model into an imperative model is needed.

Extended MINERful algorithm [1] is an additional method in MINERful algorithm. It converts a declarative model into an imperative model in the form of Petri Net model. This method defines several rules of the declarative model and finds intersections between those rules to build an imperative model. The imperative model of extended MINERful can depict several primary control-flow patterns, such as patterns of sequence relation, XOR and AND relations. However, this method cannot discover other patterns, such as

DOI: 10.24507/ijicic.14.04.1285

invisible task and invisible task in non-free choice relations. This is because the method only observes connectivity between rules in the declarative model without considering the connectivity between discovered primary patterns.

The modeling of invisible tasks and invisible task in non-free choice depicts real processes in particular situations. For example, in Port Container Handling processes, there is a process that the customs immediately issued a release letter of commodities and overrides other activities. This applies if the commodities do not require multiple checks. Even though the process skips several activities, it is not anomaly or failure. Therefore, an invisible task is added in the model to provide a skip condition.

The second example is the determination of activities that are executed before the commodities are transported into trucks. There are three commodities that are arranged in Port Container Handling, such as the dry container, the uncontainer, and the reefer container. Unlike other commodities, the dry container is directly inserted into the truck. If a model sets several choice activities before transporting commodities into the truck, the process of the dry container will be the wrong process. An invisible task in non-free choice is used in a model to show the condition wherein a skipped process is allowed if the commodities are determined dry container.

This research proposes a method to construct the control-flow patterns, including patterns of the invisible task [11] and patterns of non-free choice relation [12] based on a declarative model and to arrange those patterns into a tree model. The proposed method forms the rules for obtaining the patterns by considering connectivity between rules of declarative model and connectivity between discovered primary patterns. The consideration of the connectivity between discovered primary patterns can establish patterns of invisible task and patterns of non-free choice relation. The rules or templates that are used in the declarative model are *chain_response, exclusive_choice*, *response* and *existence2*. The primary patterns that are constructed in this research are not the only patterns for determining XOR, sequence, AND relations, but also OR relations. Those discovered control-flow patterns are written in Linear Temporal Logic (LTL) and a tree model is composed of those patterns. Linear temporal logic is chosen because it is the formal language to describe the relation of components that are related to time and the tree model is chosen because it can be formed from logical operators that are contained within LTL.

This research evaluates the quality of process models. The process model is evaluated based on the two aspects, i.e., fitness and precision [13]. The discovered process model will be compared with an imperative process model by the extended MINERful algorithm. This research is constructed as follows. Section 2 presents literature review to clarify this research. Section 3 describes the proposed method for discovering and composing the patterns. Section 4 reports final outputs of the experimental process. The last section, Section 5, presents the conclusions of the research.

## 2. **Literature Review.**

### 2.1. **Invisible tasks and non-free choice.**

A good process discovery is process discovery that can accommodate various process conditions. Primary processes are sequential processes and parallel processes [4]. Parallel processes contain XOR relation, AND relation, and OR relation. In addition to these processes, there are processes that need to be considered. The processes are processes requiring invisible tasks and non-free choice relation.

Invisible tasks are additional tasks of a process model that helps clarify some process conditions. The conditions are skip condition, repetitive condition, and moving condition

[11]. The illustration of those conditions is shown Figure 1. Firstly, skip condition happens when several activities are skipped in the process. To differentiate between skip condition and skip processes as fraud or failure, the appearance of the process becomes the main benchmark. The process has skip condition if this type of process often appears in an event log, and the skip process is fraud or failure if this process rarely appears in an event log. In Figure 1, the percentage of appearance of processes [AC] is 66.7% in the log, so this process has skip condition. Therefore, an invisible task that is symbolized as a black box is added to connect activity A and activity C. Secondly, repetitive condition happens when several activities are executed repeatedly. This condition is commonly known as *redo condition*. In the illustration, activities B and C are executed repeatedly, so an invisible task is added to the process model. Thirdly, moving condition happens when the next activity of a choice activity is the next activity of other choice activities. Based on the log of moving condition in Figure 1, activity B as the choice activity has activity C as its next activity or activity F as its next activity; meanwhile, activity F is also the next activity of activity E as the other choice activities. To connect activity B and activity F, an invisible task is added.

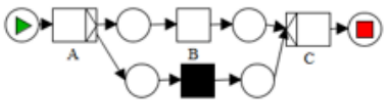
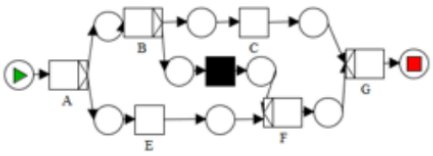
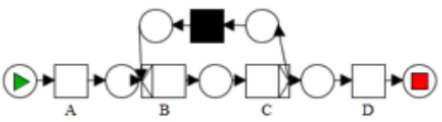
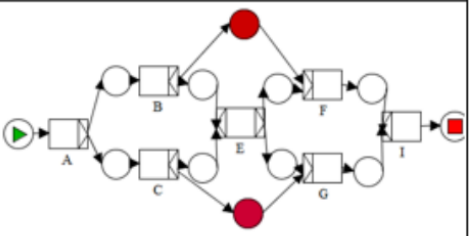
| Condition | Event Log | Process Model (YAWL) |
|---|---|---|
| Skip | [ABC], [AC], [AC] |  |
| Moving | [ABCG], [ABFG], [AEFG], [AEFG] |  |
| Repetitive | [ABCBCD], [ABCBCBCD] |  |
| Non-Free Choice | [ABEFI], [ACEGI], [ACEGI], [ABEFI] |  |

FIGURE 1. The illustration of invisible tasks and non-free choice

Non-free choice relation connects an activities in a choice relation with other activities in a previous choice relation. This relation shows that an activity of choice relation cannot be freely chosen, but it is influenced by the activity of the previous choice relation. The illustration is shown in Figure 1. Although activities F and G are the activities of choice relation, activity F is always executed if activity B was executed, and activity G is always executed if activity C was executed. Consequently, non-free choice relations that are symbolized as red circles in the process model of Figure 1 are added to describe that condition.

2.2. **Control-flow patterns.** Standardized pattern to explain executions of activities in the sequence and parallel conditions have been defined in a process model [14,15]. From

the twenty workflow patterns that are introduced, there are several patterns that are often applied to the process model. Those patterns are parallel split, exclusive choice, multi-choice, sequence, simple merge, synchronization, and synchronization merge.

Sequence pattern denotes activities that run sequentially. Parallel split and synchronization pattern are used to describe activities parallelly or describe AND relation in a process model. Simple merge pattern and also exclusive choice is used to signify the execution of one of the choice activities or describe XOR relation in a process model. Then, multi-choice and synchronization merge patterns are used to describe the execution of more than one choice activities or describe OR relation in a process model. Those patterns, apart from being chunks of a model, serve as error checkers on a process model that is depicted from the event log by constructing control-flow patterns from Standard Operational Procedure (SOP) process model [16].

Besides the primary patterns, this research presents additional patterns, i.e., non-free choice relation and patterns with invisible tasks. The primary patterns and additional patterns are depicted as YAWL format in Figure 2. The graphical patterns are illustrated by the pattern in YAWL format. The green circles are used as the start and the red circles are used as the ending. A split pattern only has green circles and a join pattern has red

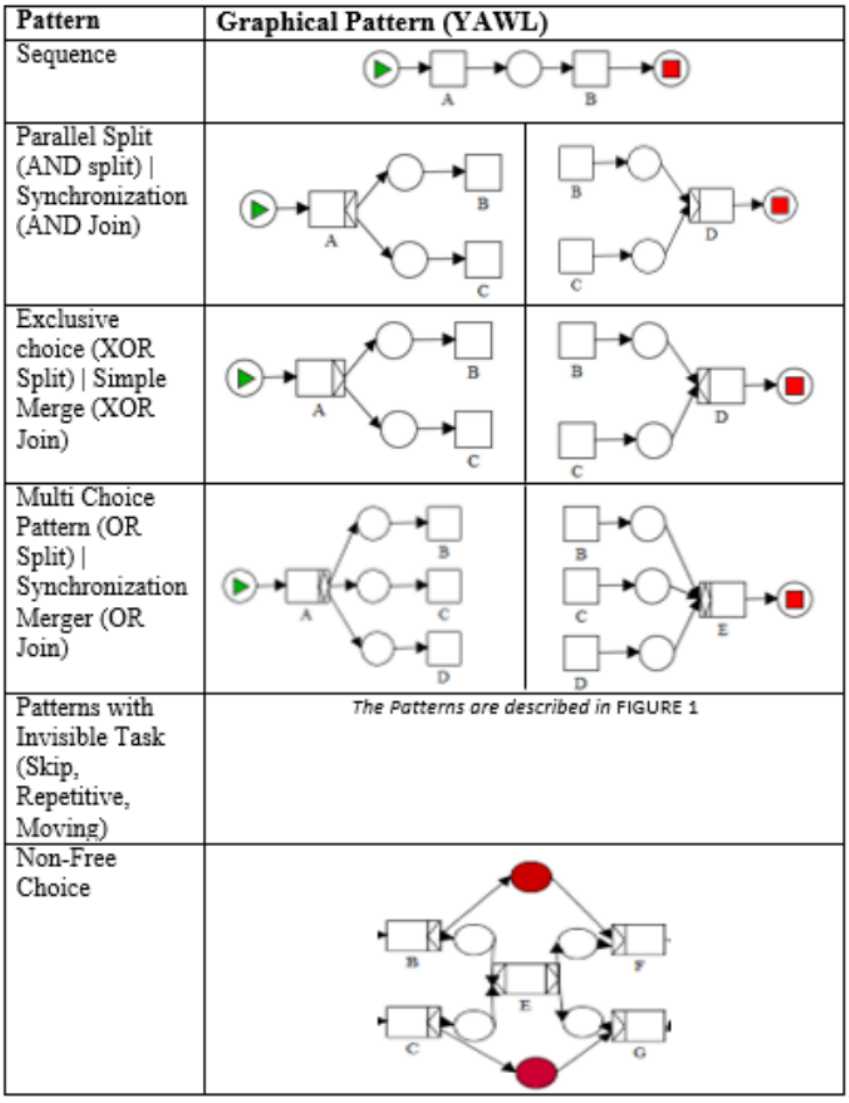



FIGURE 2. Primary and additional control-flow patterns

circles to denote that a split pattern must be placed at the beginning of join pattern to depict parallel relations. The non-free choice pattern does not have green circles or red circles because this pattern needs split and join pattern to describe a full relation.

This research constructs the control-flow patterns based on Declare templates by Declare Miner algorithm. In line with Declare templates that are formed in Linear Temporal Logic (LTL), this research proposes control-flow patterns in Linear Temporal Logic (LTL).

2.3. **LTL (Linear Temporal Logic) and Declare template.** LTL is a formal language that describes several temporal logics that refers to the time. LTL is constructed from constants (right and wrong), a group of atomic propositions, logical operators ($\neg$, $\vee$, $\wedge$, $\rightarrow$) and temporal capital operators. There are four temporal capital operators used in linear temporal logic. The explanation of four operators is described in Figure 3. The diagrams describe the events when the operators are used. For the first operator, $X$, it declares that inherent activities of the operator are executed at the next state. The first diagram in Figure 3 shows task $k$ is in a second circle as the next state. The second operator, $G$, declares that inherent activities of an operator can be on the whole following path. The third operator, $F$, declares that inherent activities of the operator can be executed at somewhere before the state of activities that are written before the operator. In the diagram, the activity that is written before the operator is symbolized as $\neg k$. The last operator, $U$, declares that the first inherent activity of the operator can be executed until the last inherent activity is executed. In the diagram, the first inherent activity is symbolized as $k$ and the second inherent activity is symbolized as $l$.

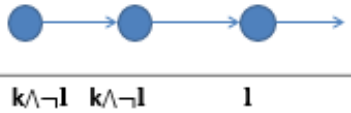
| Textual | Symbolic | Explanation | Diagram |
|---|---|---|---|
| Xk | ◯k | k has to hold at the next state |   |
| Gk | □k | k has to hold on the entire subsequent path |   |
| Fk | ◊k | k has to hold somewhere on the path |   |
| kUl | kUl | k has to hold until at some position l holds. |   |

FIGURE 3. The operators of linear temporal logic

The illustrations of usage of operators are given to clarify the understanding. The example of the first operator is *Registration -> X (Go to The Venue)*. It is said that each visitor must go to the venue right after registering. The example of the second operator is *G (Registration -> X (Go to The Venue))*. It is said that the activity registration and then go to the venue can happen in the first process, in the middle process, or the end of the process. The example of the third operator is *Registration -> F (Go to The Venue)*. It is said that after doing registration, each visitor can do other activities before going to the venue or going to the venue directly. The example of the last operator is *Registration U Starting Event*. This statement tells that the registration will continue to open until the event is started.

By extracting processes, LTL is formed into activity relations, that are called Declare templates or rules [17]. This template is used by Declare Miner algorithm [18] to form a

TABLE 1. Several declare templates

| Declare Template (Rules) | LTL | Description |
|---|---|---|
| *chain_response*(K1, S2) | $\Box(\text{K1} \rightarrow \text{O(R2)})$ | If activity K1 is recorded, then the next recorded activity is R2 |
| *exclusive_choice*(K1, S2) | $(\Diamond(\text{K1} \vee \text{R2}) \wedge !(\Diamond(\text{K1} \wedge \text{R2})))$ | Both of K1 and R2 recorded in the process, but they are recorded in different processes. |
| *response*(K1, S2) | $\Box(\text{K1} \rightarrow \Diamond(\text{R2}))$ | Activity R2 must be executed after activity K1 was executed. |
| *existence2*(K1) | $\Diamond(\text{K1} \wedge \text{O}(\Diamond(\text{K2})))$ | Activity K1 appears in the processes two times or more. |

model of cases of an event log. That model is described by a declare template graph. Table 1 shows Declare templates or rules, such as *chain_response, exclusive_choice, response*, and *existence2*. The statements in the LTL column of the table denote the LTL form of the rules or Declare templates. The description of all rules explains the processes that occur based on the LTL form. Those templates will be used in this research to construct the control-flow patterns.

2.4. **The quality of process model.** There are several aspects to measure the process model quality. This research chooses fitness and precision as the measurement aspects [13]. Fitness measures how many processes that are depicted in the model. Otherwise, precision measures how many traces from a model process are recorded in the model. The equations of precision and fitness calculation are given in Equation (1) and Equation (2). In Equation (1), variable $n(Cases\_Captured\_In\_Model)$ stores the number of cases in the event log that are depicted in the model and $n(Cases\_In\_Log)$ stores the amount of all cases of the event log. Otherwise, in Equation (2), variable $n(Traces\_OfModel\_Captured\_In\_Log)$ stores the amount traces of a model that are the same as cases in the event log and variable $n(Traces\_OfModel)$ stores the amount traces of a model. For example, there are four cases of an event log, i.e., [ABEFI], [ACEFI], [ABEGI], and [ACEGI]. Using a model of non-free choice relation in Figure 1, a fitness value of the model is 0.5 because only two cases, [ABEFI] and [ACEGI], out of four cases are depicted in the model. Then, a precision value of the model is 1.0 because all of the traces of the model, [ABEFI] and [ACEGI] are captured in the event log.

$$Fitness(k) = \frac{n(Cases\_Captured\_In\_Model)}{n(Cases\_In\_Log)} \tag{1}$$

$$Precision(k) = \frac{n(Traces\_OfModel\_Captured\_In\_Log)}{n(Traces\_OfModel)} \tag{2}$$

3. **Proposed Method.** There are two parts of the proposed method. The first step is discovering control-flow patterns based on the declarative model. The second step is composing the patterns into a tree model.

3.1. **Discovering control-flow patterns.** In discovering various patterns, several rules are proposed. The declare model that is used in this research contains *chain_response, exclusive_choice, response*, and *existence2*. The hierarchical steps of discovering the patterns are shown in Figure 4. The primary control-flow patterns are directly formed based on the rules in a declarative model. The examples are *synchronization* and *simple merge*. Based

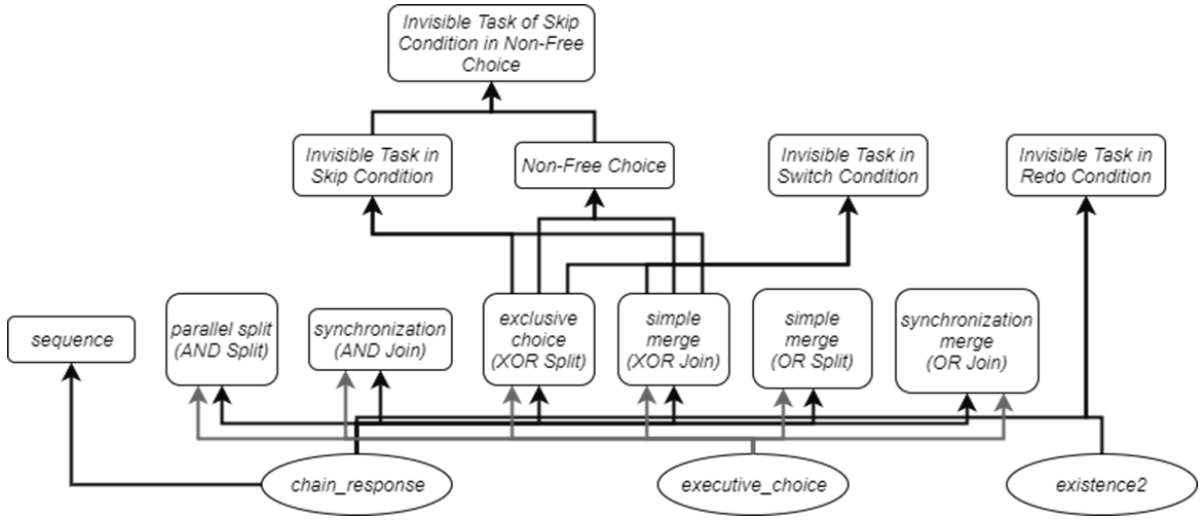



FIGURE 4. Hierarchy of dependence in pattern formation

on Figure 4, those patterns are formed by considering *exclusive_choice* and *chain_response* rules. Then, patterns of invisible tasks and non-free choice are created by considering other patterns. Invisible tasks in skip condition and invisible tasks in switch condition are created with a simple merge pattern or exclusive choice pattern if the beginning activity in exclusive choice pattern occurs as the beginning activity in simple merge pattern and vice versa. A non-free choice is created by considering the *exclusive_choice* of activities in exclusive choice pattern and activities in simple merge pattern. Lastly, invisible tasks in non-free choice are created by combining invisible task in skip condition and non-free choice.

The detailed method is explained in Table 2. In the method, the variables of total_next and total_before are used to determine the primary patterns, i.e., sequence and patterns for building XOR, OR and AND. Then, the variables of total_ex and total_chain are variables which are considered in the formation of split and join patterns. The results of all patterns are described in the LTL row of the table. There are two simple illustrations to describe the method. For the first illustration, there are *chain_response*(K, R), *chain_response*(A, K), *chain_response*(A, M). Those rules have more than zero support value. Task K has one total_next because it only becomes the initial component of one *chain_response*, and task A has two total_next because it is the initial component of two *chain_response*. From these results, there is a discovered sequence pattern. That is $k$ -> *O(R)*. In the second illustration, there are *chain_response*(A, K), *chain_response*(A, R), and *exclusive_choice*(K, R). The total_after of task A is two because there are two *chain_response* that has task A as its initial component. Then, total_ex_after of A is one because there is *exclusive_choice* between tasks K and R as the act_next of task A. Because there is total_chain_after(A), there is a discovered XOR Split, which is $A$ -> $O((K \vee R))$.

In Table 2:

Total_next(act): the amount of *chain_response*(act, other activities) which has more than zero support value

Total_before(act): the amount of *chain_response*(other activities, act) which has more than zero support value

Total_ex_after(act): the amount of *exclusive_choice*(act_next of act, act_next of act) which has more than zero support value

Total_ex_before(act): the amount of *exclusive_choice*(act_before of act, act_before of act) which has more than zero support value

TABLE 2. Method of constructing patterns

| **Pattern** | **Rule to build Patterns and LTL as results** |
|---|---|
| Sequence | total_next(act) == 1 |
| | LTL: act -> O(y) |
| Parallel Split (AND Split) | total_next(act) > 1 and total_ex(act) == 0 |
| | LTL: act -> ♢((y1 ∧ y2 ∧ ... ∧ yn)) |
| Synchronization (AND Join) | total_before(act) > 1 and total_ex(act) == 0 |
| | LTL: ♢ ((y1 ∧ y2 ∧ ... ∧ yn)) -> O(act) |
| Exclusive choice (XOR Split) | total_next(act) > 1 and total_ex_after(act) > 0 and (total_chain_after(act) < total_after(act)) |
| | LTL: act -> O((y1 ∨ y2 ∨ ... ∨ yn)) |
| Simple Merge (XOR join) | total_before(act) > 1 and total_ex_before(act) > 0 and (total_chain_before(act) < total_next(act)) |
| | LTL: O((y1 ∨ y2 ∨ ... ∨ yn)) -> O(act) |
| Multi Choice Pattern (OR Split) | total_next(act) > 1 and total_ex_after(act) > 0 and (total_chain_after(act) >= total_next(act)) |
| | LTL: act -> ♢((y1 ∨ y2 ∨ ... ∨ yn)) |
| Synchronization Merger (OR Join) | total_before(act) > 1 and total_ex_before(act) > 0 and (total_chain_before(act) >= total_next(act)) |
| | LTL: ♢((x1 ∨ x2 ∨ ... ∨ xn)) -> O(act) |
| Non-Free Choice Relation | act in Exclusive Choice and act in Simple Merge and total_ex_aftbef(act) < amount(act_before)$^2 - 1$ |
| | LTL: ♢((x1 ∧ act ∧ k) ∨ (x2 ∧ act ∧ k) ∨ ... ∨ (xn ∧ act ∧ k)), where x1, ..., xn ∈ act_before of act, $k$ = act_next in *exclusive_choice*(x, act_next) which has minimum support value |
| Invisible Task in Skip and Switch Condition | act appears in Exclusive Choice before "->" and appears in Simple Merge before "->" |
| | LTL: Change O((act ∨ y2 ∨ ... ∨ yn)) -> O(other_act) into O((Invisible_Task ∨ y2 ∨ ... ∨ yn)) -> O(other_act) |
| | act appears in Exclusive Choice after "->" and appears in Simple Merge after "->" |
| | LTL: Change other_act -> O((act ∨ y2 ∨ ... ∨ yn)) into other_act -> O((Invisible_Task ∨ y2 ∨ ... ∨ yn)) |
| Invisible Task in Redo Condition | act appears in LTLSequence before "->" and act not in *existence2* and act_after(act) in *existence2* |
| | LTL: act -> ♢((act_after1 ∧ O(act_after2) ∧ ... ∧ O(act_after_n))) |
| Invisible Taks in Non-Free Choice Relation | act appears in Simple Merge after "->" and act appears in Non-Free Choice Relation |
| | LTL: Change ♢((x1 ∧ other_act ∧ act) ∨ (x2 ∧ other_act ∧ k)) into ♢((x1 ∧ other_act ∧ Invisible_Task) ∨ (x2 ∧ other_act ∧ k)) |
| | act appears in Simple Merge before "->" and act appears in Non-Free Choice Relation > 1 |
| | LTL: Change O((y1 ∨ act ∨ ... ∨ yn)) -> O(other_act) into O((y1 ∨ Invisible_Task ∨ ... ∨ yn)) -> O(other_act) |

Total_chain_after(act): the amount of *chain_response*(act_next of act, act_next of act) which has more than zero support value

Total_chain_before(act): the amount of *chain_response*(act_before of act, act_before of act) which has more than zero support value

Total_ex_aftbef(act): the amount of *exclusive_choice*(act_before of act, act_next of act) which has more than zero support value

Besides those patterns, the first activity and the last activity are also identified along with the control-flow patterns in LTL. The first activity is an activity that does not have total_before and the last activity is an activity that does not have total(after). The first activity is symbolized as FirstActivity(activity) and the last activity is symbolized as LastActivity(activity).

3.2. **Building a tree model.** A tree model is composed of all discovered patterns. The method for composing the patterns is explained in Table 3. The method provides rules to convert each LTL that is described as variable LTL. The first step to build the model is choosing the first LTL. The first LTL is a control-flow pattern that has activity in FirstActivity(activity). Then, this LTL will be split into a list of symbols and activities. For example, *Registration -> O*((*Go to The Venue* $\vee$ *Waiting The Others*)) can be split as

Table 3. A method of composing tree model based on patterns

| No | Rules |
|---|---|
| 1 | LTL = Control-Flow Pattern in LTL that has activity in FirstActivity(activity) |
| 2 | While all LTL are modeled: |
| 3 | Split LTL into parts that are called part_LTL |
| 4 | For part_LTL in LTL: |
| 5 | If part_LTL is "(": |
| 6 | *Make a child of node and the position is in the child of node* |
| 7 | Else if part LTL is ")": |
| 8 | *Position is in the parent of node* |
| 9 | Else if part LTL is "O": |
| 10 | If node of this position is "$\vee$": |
| 11 | node = $\circlearrowleft$ |
| 12 | Else: |
| 13 | *node = "O" and the bracket beside this part is not executed* |
| 14 | Else if part LTL is "$\diamondsuit$" |
| 15 | *node =* $\diamondsuit$ *and the bracket beside this part is not executed* |
| 16 | Else if part LTL is "$\vee$": |
| 17 | If node of this position is "O": |
| 18 | *node = "x"* |
| 19 | If node of this position is "<>": |
| 20 | *node =* "$\vee$" |
| 21 | Else if part LTL is "$\wedge$" or "->": |
| 22 | If node of this position is "x" or "$\vee$": |
| 23 | *Add child node and fill with "->" or "$\wedge$"* |
| 24 | *All of child of node "x" or "v" become child of node "->" or "$\wedge$"* |
| 25 | Else: |
| 26 | *Fill the node name with the name of part_LTL position is in the parents* |
| 27 | LTL = LTL which has activity that are also on previous LTL |

Part_LTL = [*Registration*, ->, *O*, (, (, *Go to The Venue*, ∨, *Waiting The Others*, ), ) ]. A Part_LTL of the first LTL is executed from the first part in part_LTL, but the execution of other Part_LTLs is started from the first activity in Part_LTL. The tree model is formed when all of LTL are already processed.

In the processing of Part_LTL, an open parenthesis is used to add a child of a node and closing parenthesis is used to return the position to the parent of that node. The other symbols are used to determine the symbols of relations in the tree model. There are five symbols that are used in the tree model, i.e., "x" for XOR Relation, "∨" for OR relation, "^" for AND relation, "->" for sequence relation and "↺" for redo condition.

4. **Result and Analysis.** The proposed method is already implemented in programs and is evaluated using import processes of Port Container Handling in January 2016. The processes that are used do not contain failure processes. This import process consists of three big parts. The parts are Discharge, Customs with Quarantine, and Delivery. Discharge part contains activities from the vessels berthing until the stacking of commodities in the yard. Customs with Quarantine part contain activities that are performed in the customs (*behandle*) area or the quarantine area. The Delivery part contains activities from Customs with Quarantine part until the commodity carrier trucks go out from Port Container Handling. The types of handled commodities are the dry container, the reefer container, and the uncontainer. This experiment uses Delivery part and Customs with Quarantine part separately to evaluate the proposed method.

A declarative model of Delivery part by Declarative Miner algorithm [17] is shown in Figure 5. In the model, *exclusive_choice* and *chain_response* rules are displayed. The colors of the rules indicate high or low support values of the rules. The color will fade if the support value is low, and vice versa. The color of *chain_response* between *Determine_Container_Type* and *Determining_Uncontainer* is more faded than the color of *chain_response* between *Determine_Container_Type* with other activities. It indicates

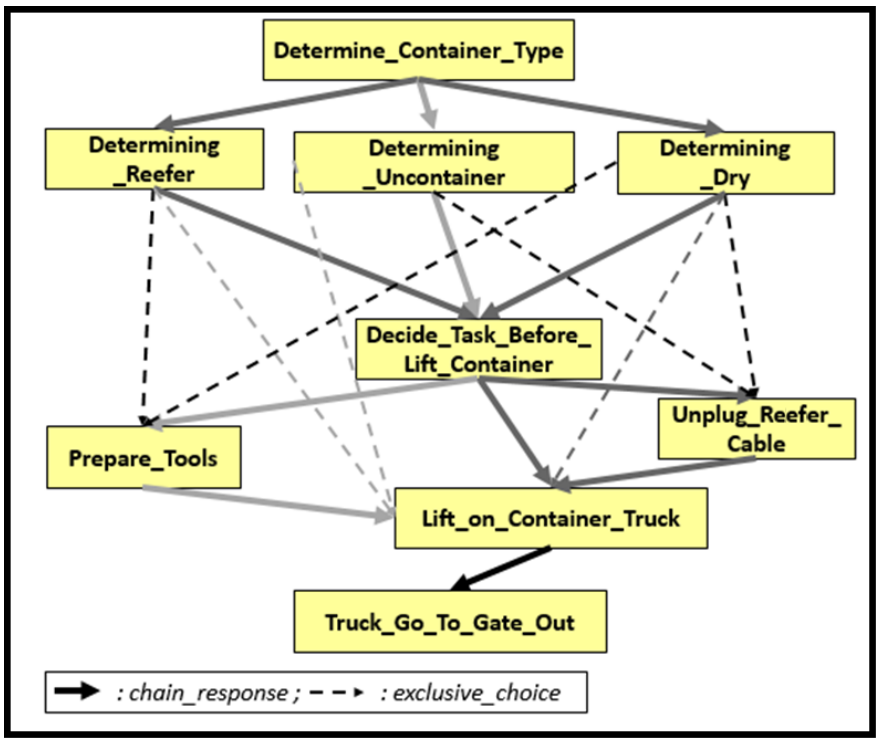



FIGURE 5. Declarative model of Delivery part process

*chain_response* between *Determine_Container_Type* and *Determining_Uncontainer* has the lowest support value among all *chain_response* rules involved *Determine_Container_ Type*. The rules which have zero support value are not displayed in the model.

Based on this model, the discovered control-flow patterns are automatically formed based on the declarative model in Figure 5. The result of discovered patterns is shown in Figure 6. There are four discovered patterns, i.e., (1) non-free choice, (2) sequence, (3) split in XOR relation and (4) join in XOR relations. *Determining_Reefer* $\wedge$ *Decide_Task_Before_Lift_Container* $\wedge$ *Unplug_Reefer_Cable* in LTL of control-flow patterns indicates a non-free choice relation between *Determining_Reefer* and *Unplug_Reefer_Cable*. The non-free choice relation occurs because the *exclusive_choice* rule of those activities has zero support value that is the lowest support value of *exclusive choice* rules involved *Determining_Reefer*. In the case of *Determining_Dry*, the *exclusive_choice* of this activity and *Lift_on_Container_Truck* has the lowest support among all of *exclusive_choice* involved *Determining_Dry*. Because *Lift_on_Container_Truck* is the next activity of *Unplug_Reefer_Cable* and *Prepare_Tools*, *Lift_on_Container_Truck* is changed to *Invisible_Task* and a pattern of non-free choice has *Determining_Dry* $\wedge$ *Decide_Task_Before_Lift_Container* $\wedge$ *Invisible_Task*.

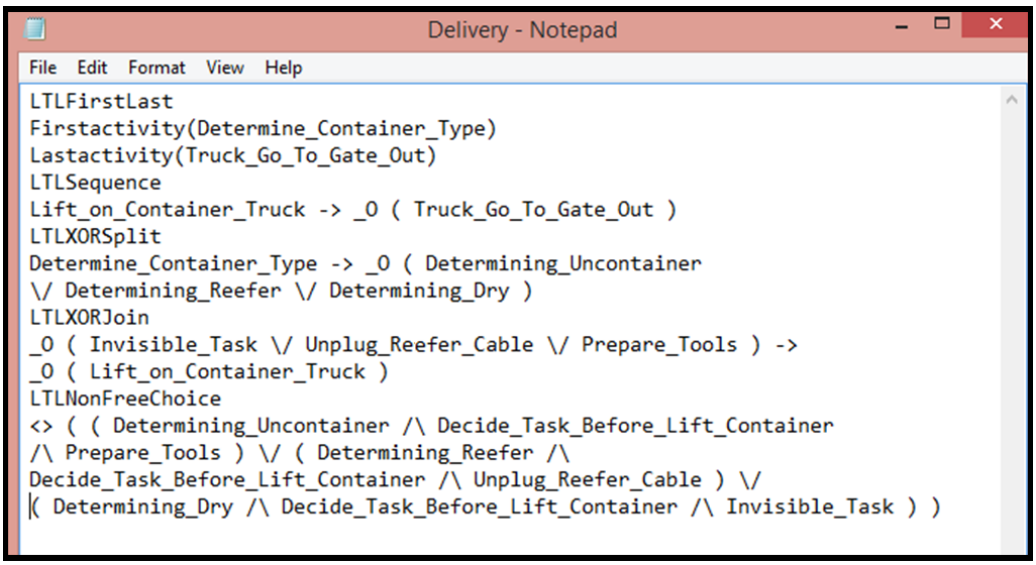



FIGURE 6. Control-flow patterns in LTL form of Delivery part process

The last step of the method is composing the discovered control-flow patterns in a tree model. The discovered tree model is depicted in Figure 7. The tree model is read from the top to the bottom of the model. The lines that connect the activities and the operators show the connections between parent nodes and child nodes. For example, the operator -> has connections with *Determine_Container_Type*, operator $x$, *Lift_on_Container_Truck*, and *Truck_Go_To_Gate_Out*. Because operator -> is a base of those connections, operator -> is a parent node and the others are child nodes of the operator ->. The operator -> chooses all of its child nodes that are taken sequentially; meanwhile, the operator $x$ chooses one of its child nodes.

Using processes in Delivery part, the model obtained by the extended MINERful algorithm is shown in Figure 8. The difference between the model obtained by the proposed method and the model obtained by the extended MINERful is non-free choice relations. The extended MINERful algorithm does not describe the non-free choice between *Determining_Uncontainer* and *Prepare_Tools* and the non-free choice between *Determining_Dry*

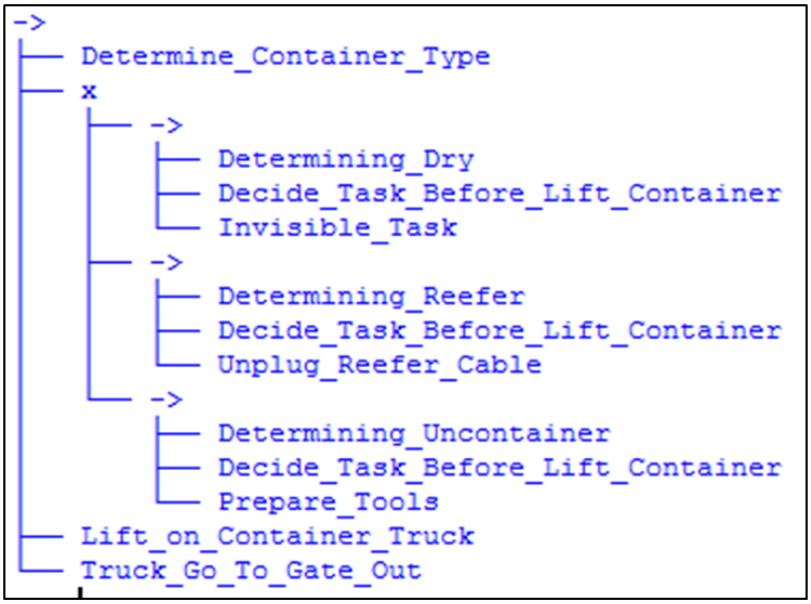



FIGURE 7. Tree model of Delivery part by the proposed method

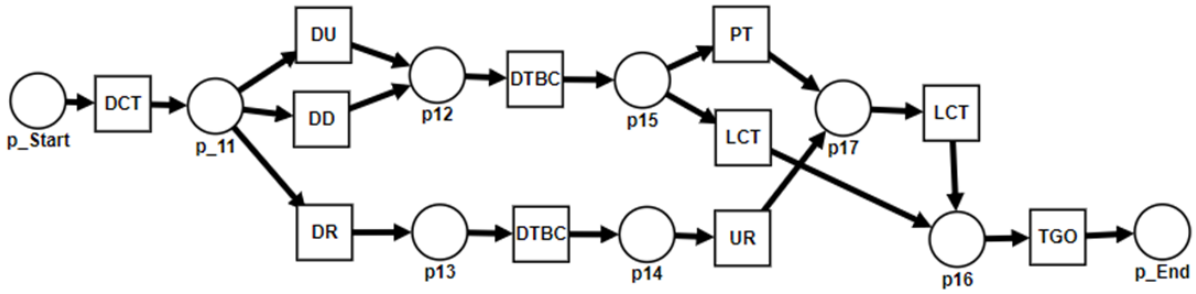



FIGURE 8. Petri Net of Delivery part by extended MINERful

and *Lift_On_Container_Truck*. It is because that algorithm cannot detect a pattern of the invisible task in non-free choice. Using processes of Customs with Quarantine part as the data for the second evaluation, the model processes by the proposed method and by the extended MINERful are displayed in Figure 9 and Figure 10. Activities on Petri Net constructed by MINERful use alias names. Real names of those activities are in Table 4. The differences between models by the proposed method and extended MINERful algorithms are activity *Verification_Document_Behandle* and *Create Document SPBB*. Because extended MINERful cannot detect invisible tasks, those activities are defined redundantly. Otherwise, the proposed method can detect invisible task in skip condition, so those redundant activities can be changed into invisible tasks.

All those process models are measured by the fitness and the precision. The measurement aspects of those results are presented with a chart of Figure 11. Although the models by the proposed method and extended MINERful have high fitness, the model of Delivery part by the extended MINERful is a wrong model based on the log of Delivery part. It is because the model has a low precision which indicates several traces of the model are not stored in the log of Delivery part. Conversely, the models of the proposed method have high fitness and high precision that indicate the proposed method provides good models, both in terms of fitness aspect and precision aspect.

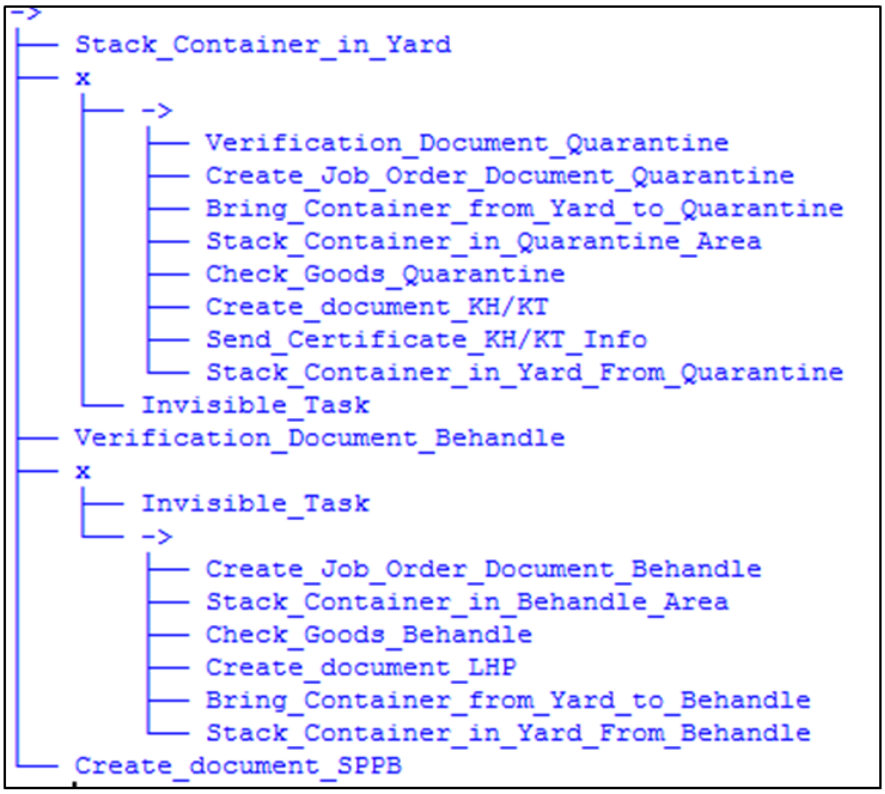



FIGURE 9. A tree model of Customs with Quarantine part by the proposed method

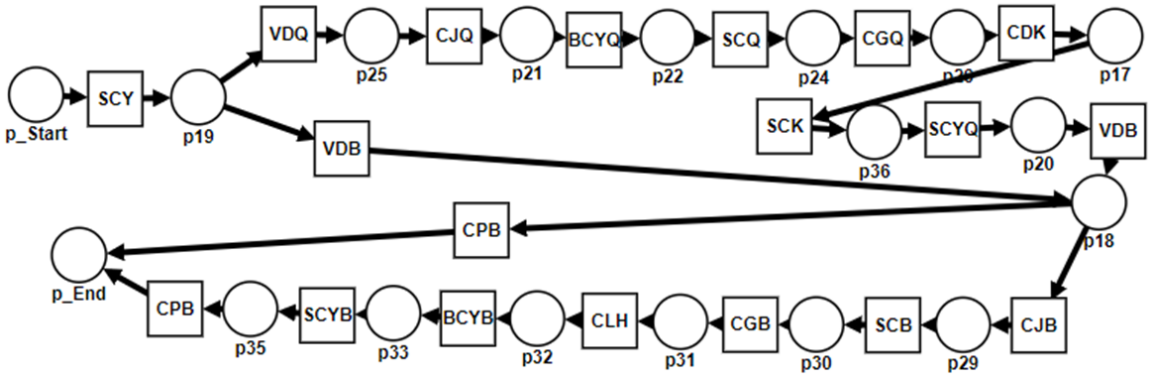



FIGURE 10. Petri Net of Customs with Quarantine part by extended MINERful

5. **Conclusions.** This research proposes a method for constructing the control-flow patterns, including patterns of invisible tasks and non-free choice, based on the rules in a declarative model and composing those patterns into an imperative model. This research does not only consider the connection between rules in the declarative model but also consider the discovered pattern based on the rules to build other patterns.

There are two main steps in the proposed method of this research. Firstly, this research provides several rules to construct the control-flow patterns. The primary patterns, such as patterns for building XOR, sequence, OR, and AND relations, are formed based on several rules in the declarative model. Patterns of invisible tasks and invisible task in non-free choice are built based on the discovered primary patterns. Lastly, a tree model is composed of the discovered patterns to describe the whole processes.

TABLE 4. List of activity names initialized in Petri Net MINERful

| Alias Task names | Real Task Names | Alias Task names | Real Task Names |
|---|---|---|---|
| DCT | Determine_Container_Type | CGQ | Check_Goods_Quarantine |
| DD | Determine_Dry | CDK | Create_document_KH/KT |
| DR | Determine_Reefer | SCK | Send_Certificate_KH/KT_Info |
| DU | Determine_Uncontainer | SCYQ | Stack_Container_in Yard_From_Quarantine |
| DTBC | Decide_Task_Before Lift_Container | VDB | Verification_Document_Behandle |
| UR | Unplug_Reefer_Cable | CJB | Create_Job_Order Document_Behandle |
| PT | Prepare_Tools | SCB | Stack_Container _in_Behandle_Area |
| LCT | Lift_on_Container_Truck | CGB | Check_Goods_Behandle |
| SCY | Stack_Container_in_Yard | CLH | Create_document_LHP |
| VDQ | Verification_Document_Quarantine | BCYB | Bring_Container_from Yard_to_Behandle |
| CJQ | Create_Job_Order_ Document_Quarantine | SCYB | Stack_Container_in Yard_From_Behandle |
| BCYQ | Bring_Container_ from_Yard_to_Quarantine | CPB | Create_document_SPPB |
| SCQ | Stack_Container_in Quarantine_Area | TGO | Truck_Go_To_Gate_Out |

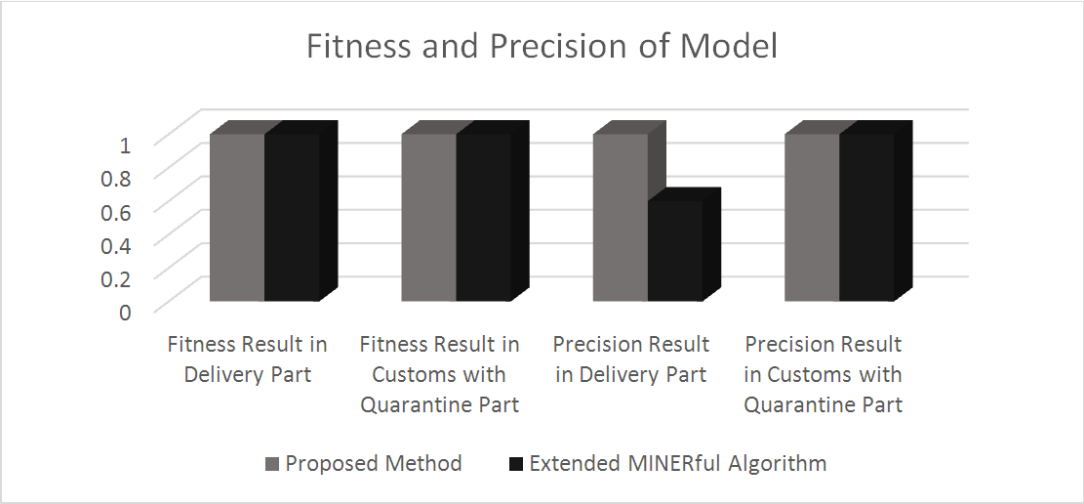



FIGURE 11. The fitness and precision of model by the proposed method and extended MINERful

The evaluation of this experiment used data from Port Container Handling. The results of evaluation verify the models by the proposed method are better than models by the extended MINERful algorithm. This is indicated by the precision of the models using the proposed method is higher than that using extended MINERful. The high precision of the model of the proposed method is supported by the capability of the method in the detection of invisible tasks and invisible tasks in non-free choice. This method uses several rules of the declarative model and it is implemented in separate programs. Therefore, the development of this research is the utilization of other rules of the declarative model to construct control-flow patterns and the establishment of a complete program of the proposed method.

**Acknowledgment.** Institut Teknologi Sepuluh Nopember and The Ministry of Research, Technology and Higher Education of Indonesia support this research. The authors admire the supportive comments or suggestions by the reviewers, in improving the presentation.

## REFERENCES

[1] J. Prescher, C. D. Ciccio and J. Mendling, From declarative processes to imperative models, *CEUR Workshop Proceedings*, vol.1293, pp.162-163, 2014.

[2] R. Sarno, C. A. Djeni, I. Mukhlash and D. Sunaryono, Developing a workflow management system for enterprise resource planning, *Journal of Theoretical & Applied Information Technology*, vol.72, no.3, pp.412-421, 2015.

[3] R. Sarno and K. R. Sungkono, Coupled hidden Markov model for process mining of invisible prime tasks, *International Review on Computers and Software (IRECOS)*, vol.11, no.6, pp.539-547, 2016.

[4] R. Sarno and K. R. Sungkono, Hidden Markov model for process mining of parallel business processes, *International Review on Computers and Software (IRECOS)*, vol.11, no.4, pp.290-300, 2016.

[5] R. Sarno and K. R. Sungkono, Coupled hidden Markov model for process discovery of non-free choice and invisible prime tasks, *Procedia Computer Science*, vol.124, pp.134-141, 2018.

[6] R. Sarno, R. D. Dewandono, T. Ahmad, M. F. Naufal and F. Sinaga, Hybrid association rule learning and process mining for fraud detection, *IAENG International Journal of Computer Science*, vol.42, no.2, pp.59-72, 2015.

[7] S. Huda, R. Sarno and T. Ahmad, Increasing accuracy of process-based fraud detection using a behavior model, *International Journal of Software Engineering and Its Applications*, vol.10, no.5, pp.175-188, 2016.

[8] W. Chomyat and W. Premchaiswadi, Process mining on medical treatment history using conformance checking, *The 14th International Conference on ICT and Knowledge Engineering (ICT&KE)*, pp.77-83, 2016.

[9] T. Erdogan and A. Tarhan, Process mining for healthcare process analytics, *Software Measurement and the International Conference on Software Process and Product Measurement (IWSM-MENSURA)*, pp.125-130, 2016.

[10] A. S. Osses, L. Q. D. Silva, B. F. Cobo, M. Arias et al., Business process analysis in advertising: An extension to a methodology based on process mining projects, *The 35th International Conference of the Chilean Computer Science Society (SCCC)*, pp.1-12, 2016.

[11] L. Wen, J. Wang, W. M. P. van der Aalst, B. Huang and J. Sun, Mining process models with prime invisible tasks, *Data & Knowledge Engineering*, vol.69, no.10, pp.999-1021, 2010.

[12] L. Wen, W. M. P. van der Aalst, J. Wang and J. Sun, Mining process models with non-free-choice constructs, *Data Mining and Knowledge Discovery*, vol.15, no.2, pp.145-180, 2007.

[13] J. C. A. M. Buijs, B. F. van Dongen and W. M. P. van der Aalst, On the role of fitness, precision, generalization and simplicity in process discovery, *International Conference on OTM Federated Conferences*, vol.7565, no.1, pp.305-322, 2012.

[14] N. Mulyar, N. Russell, A. H. M. Ter Hofstede and W. M. P. van der Aalst, Towards a WPSL?: A critical analysis of the 20 classical workflow control-flow patterns, *BPM Reports*, pp.1-66, 2006.

[15] N. Russell, A. H. M. Ter Hofstede, W. M. P. van der Aalst and N. Mulyar, Workflow control-flow patterns: A revised view, *BPM Center Report*, vol.2, pp.6-22, 2006.

[16] R. Sarno, W. A. Wibowo, K. Kartini, Y. A. Effendi and K. R. Sungkono, Determining model using non-linear heuristics miner and control-flow pattern, *TELKOMNIKA (Telecommunication, Computing, Electronics and Control)*, vol.14, no.1, pp.349-360, 2016.

[17] F. M. Maggi, A. J. Mooij and W. M. P. van der Aalst, User-guided discovery of declarative process models, *IEEE Symposium on Computational Intelligence and Data Mining (CIDM)*, pp.192-199, 2011.

[18] F. M. Maggi, Declarative process mining with the declare component of ProM, *CEUR Workshop Proceedings*, vol.1021, 2013.