Big Data and Information Analytics
©American Institute of Mathematical Sciences
Volume **2**, Number **2**, April **2017**
doi:10.3934/bdia.2017016
pp. **177–189**

AN ONTOLOGICAL ACCOUNT OF FLOW-CONTROL COMPONENTS IN BPMN PROCESS MODELS

Xing Tan*

Information Retrieval and Knowledge Management Research Lab
School of Information Technology, York University, Toronto, ON, Canada

Yilan Gu

TD Bank Financial Group, 66 Wellington Street
ON, M5K 1A2, Canada

Jimmy Xiangji Huang

Information Retrieval and Knowledge Management Research Lab
School of Information Technology, York University, Toronto, ON, Canada

(Communicated by Aijun An)

Abstract. The Business Process Model and Notation (BPMN) has been widely adopted in the recent years as one of the standard languages for visual description of business processes. BPMN however does not include a formal semantics, which is required for formal verification and validation of behaviors of BPMN models.

Towards bridging this gap using first-order logic, we in this paper present an ontological/formal account of flow-control components in BPMN, using Situation Calculus and Petri nets. More precisely, we use SCOPE (Situation Calculus Ontology of PEtri nets), developed from our previous work, to formally describe flow-control related basic components (i.e., events, tasks, and gateways) in BPMN as SCOPE-based procedures. These components are first mapped from BPMN onto Petri nets.

Our approach differs from other major approaches for assigning semantics to BPMN (e.g., the ones applying communicating sequential processes, or abstract state machines) in the following aspects. Firstly, the approach supports direct application of automated theorem proving for checking theory consistency or verifying dynamical properties of systems. Secondly, it defines concepts through aggregation of more basic concepts in a hierarchical way thus the adoptability and extensibility of the models are presumably high. Thirdly, Petri-net-based implementation is completely encapsulated such that interfaces between the system and its users are defined completely within a BPMN context. Finally, the approach can easily further adopt the concept of time.

1. **Introduction.** The Business Process Model and Notation (BPMN) [15] has been widely adopted in recent years as one of the standard languages for visual description of business processes. In particular BPMN has been extensively used in facilitating effective communication among technical experts, business users and

2010 *Mathematics Subject Classification.* Primary: 58F15, 58F17; Secondary: 53C35.
Key words and phrases. Dimension theory, Poincaré recurrences, multifractal analysis.
The first and third authors are supported by NSERC CREATE ADERSIM.
* Corresponding author: Xing Tan.

other stakeholders within a work group, for business process design and implementation, or problem solving in business domains.

BPMN however does not contain a formal semantics. Consequently, rigorously specifying dynamical behaviors of objects in a BPMN specified business process model can be challenging and error-prone. For critical systems (e.g., a medical process specified in BPMN), the existence of semantic ambiguities in these systems is costly from every perspective and thus should be eliminated as much as possible. For these reasons, in recent years, several efforts have been made to assign BPMN with formal semantics. Among them, [3] and [4] in particular proposed to map a BPMN model into a Petri net, which is a mathematical modeling language (although a Petri net can be graphical too) that has precise execution semantics. In addition, decades of research in Petri nets offers well-established theories to support analysis of dynamic properties in Petri-net specified systems. In [3] and [4], a given BPMN model is first dissected into several BPMN elements. For each type of the element, a mapping from the element to a Petri net module is proposed. After the dissection step and mapping step, these Petri nets modules are eventually integrated together to construct a single Petri net.

In our previous work ([17], [18], [19]) meanwhile we presented a straightforward first-order-logic (FOL) based framework for Petri nets. More specifically, we have shown that: a) we can represent Petri nets and their variants as Situation-Calculus[1] based action theories, and the theories for basic Petri nets are called in short as SCOPE, standing for Situation Calculus Ontology for PEtri nets ([17], [18]); b) given an instance of SCOPE theory, we could further build up Golog procedures[2], where sequential, iterative, or nondeterministic composite sequences of transition firings in Petri nets and their variants can be further axiomatized through defining macros; these macros are built on top of more primitive procedures and the only basic action in the domain *transition firing* [19]; c) executability testing of Golog procedures via theorem-proving can be implemented through efficient logic programming in Prolog ([18], [19]).

In this paper, we show that the mapping method as proposed in ([3] and [4]) can be naturally extended into SCOPE context. That is, using SCOPE, we can easily define a Petri net module, resulted from performing mapping on a BPMN object, as a Golog complex procedure. These Golog procedures act as interfaces encapsulating Petri net from being directly accessed by end users. Consequently, stakeholders can still work at BPMN level, while the model is enriched with a Petri-net specified formal semantics. Additionally, with these procedures, users can write more complicated user-defined BPMN procedures in a hierarchical way. Since a BPMN procedure can be further translated into Prolog programs [19], our approach support automated verification of important system dynamic properties, such as the executability of these procedures.

The remainder of this paper is organized as follows. Section 2 provides introductory reviews to the key technologies employed in this paper. Enriched with an example, Section 3 is the core section that explains our approach. Conclusive remarks are presented in Section 4.

[1]The Situation Calculus ([10, 16]) is a particular formal language for describing changes upon actions in dynamical domains.

[2]In the Situation Calculus, a Golog procedure defines a complex action that is composed of basic actions.

2. **Preliminaries.** In this section, we give a brief introduction to BPMN and Petri Nets. The method of mapping from BPMN onto Petri Nets as stated in Section 3 of [4] is then briefly reviewed. We also introduce here Situation Calculus/Golog and SCOPE.

2.1. **Petri nets & BPMN.** Business Process Model and Notation (BPMN) is a notational system for graphical specification of processes in business models. BPMN was developed and has been maintained by the Object Management Group (OMG) [15]. Control-flow related elements in BPMN are the focus of this paper and they include objects of *events*, *tasks* and *gateways*.

An event denotes happening of something in a model and is represented as a circle. In particular, a *start/end event* signals the start/finish of a process. An activity denotes work to be done in a model and is represented as a rounded-corner rectangle. A *task* is an atomic activity. Gateways are diamond-shaped quadrilaterals and they denote different routing constructs. Among them, a *fork* gateway (AND-split) creates concurrent flows and a *join* gateway synchronises these flows. A *decision* gateway (XOR-split) selects exactly one flow from a set of them based on certain criteria (data or event) and a *merge* gateway (XOR-join) merges the alternative flows in the set into one flow. When the choice is non-unique, the pair of gateways OR-split and OR-join is applied instead. Graphically these BPMN objects are connected to each other by arrows. Figure 2 is an example BPMN-denoted business process (adapted from Fig. 13 (a) in [4]).

A Petri net (PN) [14] is a pair (N, M_0), where N is a triple (P, T, F) such that P is a finite set of node elements called places, T is a finite set of node elements called transitions, $F \subseteq (P \times T) \cup (T \times P)$ consists of ordered pairs, and M_0 the initial marking, is a mapping in the form $M\colon P \to \mathcal{N}$, indicating the initial assignment of a non-negative integer k to each place p in P. (In this case, we say that the place p is marked with k tokens.) A marking M for N in Petri net PN is defined as a vector $(M(p_1), \ldots, M(p_m))$, where $p_1, \ldots, p_m$ is an enumeration of P and $M(p_i)$ tokens are assigned to node p_i, for all i such that $1 \leq i \leq m$. The same process, depicted in Figure 2 as a BPMN model, is presented as a Petri net in Figure 3.

The method of mapping a BPMN process into a Petri net introduced in [4] assumes that the BPMN process is in the first place "well-formed". A well-formed BPMN should satisfy several conditions (e.g., a start event has one outgoing flow and no incoming flow.). It is shown that a given BPMN process can always be transformed into a well-formed one ([1, 9, 25, 26]). The mapping process by itself is straightforward and are summarized in Figure 1 (Several issues are identified and addressed accordingly in [4]; however they are not discussed here). In the figure, note that places in the Petri net modules/components are dashed, meaning that these places would act as connectors to link multiple components into a Petri net through mapping. As an example, the Petri net in Figure 3 is resulted from performing mapping on the BPMN model in Figure 2.

2.2. **Situation Calculus & SCOPE.** The *Situation Calculus* is a logical language for representing actions and changes in a dynamical domain. It is first proposed by McCarthy and Hayes in 1969 [10]. The language $\mathcal{L}$ of Situation Calculus as stated by [16] is a second-order many-sorted language with equality. Any Situation Calculus-based action theory include: *Fluent conditions*, whose actual values can be updated from applications of relevant actions; *Actions*, applications of which will bring effects in term of changes of values of certain fluent conditions in a system. The

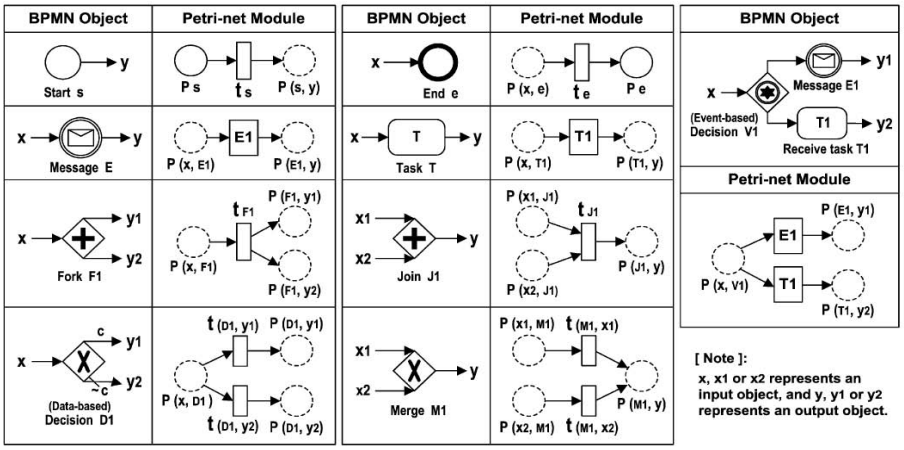

FIGURE 1. Mapping tasks, events, and gateways onto Petri-net components (Fig. 3. in [4] is copied here).

initial situation is called S_0. Starting from S_0, the system evolves on a sequence of actions into its future situations. *Golog* is a logic programming language for description and execution of complex actions using domain-specific Situation Calculus primitive actions. It provides imperative programming constructs, including (1) a, a primitive action; (2) $\alpha;\beta$, action α is followed by action β; (3) $p?$, test action on the condition p; (4) **if** p **then** α **else** β, conditionals; (5) $\alpha|\beta$, nondeterministic choice of action α or action β; (6) $(\pi x)\alpha(x)$, nondeterministic choice of arguments; (7) $\alpha^\star$, nondeterministic iteration; and (8) Procedures. The semantics of Golog programs is defined on the abbreviation $Do(\delta, s_1, do(\vec{a}, s_1))$, which denotes that execution of Golog program δ in the situation s_1 leads to $do(\vec{a}, s_1))$, an abbreviation to for the situation of performing a sequence of actions $\vec{a} = [\alpha_1, \ldots, \alpha_{n-1}, \alpha_n]$ starting from s_1, i.e., $do(\alpha_n, do(\alpha_{n-1}, \ldots, do(\alpha_1, s_1)\ldots))$. The structure of δ is defined inductively through macro-expansions on the above eight constructs.

The theory $\mathcal{D}_{scope}$ for Situation Calculus Ontology of PEtri nets (SCOPE) is first proposed in [17] and [18]. In $\mathcal{D}_{scope}$, the only action is *fire* and the only fluent is *Tkns*.

Primitive Action $fire(t)$**:** the transition t fires.
Fluent $Tkns(p, s)$**:** the number of tokens at place p at situation s.

Situation-Independent relations *pre* and *post* are introduced to specify the topology of a given Petri net.

- $pre(m, n)$. Node m enters node n.
- $post(m, n)$. Node n enters node m.
- $pre(m, n) \equiv post(n, m)$.

The Foundational Axioms $\mathcal{D}_f$ (not listed here) characterize a generic situation tree for any Basic Action Theory (BAT) written in the Situation Calculus. Based on $\mathcal{D}_f$, the Primitive Action Precondition Axiom for primitive actions in $\mathcal{D}_{scope}$

$$(\forall s, p, t)\,(Poss(fire(t), s) \equiv (pre(p, t) \supset Tkns(p, s) \geq 1))$$

defines the condition for a transition node t to fire legally. That is, the transition t is enabled to fire at situation s iff each place that enters the transition node t

contains at least one token.The Successor State Axiom in $\mathcal{D}_{scope}$ defines the effects firing of a transition node would bring to the system.

$$(\forall s, p, a, n)\,(Tkns(p, do(a, s)) = n \equiv \gamma_f(p, n, a, s) \vee$$

$$(Tkns(p, s) = n \wedge \neg \exists n'\, \gamma_f(p, n', a, s))),$$

where $\gamma_f(p, n, a, s) \stackrel{def}{=} \gamma_{f_e}(p, n, a, s) \vee \gamma_{f_l}(p, n, a, s)$, referring to the two sets of firing actions that cause the number of tokens at place p on situation $do(a, s)$ to be equal to n:

- $\gamma_{f_e}(p, n, a, s) \stackrel{def}{=} (\exists t)\,(pre(t, p) \wedge \neg post(t, p) \wedge n = Tkns(p, s) + 1 \wedge a = fire(t))$ (the number of tokens at place p at situation s is $(n-1)$, and a is an action of firing transition t, which enters p);
- $\gamma_{f_l}(p, n, a, s) \stackrel{def}{=} (\exists t)\,(pre(p, t) \wedge \neg post(p, t) \wedge n = Tkns(p, s) - 1 \wedge a = fire(t))$ (the number of tokens at place p at situation s is $(n+1)$, and a is an action of firing transition t, which leaves p);

The above Successor State Axiom summarizes all conditions where the number of tokens at place p is n at situation $do(a, s)$: n could be achieved by action a from situation s, or at situation s the number of tokens at p is already n and the action a that occurs in s will not change it to some other values. Note in particular that, aside from the accommodation of time, this axiom is the same as the one in $\mathcal{D}_{scope}$, since introducing of inhibitor arc only changes the preconditions where a transition node is enabled to fire.

As an example, we define Petri net in Figure 3 as a SCOPE theory. The axiomatic part of SCOPE, including Foundational Axioms, Precondition Axioms, and Successor State Axioms remain unchanged. We only need to specify the initial marking and the situation-independent, topological structure of the Petri-net in Figure 3. That is, the number of tokens at place P_1 is 0. For any other place node, the number is zero: $tkns(P_1) = 1$, and $tkns(P_i) = 0$ for $2 \leq i \leq 12$. Place node P_1 enters transition node T_1, and T_1 enters P_2, etc.

$$pre(P_1, T_1),\ pre(T_1, P_2),\ \ldots,\ pre(T_{11}, P_{12}).$$

3. **An ontological account of BPMN flow-control components.** This section includes two parts. We first specify formally the restrictions for well-formed BPMN processes, this would facilitate implementation of automated tools for evaluating formally whether a BPMN process is well-formed (using a FOL theorem prover for example). We then demonstrate how Petri nets modules for their corresponding BPMN objects can be formally described through carrying out further axiomatization on top of SCOPE.

3.1. **Well-formed BPMN process.** From Section 3 of [4], a well-formed BPMN process has to satisfy the following five criteria:

1. a start event or an exception event has just one outgoing (sequence) flow but no incoming flow;
2. an end event has just one incoming flow but no outgoing flow;
3. activities and intermediate events have exactly one incoming flow and one outgoing flow;
4. fork or decision gateways have one incoming flow and more than one outgoing flows; and

5. join or merge gateways have one outgoing flow and more than one incoming flows.

In this section, first-order axiomatization of these criteria are presented (assuming that objects in the domain are sorted in logical sense).

- $startM(x)$ Message x starts.
- $endM(x)$ Message x ends.
- $(\forall\, x, y)(pre(x, y) \equiv post(y, x))$.
- $(\forall\, x)startE(x) \equiv start(x) \vee startM(x)$.
- $(\forall\, x)(interE(x) \equiv message(x) \vee timer(x) \vee error(x))$.
- $(\forall\, x)endE(x) \equiv (end(x) \vee endM(x))$.
- A *start/exception* event has one outgoing flow and does not have incoming flow

$$(\forall\, x, y)\Big(\neg\big(pre(x, start(y)) \wedge \neg\big(pre(x, exception(y))\big)\Big),$$

$$(\forall\, y)\Big((start(y) \vee exception(y)) \supset \big((\exists\, x)pre(y, x) \wedge$$

$$\neg(\exists\, x_1, x_2)(pre(y, x_1) \wedge pre(y, x_2) \wedge x_1 \neq x_2)\big)\Big).$$

- An *end* event has one incoming flow and does not have outgoing flow

$$(\forall\, x, y)\ \neg(pre(end(y), x)).$$

$$(\forall\, y)\Big(end(y) \supset \big((\exists\, x)pre(x, y) \wedge$$

$$\neg(\exists\, x_1, x_2)(pre(x_1, y) \wedge pre(x_2, y) \wedge x_1 \neq x_2)\big)\Big).$$

- An *activity/intermediate-event* has one and only one incoming flow and outgoing flow

$$(\forall\, y)\Big((activity(y) \vee interE(y)) \supset$$

$$\big((\exists\, x)pre(y, x) \wedge \neg(\exists\, x_1, x_2)(pre(y, x_1) \wedge pre(y, x_2) \wedge x_1 \neq x_2)\big)$$

$$\big((\exists\, z)pre(z, y) \wedge \neg(\exists\, z_1, z_2)(pre(z_1, y) \wedge pre(z_2, y) \wedge z_1 \neq z_2)\big)\Big).$$

- An gateway *fork* or *decision* has one incoming flow and more than one outgoing flows

$$(\forall\, y)\Big((fork(y) \vee decision(y)) \supset \big((\exists\, x)pre(x, y) \wedge$$

$$\neg(\exists\, x_1, x_2)(pre(x_1, y) \wedge pre(x_2, y) \wedge x_1 \neq x_2)\big) \wedge$$

$$(\exists\, x_1, x_2)(pre(y, x_1) \wedge pre(y, x_2) \wedge x_1 \neq x_2)\big)\Big).$$

- An gateway *join* or *merge* has one outcoming flow and more than one ingoing flows

$$(\forall\, y)\Big((join(y) \vee merge(y)) \supset \big((\exists\, x)pre(y, x) \wedge$$

$$\neg(\exists\, x_1, x_2)(pre(y, x_1) \wedge pre(y, x_2) \wedge x_1 \neq x_2)\big) \wedge$$

$$(\exists\, x_1, x_2)(pre(x_1, y) \wedge pre(x_2, y) \wedge x_1 \neq x_2)\big)\Big).$$

3.2. **A theory of BPMN using SCOPE.** It is known from our previous work that, given a SCOPE theory, we can write SCOPE procedures through defining macros, where these macros are built on top of more primitive, existing procedures and the only basic action in the domain – transition node *fire*. With respect to the Petri net in Figure 3, we can define a set of simple SCOPE procedures. For example,

proc $checkCreditCardFirst$
$fire(T_1); fire(T_2); fire(T_4); fire(T_6)$
endProc
proc $prepareAndShipProducts$
$fire(T_3); fire(T_5); fire(T_8)$
endProc

The procedure $checkCreditCardFirst$ sequentially fires the transition nodes T_1, T_2, T_4 (the activity of checking credit card), T_6. The procedure is executable from initial setting, as there is a token in P_1, thus T_1 is initially enabled to fire. However, firing of T_4 has non-deterministic effects, the number of tokens in P_6 is increased by one, which non-deterministically enables either T_6 or T_7 (but not both) to fire, reflecting the non-deterministic fact that the credit card can either be accepted or rejected.

The procedure $prepareAndShipProducts$ sequentially fires the transition nodes T_3 (the activity of preparing products), T_5 and T_8 (the activity of shipping products). Note that, if procedure $prepareAndShipProducts$ is executable, then each of these three transition nodes including T_5 is executable. Nevertheless this means that the credit card must be accepted, otherwise T_7, not T_5, would be enabled to fire, so P_7 would never receive a token and T_5 can not be enabled to fire consequently.

With two procedures: $checkCreditCardFirst$, and $prepareAndShipProducts$, we define below a new procedure called $exampleExectutionOfAnOrder$. The procedure first checks the validity of the credit card, if it is accepted; then the process starts to prepare and to ship products.

proc $exampleExectutionOfAnOrder$
$checkCreditCardFirst$;
$(\pi n)(tkns(P_7, n) \wedge n = 0) \mid prepareAndShipProduct$
endProc

To this end, it is quite clear how methodologically we are able to axiomatize these BPMN flow-control components: we can simply define those Petri-net modules transformed from those BPMN objects, as illustrated in Figure 1, as SCOPE procedures.

We start from axiomatizing a Petri-net module for the start object N in BPMN. Suppose in the module, place P_1 enters transition T, which in turn enters P_2. Accordingly we define a SCOPE procedure

proc $startEvent(N)$
$[pre(P_1, T) \wedge pre(T, P_2)]?\ ;\ fire(T)$
endProc

That is, as long as the topological relationship is verified, T is suggested to fire in this procedure. In almost the same way, we can define procedures $endEvent$ and $task$ (definitions are skipped here).

The definition of a procedure corresponding to a fork gate in BPMN involves one place node that enters (and multiple place nodes that leave) the transition node to be fired. Suppose that P_1 enters T, and T in turn enters P_2, ..., P_n, we have

proc $forkGate(N)$
$[pre(P_1, T) \wedge pre(T, P_2) \wedge \ldots \wedge pre(T, P_n)]?$; $fire(T)$
endProc

The definition of a procedure corresponding to a join gate in BPMN involves one place node that leaves (and multiple place nodes that enter) the transition node to be fired. Suppose P_1 leaves T and P_2, ..., P_n enter T, we have

proc $joinGate(N)$
$[pre(P_2, T) \wedge \ldots \wedge pre(P_n, T_5) \wedge pre(T, P_1)]?$; $fire(T_5)$
endProc

The definition of a procedure corresponding to a decision gate in BPMN involves one place node that enters at least one transition node, where each transition node in turn enters a place node. Suppose that P_0 enters T_1, ..., T_n, in turn, T_1 enters P_1, T_2 enters P_2, ... T_n enters P_n, we have

proc $decisionGate(N)$
$[pre(P, T_1) \wedge \ldots \wedge pre(P, T_n) \wedge pre(T_1, P_1) \wedge \ldots \wedge pre(T_n, P_n)]?$
; $[fire(T_1) \mid \ldots \mid fire(T_n)]$
endProc

The definition of a procedure corresponding to a merge gate in BPMN involves one place node that multiple transition nodes enter, where for each transition node there is a place node that enters the node. Suppose that T_1, ..., T_n enters P, in addition, P_1 enters T_1, P_2 enters T_2, ... P_n enters T_n, we have

proc $mergeGate(N)$
$[pre(T_1, P) \wedge \ldots \wedge pre(T_n, P) \wedge pre(P_1, T_1) \wedge \ldots \wedge pre(P_n, T_n)]?$
; $[fire(T_1) \mid \ldots \mid fire(T_n)]$
endProc

3.3. **The order process example.** This section includes an example of an order process (adapted from Fig. 13(a) in [4]). When an order request arrives from a customer, both tasks "CheckCreditCard" and "Prepare Products" are initialized. If the credit card is OK, the packed products would be shipped (the task "ShiProducts" is executed) and the process finishes. Alternatively, if the credit card fails, the process finishes directly. BPMN representation of the process is depicted in Fig. 2.

The following sentences capture the topology of the Order Process Diagram.

$start(N_1)$, $pre(N_1, N_2)$, $fork(N_2)$, $pre(N_2, N_3)$, $pre(N_2, N_4)$,
$activity(N_3)$, $pre(N_2, N_3)$, $pre(N_3, N_5)$, $activity(N_4)$,
$pre(N_2, N_4)$, $pre(N_4, N_6)$, $decision(N_5)$, $pre(N_5, N_8)$,
$pre(N_5, N_6)$, $join(N_6)$, $pre(N_6, N_7)$, $activity(N_7)$,
$pre(N_7, N_8)$, $merge(N_8)$, $pre(N_8, N_9)$, $end(N_9)$.

The nine BPMN objects in Figure 2 are defined as the following SCOPE procedures.

proc $startEvent(N_1)$
$[pre(P_1, T_1) \wedge pre(T_1, P_2)]?$; $fire(T_1)$
endProc

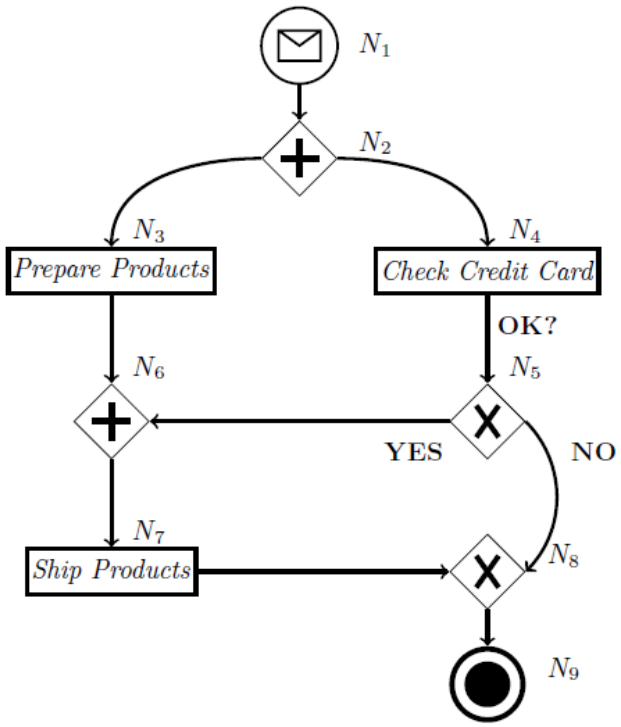

FIGURE 2. An Order Process in BPMN

proc $forkGate(N_2)$
$[pre(P_2, T_2) \wedge pre(T_2, P_3) \wedge pre(T_2, P_4)]?$; $fire(T_2)$
endProc

proc $task(N_3)$
$[pre(P_3, T_3) \wedge pre(T_3, P_5)]?$; $fire(T_3)$
endProc

proc $task(N_4)$
$[pre(P_4, T_4) \wedge pre(T_4, P_6)]?$; $fire(T_4)$
endProc

proc $decisionGate(N_5)$
$[pre(P_6, T_6) \wedge pre(P_6, T_7) \wedge pre(T_6, P_7) \wedge pre(T_7, P_9)]?$
;
$fire(T_6) \mid fire(T_7)$
endProc

proc $joinGate(N_6)$
$[pre(P_5, T_5) \wedge pre(P_7, T_5) \wedge pre(T_5, P_8)]?$; $fire(T_5)$
endProc

proc $task(N_7)$
$[pre(P_8, T_8) \wedge pre(T_8, P_{10})]?$; $fire(T_8)$
endProc

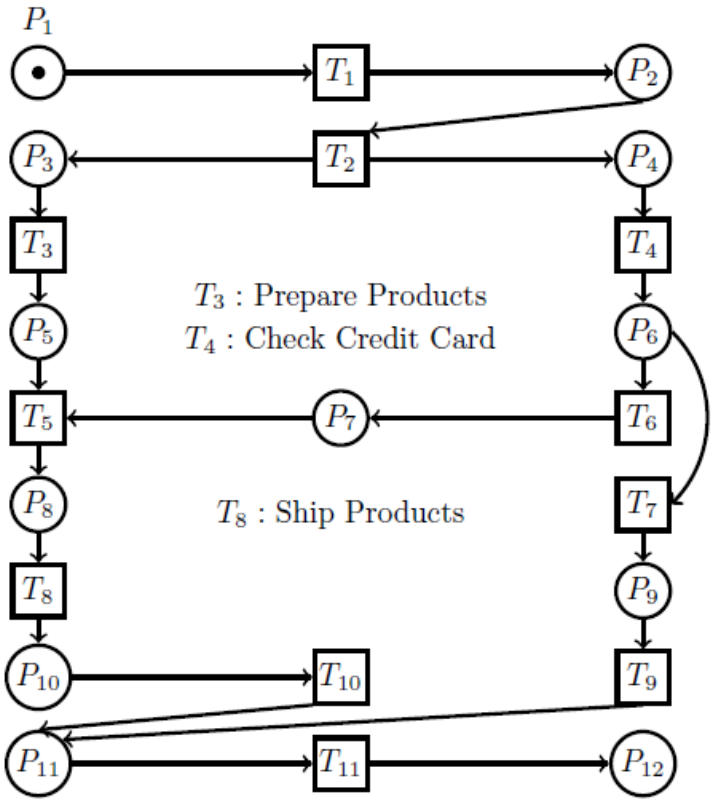

FIGURE 3. A Petri Net for the Order Process (Transformed from BPMN)

proc $mergeGate(N_8)$
$[pre(P_9, T_9) \wedge pre(P_{10}, T_{10}) \wedge pre(T_{10}, P_{11}) \wedge pre(T_9, P_{11})]?$
;
$fire(T_9) \mid fire(T_{10})$
endProc

proc $endEvent(N_9)$
$[pre(P_{11}, T_{11}) \wedge pre(T_{11}, P_{12})]?$; $fire(T_{11})$
endProc

It is handy to create further user-defined new procedures through macro-actions, i.e., through hierarchically constructing more complex procedures from existing ones. For example, the following procedure presumably specifies all possible executions of the BPMN process as specified in Figure 2.

proc $orderProcess$
$startEvent(N_1); forkGate(N_2);$
$[task(N_3); task(N_4)]|[task(N_4); task(N_3)];$
$decisionGate(N_5);$
$[joinGate(N_6); task(N_7); |;]$; $mergeGate(N_8)$;$endEvent(N_9)$
endProc

Feasibility/executability of procedure $orderProcess$ could be evaluated via querying the theory with

$$? - Do(orderProcess, S_0, S),$$

any instantiation of the situation variable S would correspond an execution from the initial situation S_0. We can also work directly on the Petri-net resulted form mapping (Figure 3). We can run queries to test the resulted Petri-net markings from a given execution instance of $orderProcess$. It can be verified that for some situation S_i, number of tokens in P_5 is one, indicating that the order process might complete improperly: since the credit card is rejected, the order process is discarded. The products are already prepared to be shipped, though shipping will never happen in this case.

4. **Conclusive remarks.** This paper proposes to axiomatize the Petri nets modules, resulted from transforming BPMN objects, as SCOPE-based procedures. In fact, the impressive feature of high adaptability and extensibility is shared by all Situation-Calculus-based Basic Action Theories including SCOPE. In a similar way, we recently used SCOPE to aggregate event patterns in Supply Chain Management [21], and to mitigate adverse interaction analysis for concurrent application of clinical practise guidelines for comorbid patients [11, 20]. Ongoing approaches using directly first-order logic for integrating medical treatments to manage multimorbidity include [11, 27, 12, 13, 22]. Formal process knowledge representation and reasoning might also find its applicability in combining probabilistic graphical models to build up frameworks and systems in areas for example information retrieval [23, 24].

Our previous research ([18] and [19]) indicates that we can easily implement software tools using logic programming language Prolog [2]. Further, we might consider adopting tractable variant to Situation Calculus (for example, a sequence of research on the topic proposed in [6, 7, 5] is quite interesting and relevant) so an actual implementation became practically feasible. Given the rapid development of first-order theorem proving technologies in the recent years, theorem provers have recently been used to support reasoning in ontologies [8]. As part of our initiative, we plan to adapt theorem proving technologies for process ontologies like SCOPE theories.

Finally, we remark that our approach is able to incorporate processes specified with temporal constraints, since in our previous research reported in [19], a variant to SCOPE which axiomatize Time Petri-nets are presented.

Acknowledgments. We are thankful to the insightful reviews we have received to improve both the content and presentation of the paper. We gratefully acknowledge the support by NSERC (Natural Sciences and Engineering Research Council of Canada) CREATE (Collaborative Research and Training Experience Program) award in ADERSIM (Advanced Disaster, Emergency and Rapid-response Simulation)[3], ORF-RE (Ontario Research Fund - Research Excellence)[4] award in BRAIN Alliance[5], Dapasoft Inc. in Toronto, ON, Canada[6], and the York Research Chairs Program.

[3] http://www.yorku.ca/adersim/NSERC/
[4] https://www.ontario.ca/page/ontario-research-fund-research-excellence
[5] http://www.brainalliance.ca/
[6] http://dapasoft.com/

REFERENCES

[1] C. Alvarenga and R. Schoenthaler, A New Take on Supply Chain Event Management, *Supply Chain Management Review*, **7**, 2003, 29–35.

[2] I. Bratko, *PROLOG Programming for Artificial Intelligence, Fourth Edition*, Addison-Wesley, 2011.

[3] R. M. Dijkman, M. Dumas and C. Ouyang, Formal Semantics and Automated Analysis of BPMN Process Models, *Preprint*, (2007).

[4] R. M. Dijkman, M. Dumas and C. Ouyang, Semantics and Analysis of Business Process Models in BPMN, *Information and Software Technology*, **50** (2008), 1281–1294.

[5] Y. Gu, *Advanced Reasoning about Dynamical Systems*, PhD thesis, University of Toronto, 2010.

[6] Y. Gu and M. Soutchanski, Decidable Reasoning in a Modified Situation Calculus, in *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, 2007, 1891–1897, URL http://ijcai.org/Proceedings/07/Papers/305.pdf.

[7] Y. Gu and M. Soutchanski, Reasoning About Large Taxonomies of Actions, in *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*, 2008, 931–937, URL http://www.aaai.org/Library/AAAI/2008/aaai08-148.php.

[8] I. Horrocks and A. Voronkov, Reasoning Support for Expressive Ontology Languages Using a Theorem Prover, in *Foundations of Information and Knowledge Systems, 4th International Symposium, FoIKS 2006, Budapest, Hungary, February 14-17, 2006, Proceedings*, 2006, 201–218.

[9] R. Liu, A. Kumar and W. M. P. van der Aalst, A Formal Modeling Approach for Supply Chain Event Management, *Decision Support Systems*, **43** (2007), 761–778.

[10] J. McCarthy and P. Hayes, Some Philosophical Problems from the Standpoint of Artificial Intelligence, *Readings in Artificial Intelligence*, (1981), 431–450.

[11] M. Michalowski, S. Wilk, D. Lin, W. Michalowski, X. Tan and S. Mohapatra, Procedural Approach to Mitigating Concurrently Applied Clinical Practice Guidelines, in *Proceedings of the First Workshop on Expanding the Boundaries of Health Informatics Using Artificial Intelligence (HIAI13)*, 2013.

[12] M. Michalowski, S. Wilk, X. Tan and W. Michalowski, First-Order Logic Theory for Manipulating Clinical Practice Guidelines Applied to Comorbid Patients: A Case Study, in *AMIA 2014, American Medical Informatics Association Annual Symposium*, Washington DC, USA, November 15-19, 2014.

[13] M. Michalowski, S. Wilk, D. Rosu, M. Kezadri, W. Michalowski and M. Carrier, Expanding a First-Order Logic Mitigation Framework to Handle Multimorbid Patient Preferences, in *AMIA 2015, American Medical Informatics Association Annual Symposium*, San Francisco, CA, USA, November 14-18, 2015.

[14] T. Murata, Petri Nets: Properties, Analysis and Applications, *Proceedings of the IEEE*, **77** (1989), 541–580.

[15] OMG, *Documents Associated With Business Process Model And Notation (BPMN), Version 2.0*, 2011, URL http://www.omg.org/spec/BPMN/2.0/.

[16] R. Reiter, *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*, MIT Press, Cambridge, MA, USA, 2001.

[17] X. Tan, SCOPE: A Situation Calculus Ontology of Petri Nets, in *6th International Conference of Formal Ontology in Information Systems, Toronto, Canada*, 2010, 227–240.

[18] X. Tan, *The Application of Ontologies to Reasoning with Process Modeling Formalisms*, PhD thesis, University of Toronto, 2012.

[19] X. Tan, Go beyond the SCOPE: A Temporal Situation Calculus-based Software Tool for Time Petri Nets, in *25th International Conference on Industrial Engineering and Other Applications of Applied Intelligent Systems, Dalian, China*, **7345** (2012), 134–143.

[20] X. Tan, Towards a Formal Representation of Clinical Practice Guidelines for the Treatment of Comorbid Patients, in *Seventh IEEE International Conference on Bioinformatics and Biomedicine, Shanghai, China, December 18-21, 2013*, 2013, 578–583.

[21] X. Tan and G. K. Tayi, An Ontological and Hierarchical Approach for Supply Chain Event Aggregation, in *Ninth IEEE International Conference on Semantic Computing, Anaheim, California, USA*, 2015, 69–72.

[22] X. Tan, X. An, N. Pairaudeau and J. Huang, Towards a Formal Account of the Dynamics of Knowledge and Context in Surgical Rooms for the Practice of Surgical Safety CheckLists, in *AMIA 2016, American Medical Informatics Association Annual Symposium*, Chicago, IL, USA, November 12-16, 2016.

[23] X. Tan, J. Huang and A. An, Ranking Documents Through Stochastic Sampling on Bayesian Network-based Models: A Pilot Study, in *SIGIR '16: Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 961–964, 2016.

[24] X. Tan, F. Jiang and J. Huang, On the Effectiveness of Bayesian Network-based Models for Document Ranking, in *ICTIR '17: Proceedings of the ACM SIGIR International Conference on Theory of Information Retrieval*, 309–312, 2016.

[25] W. M. P. van der Aalst, The Application of Petri Nets to Workflow Management, *Journal of Circuits, Systems and Computers*, **8** (1998), 21–66.

[26] W. M. P. van der Aalst, A. H. ter Hofstede, B. Kiepuszewski and A. P. Barros, Workflow Patterns, *Distributed and Parallel Databases*, **14** (2003), 5–51.

[27] S. Wilk, M. Michalowski, X. Tan, W. Michalowski, Using First-Order Logic to Represent Clinical Practice Guidelines and to Mitigate Adverse Interactions, in *6th International Workshop Knowledge Representation for Health Care at the the Vienna Summer of Logic*, 45-61, Vienna, Austria, 2014.

E-mail address: xtan@yorku.ca
E-mail address: Yilan.Gu@td.com
E-mail address: jhuang@yorku.ca