

Assigning Ontology-Based Semantics to Process Models: The Case of Petri Nets

Pnina Soffer¹, Maya Kaner², and Yair Wand³

¹ University of Haifa, Carmel Mountain 31905, Haifa, Israel

² Ort Braude College, Karmiel 21982, Israel

³ Sauder School of Business, The University of British Columbia, Vancouver, Canada
spnina@is.haifa.ac.il, kmay@braude.ac.il, yair.wand@ubc.ca

Abstract. Syntactically correct process models are not necessarily meaningful or represent processes that are feasible to execute. Specifically, when executed, the modeled processes might not be guaranteed to reach their goals. We propose that assigning ontological semantics to process modeling constructs can result in more meaningful models. Furthermore, the ontological semantics can impose constraints on the allowed process models which in turn can provide rules for developing process models. In particular, such models can be designed to be valid in the senses that the process can accomplish its goal when executed. We demonstrate this approach for Petri Net based process models.

1 Introduction

Process modeling is a complicated task and, hence, error-prone (e.g., [7][10]). Much effort has been devoted to the verification of process models leading to methods and tools for analyzing structural properties of process models and for detecting logical problems in them. These approaches are applied to already developed models, but do not provide guidance on how to develop valid models.

The syntax of process modeling languages specifies how to compose their constructs (which often have graphical notation) into process models. The semantics is believed to represent some real-world phenomena. These languages are usually defined textually or mathematically. Textual definitions are typically semi-formal or informal (e.g., “An event is something that “happens” during the course of a business process.” [8]). Mathematical definitions can support precise analysis of models.

Syntactically correct process models are not necessarily meaningful or feasible to execute. This entails the need for checking completed process models for structural and behavioral properties related to whether they can be successfully executed or not.

Some research evaluated process modeling languages by mapping their constructs to ontological concepts (which are assumed to convey real-world semantics) [9]. These attempts revealed various deficiencies such as ontological incompleteness, construct overload, redundancy, and excess. In particular, no ontological meaning was identified for control flow constructs which exist in practically every process modeling language (typically manifested as splitting and merging elements).

Recently, the Generic Process Model (GPM) was used to suggest an interpretation of control flow structures [12]. GPM provides a process specification semantics based

on ontological constructs. It is intended as a framework for reasoning about process models in terms of their real-world meaning. To apply the GPM for this purpose, its constructs should be mapped to the modeling languages used, which often employ graphical notation easy for human use. Ontology-based semantics imposes modeling rules in addition to the language-based syntactical restrictions. We suggest that these rules can guide the construction of meaningful and feasible process models.

In this paper we demonstrate the use of ontological semantics for Petri net based process models, or, more precisely, Workflow nets. Petri nets are widely used, provide a high degree of formality which supports model verification, and have a graphical notation with a precisely defined mathematical semantics. An extensive body of work exists on the mathematical, structural, and behavioral properties of Petri nets and Workflow nets (e.g., [2]). Furthermore, they serve for formalizing and analyzing models in other modeling languages (e.g., EPC [1]). Petri nets employ a small set of constructs, yet possess an impressive expressive power and can be used to represent precisely the entire set of workflow patterns [3].

Petri net analysis addresses the structure of the net rather than the semantics of its elements. GPM can provide such semantics in terms of state specification and state transitions of the process domain. In this paper we show that assigning this semantics to places and transitions can lead to better-designed process models and help avoid undesired situations (which in turn can be formalized using Petri net properties).

In the following, Section 2 introduces GPM and its control flow interpretation; Section 3 maps Petri net constructs to GPM. Section 4 explores restrictions on Petri nets, and introduces additional restrictions based on GPM, and their implications for Petri net properties. Section 5 is a conclusion.

2 The Generic Process Model (GPM)

The focus of GPM analysis is a *domain*, which is a part of the world consisting of interacting *things*. We describe the behavior of the domain using concepts from Bunge's ontology [4][5] and its adaptation to information systems [13][14] and to process modeling [11][12]. A domain is represented by a set of *state variables*, each depicting a property of the domain and its value at a moment in time. A successful process is a sequence of *unstable states* of the domain, leading to a *stable state*, which is in the set of goal states (simply – *goal*). An unstable state is a state that must change due to actions in the domain (an *internal event*). A stable state is a state that only changes due to action of the environment on the domain (an *external event*). Internal events are governed by *transformation (transition) laws* that define the allowed (and sometimes necessary) state transitions (manifested as events in the domain).

We formalize these concepts as follows:

Definition 1: A domain model is a set of state functions $D = \{f_1(t) \dots f_n(t)\}$. The value of $f_k(t)$ at a given time is termed a *state variable*, denoted x_k .

The set of state variables for domain D is denoted by $X^D = \{x_k; k \in I = \{1 \dots n\}\}$. The state of the domain at a given time is $s(D) = \langle x_1, \dots, x_n \rangle$ (or simply s). A set of states of domain D is denoted by $S(D)$.

Definition 2: A *transformation law* on D is a mapping $L: S(D) \rightarrow S(D)$

Definition 3: A domain will be said to be in a *stable state* if $L(s)=s$ and in an *unstable state* if $L(s)\neq s$.

Definition 4: A law will be said to be *well-defined* iff it is a function.

Often, several domain states can be considered equivalent. Hence, the transformation law can be represented as a mapping between sets of states. Such a set can be specified by a predicate $C(s)$. Specifically, the process goal is a set of stable states, specified by a predicate that manifests business objectives to be fulfilled by the process. The task of the process designer is to implement a transformation law so that the process can accomplish its goal.

To model practical situations, we consider a domain as comprising *sub-domains*, each represented by a subset of the domain state variables. Changes that occur in a sub-domain when the domain changes state, are termed the *projections* of the domain law (or domain behavior) on the sub-domain. Formally:

Definition 5: A *sub-domain* is part of the domain described by a subset of X^D .

A sub-domain D^1 of D is described in terms of $X^{D^1}\subset X^D$; $X^{D^1}=\{x_k; k\in I^1\subset I\}$.

The state of D^1 is $s(D^1)=\langle x_{k_1}, \dots, x_{k_{|I^1|}} \rangle$, $k_j \in I^1$ and $k_j \neq k_l$ for $j \neq l$.

Definition 6: Let the state of D be $s=\langle x_1, \dots, x_n \rangle$. The *projection* of s on the sub-domain D^1 is $s/D^1=\langle y_1, \dots, y_{|I^1|} \rangle$ where $y_k=x_{\Pi(k)}$.

It is possible that several domain states will map on the same state of the sub-domain. This, in turn, can result in the same sub-domain state changing in different ways, depending on the state of the whole domain.

Definition 7: Let v be a state of D^1 . The *projecting set* for v in D is the set of states of D that project v in D^1 : $S(v;D^1)=\{s \in S(D) \mid s/D^1=v\}$

We now define the effect of the domain law (L) on the sub-domain D^1

Definition 8: Let $v \in S(D^1)$ and let $s(v;D^1)$ be the projecting set of v . The *law projection* of L_D on D^1 (denoted L/D^1) for v is defined by the mapping $L_{D^1}: S(D^1) \rightarrow S(D^1)$ such that $L_{D^1}(v)=\cup\{L(s)/D^1 \mid s \in s(v;D^1)\}$.

In words – the projection of the law is defined as the union of projections of the states mapped into by the law. We are interested in cases where the projected behavior of the whole domain on a sub-domain creates a well-defined function in the sub-domain. In other words, a given unstable state of the sub-domain will always map in the same way, independent of the state of the whole domain, and hence independent of the states of other sub-domains. We will then say that the sub-domain behaves *independently*. Partitioning of the domain into independently-behaving sub-domains is often a consequence of different actors acting in the domain. These actors can be people, departments, machines, computers and combinations of those.

Definition 9: A sub-domain D^1 of D will be called an *independently behaving* (in short an *independent*) sub-domain iff the law projection on D^1 is a function.

Corollary: For an independent sub-domain the law projection depends only on state variables of the sub-domain.

Note: a sub-domain might behave independently for only a subset of the state space of D . Definition 9 can be restricted to a subset of domain states.

As an independent sub-domain changes its state to a stable one, it is possible some other independent sub-domains will become unstable and will begin transforming. Thus, a sequence of transformations occurs. This sequence comprises a process.

Process models usually include split and merge points, which reflect either concurrency or choice between possible alternative paths. We now interpret these in GPM terms. First, it is possible a set of states arrived at may be partitioned so the next transformation is defined differently for each subset of states. Such partitioning might occur because the law becomes “sensitive” to a certain state variable. Consider, for example, a process where a standard product is manufactured, and then packaged according to each customer's requirements. Manufacturing does not depend on the customer (even when the customer is known). When manufacturing is completed, customer information will determine a choice between packaging actions. This situation is an exclusive choice (an XOR split). The different actions may lead to states which are equivalent for determining the next action (the law will not distinguish between different packaging options), for example, transferring the products to finished goods inventory. This is the point where the paths merge.

Definition 10: S_{sp} is an *exclusive choice* splitting point iff there exist sets of states S_1, S_2, \dots, S_n such that $S_i \subset S_{sp}$, $S_j \cap S_k = \emptyset$, and $L(S_j) \neq L(S_k)$, $j \neq k$, $j, k = 1 \dots n$.

The corresponding form of a merge (sometimes termed simple merge) is when a single set of states is reachable by law from different sets of states.

Definition 11: Let S_1 , S_2 , and S_{me} be sets of states such that $S_1 \neq S_2$, $S_1, S_2 \subset S_{me}$. S_{me} is a *simple merge* iff $L(S_1) = L(S_2) = S_{me}$.

Also related to splitting and merging is *concurrency*. Since one domain cannot have concurrent transformations, concurrency should relate to transformations in different sub-domains. It means that if each sub-domain proceeds through a sequence of (projected) states, all combinations of the projected states of the different sub-domains are possible (in principle).

Lemma 1: Two sub-domains can transform concurrently only if they are independent.

Proof: Assume that two sub-domains are not independent. Then the transitions in one can depend on the state of the other. In this case, only some combinations of states of each sub-domain are possible.

It follows that a split leading to concurrency must be related to a decomposition of the domain into independently behaving sub-domains. In such a split, for the process to continue, at least one sub-domain must be unstable with respect to its (projected) law. If all these sub-domains are in unstable states for all states in the split, then this is a parallel split. Otherwise, several possibilities exist, depending on the number of the unstable sub-domains (see [12][11]). In particular, if exactly one sub-domain can be in an unstable state, then, based on Definition 10, this is an exclusive choice.

Definition 12: S_{sp} is a *parallel split* iff there exist at least two sub-domains such that at S_{sp} each sub-domain becomes independent and is in an unstable state.

For example, in the process discussed above, once products are ready, the process domain can be decomposed into two independent sub-domains: one where shipment is arranged and one where the products are transferred into the warehouse. These two sub-domains are independent and in an unstable state, thus they operate concurrently. A decomposable domain may entail different types of merge points (see [12][11]). In particular, a simple merge - where the completion of action of any sub-domain causes the process to proceed. Here we define a synchronizing merge, where process continuation requires that all active sub-domains complete their tasks. Consider a set of states in a merge point. These states should be unstable to enable the process to continue. They should be reachable from the split, hence their projection in each sub-domain should be reachable from the split for the sub-domain. In a synchronizing merge, each sub-domain becomes stable (“waiting” for the other sub-domains). Once all the sub-domains reach the merge, the process can continue. Formally:

Definition 13: Let $D^k \subset D$, $k=1\dots n$ be independent sub-domains operating concurrently following a split point S_{sp} . Let S_{me} be a set of unstable states in D , reachable from S_{sp} . S_{me} is a *synchronizing merge* iff $\forall s \in S_{me}, \forall k, s/D^k$ is stable.

Finally, the explicit representation of process goal in GPM supports the analysis of process models for goal reachability. A process whose design ensures its goal will always be achieved under a certain set of triggering events (which are external to the domain) is termed valid [11] with respect to this set of events.

3 GPM – Petri-Net Mapping

3.1 Petri-Nets and Workflow Nets

This section provides some definitions of Petri-nets in general and Workflow-nets in particular, and their properties which are relevant for our discussion.

A Petri-net is a directed bipartite graph with two node types called places (circles) and transitions (rectangles), connected by arcs. Connections between two nodes of the same type are not allowed.

Definition 14: A Petri-net is a triple (P, T, F) :

- P is a finite set of places;
- T is a finite set of transitions ($P \cap T = \emptyset$)
- $F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs.

At any time a place contains zero or more tokens (black dots), and the state of the net is the distribution of tokens over places. The notations $\bullet t$, $t \bullet$, $\bullet p$, $p \bullet$ indicate the sets of input and output places of transition t and the sets of transitions of which p is an input and output place, respectively. Given two states M_1 and M_n , $M_1 \rightarrow M_n$ denotes that M_n is reachable from M_1 through a firing sequence σ .

Some Petri-nets properties, relevant for our discussion, are defined below.

Definition 15: A Petri net is *bounded* iff for each place p there is a natural number n such that for every reachable state the number of tokens in p is less than n .

Definition 16: A Petri net is a *free choice* Petri net iff, for every two transitions t_1 and t_2 , $\bullet t_1 \cap \bullet t_2 \neq \emptyset$ implies $\bullet t_1 = \bullet t_2$.

A Petri net which is not free-choice usually involves a mixture of choice and parallelism, which is hard to analyze and considered inappropriate. Most of the mathematically-based properties identified and analyzed with respect to Petri nets relate to free-choice Petri nets only.

A specific form of Petri net, often used with respect to workflow modeling is Workflow net (WF net).

Definition 17: A Petri net (P, T, F) is a *Workflow net* (WF-net) iff:

- (i) There is one source place $i \in P$ such that $\bullet i = \emptyset$.
- (ii) There is one sink place $o \in P$ such that $o \bullet = \emptyset$.
- (iii) Every node $x \in P \cup T$ is on a path from i to o .

A WF-net represents the life-cycle of a single workflow case in isolation. It has an initial state following the generation of a case, where only one token exists in place i , and a final state o after which the case is deleted. The property which is ultimately sought and verified for WF nets is soundness.

Definition 18: A procedure modeled by a WF-net $PN = (P, T, F)$ is *sound* iff:






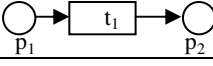
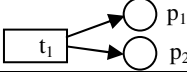
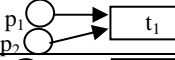
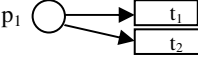
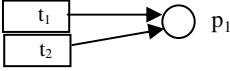
- (i) For every state M reachable from state i , there exists a firing sequence leading from state M to state o .
- (ii) o is the only state reachable from state i with at least one token in place o .
- (iii) There are no dead transitions in (PN, i) .

Soundness means that the modeled procedure will terminate eventually, and properly (i.e. no further transition will occur). Soundness can be verified in polynomial time for free-choice WF-nets. Another property, closely related to soundness, is well-structuredness. In a well-structured WF-net a splitting point and a merging point which correspond to each other are of the same type, namely, a split at a place (transition) corresponds to a merge at a place (transition).

3.2 Mapping Workflow-Nets to GPM

Table 1 presents the GPM interpretation of WF-net basic constructs and combinations which form control-flow basic building blocks. We focus our discussion on WF-nets, since these have a distinct termination place, which may correspond to GPM's goal concept. Nevertheless, most of the discussion is applicable to Petri nets in general. As shown in Table 1, every basic WF net construct and building block can be assigned a GPM-based interpretation. We do not attempt to do a reverse mapping, namely interpret GPM terms using WF nets, since GPM addresses issues beyond the control flow of the process, which are not in the scope of WF nets. Nevertheless, considering the proposed control flow mapping, this interpretation assumes certain domain semantics assigned to the places and transitions of a WF net. This semantics poses requirements which do not exist in the WF net syntax. If these requirements are not met, a WF net cannot be transformed into a meaningful GPM specification. It can thus be claimed that the expressive power of WF nets exceeds the expressive power of GPM with respect to control flow. Alternatively, it can be argued that WF net

Table 1. GPM interpretation of WF-Net constructs and basic building blocks

WF-net construct / building block	GPM interpretation
Place p 	A set of states of a sub-domain (the projection of a set of states over a sub-domain)
Transition t 	A transformation in a sub-domain
Arc 	An unstable state leading to a transformation. A transformation leads to a state.
Initial place i 	The initial set of states I
Final place o 	The goal set G
Sequence 	t_1 is a transformation in a sub-domain, from a set of states p_1 into a set of states p_2
A parallel split 	t_1 is a transformation after which the domain becomes decomposable, where p_1 and p_2 are state projections over different sub-domains
A synchronizing merge 	t_1 is a transformation whose initial set of states is $p_1 \cap p_2$ ¹
An exclusive choice 	p_1 is a set of states in which one of two sub-domain transformations is possible
A simple merge 	p_1 is a set of states reachable by two transformations, t_1 and t_2 , separately

syntax, being anchored in mathematical semantics of graphical symbols, allows structures which are not necessarily possible in reality.

4 Mapping-Based Modeling

4.1 Modeling Requirements

Assigning a GPM meaning to WF-nets imposes additional requirements on the use of WF-nets for process modeling. In particular, we will use the goal concept of GPM to form these requirements with respect to WF-nets.

Definition 19: A WF-net is a *well-mapped domain representation* iff it can be mapped to a GPM specification.

To make this concept operational, we derive necessary conditions for a WF-net to be a well-mapped domain representation in the sense of this definition. First, every place in the net should represent a set of states of some defined sub-domain ($x_1..x_k$).

Necessary condition 1: $\forall p_k \in P, \exists D_k \subseteq D$ such that p_k is active iff D_k is in a given set of states $S \subseteq S(D_k)$.

We denote the sub-domain as D_{p_k} , its state variables as X^{p_k} and the set of states by S_{p_k} . S_{p_k} can be specified by a predicate $C(X^{p_k})$. p_k is active iff $C(X^{p_k})$ is TRUE.

Regarding condition 1, consider the meaning of several tokens in a place. The predicate C can be some composite expression, which may assume the value TRUE in

¹ GPM merge relates to the states, while Petri-net relates to the transition that follows the states.

more than one situation (each applicable to a subset of states of the sub-domain). In particular, if the predicate is of the form $\langle \text{Expression OR Expression} \rangle$, then it may assume the value TRUE when each of the expressions becomes TRUE. In this case, each atomic expression may stand for a token. As an example, consider an order delivery process, where the customer orders several items which can be manufactured concurrently, and each item is delivered once it is ready. This can be modeled by a single place prior to delivery, whose predicate would be ((Item A=ready) OR (Item B=ready) OR...). The arrival of each item is modeled as adding a token to that place, triggering the delivery transition. The maximal number of tokens in a place is the number of atomic expressions related by an OR in the predicate that defines it. If the predicate cannot be decomposed to the form $\langle \text{Expression OR Expression} \rangle$, then the maximal number of tokens allowed in that place is 1.

Lemma 2: A WF-net which is a well-mapped domain representation is bounded.

Proof: Directly from the token interpretation. It is not possible to formulate a predicate composed of an infinite number of expressions.

As a second requirement, every transition should transform the state of a defined sub-domain (based on its input places). To continue, it must place a new sub-domain in an unstable state. Hence, the domains that correspond to the set of output places of a transition should have some common state variables with the domain corresponding to each of its input places. We denote the sub-domain transformed by transition t as D_t , and its set of state variables is X^t .

Necessary condition 2: $\forall t \in T, p_j \in \bullet t, p_k \in t \bullet$, it must be that $X^t \subseteq \cup_j X^{p_j}$; $X^t \cap X^{p_k} \neq \emptyset$.

In other words, for every transition, the state variables of the sub-domain in which the transition occurs, should: (a) be a subset of the state variables of all sub-domains represented by the input places (leading to it), and (b) include some state variable of every sub-domain represented by places preceding it. For example: assume the transition represents manufacturing a product. The transition should “use” only state variables from its inputs (e.g., raw materials and resources), and must affect the state which triggers the next activity (e.g. packaging or shipping the ready product).

Third, for a transition to act within a sub-domain, the sub-domain law must be independent (for the set of states represented by the transition’s input place).

Necessary condition 3: $\forall t \in T, \forall p \in P$, if $p = \bullet t$ then D_t is independent at p .

Fourth, concurrently operating threads operate over sub-domains that do not share state variables. Note that this is necessary, but not sufficient to guarantee that these sub-domains behave independently.

Necessary condition 4: Every two transitions t_1 and t_2 that may operate concurrently, satisfy $X^{t_1} \cap X^{t_2} = \emptyset$.

In particular, the following two cases can be specified:

Parallel split: Let t_s be a transition where a parallel split occurs, so $t_s \bullet = \{p_1, p_2\}$, and consider $t_1 = p_1 \bullet$ and $t_2 = p_2 \bullet$. Then $X^{t_1} \cap X^{t_2} = \emptyset$.

Synchronizing merge: Let t_m be a transition where a synchronizing merge occurs, so $\bullet t_m = \{p_1, p_2\}$, and consider $t_1 = \bullet p_1$ and $t_2 = \bullet p_2$. Then $X^{t_1} \cap X^{t_2} = \emptyset$.

Note that while necessary condition 4 addresses concurrency situations, we make no similar requirement with respect to choice-related splits. Since these may relate to both decomposable and non-decomposable domains, no strict rules can be formed here. Choice-related splits lead the domain in one of several possible paths, which, in turn, put the domain in one of possible (alternative) states. As opposed to the path definition in Petri nets, which relates to any sequence of connected elements, we relate to *domain paths* (D-paths), which correspond to selected sequences of states of the domain. The difference between these terms can be seen with respect to parallel splits, where the domain is decomposed into independently transforming sub-domains. In the graphical Petri net representation the sequence of elements in each sub-domain forms a different “path”. However, since no choice (decision) is made, we do not address these as separate D-paths.

GPM defines a path as a set of states the domain goes through via a sequence of transitions determined by the law and by external events. Petri nets (and specifically, WF-nets) do not explicitly address external events, assuming they will occur as expected. In a WF-net, considering a sub-domain D, its state is the distribution of tokens at a given moment in all the places p_j that satisfy $X^{p_j} \subseteq X^D$. We denote the state of sub-domain D by M^D .

Definition 20: Let D be a sub-domain in a WF-net. A *domain path* (D-path) of D is a sequence of states of D $\langle M^D_1, M^D_2, \dots, M^D_k \rangle$, such that a sequence of transitions $\langle t_1, t_2, \dots, t_{k-1} \rangle$ exists that satisfies $M^D_i \xrightarrow{t_i} M^D_{i+1} \dots \xrightarrow{t_{k-1}} M^D_k$, for $1 \leq i \leq k-1$.

For clarity, we hereafter relate to domain paths as D-paths and to “ordinary” (or “traditional”) Petri net paths simply as paths.

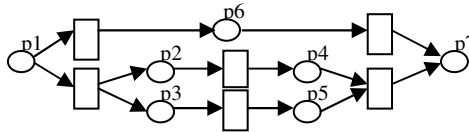


Fig. 1. D-paths example

D-paths are demonstrated with respect to the Petri net in Fig. 1, which includes four D-paths: (1) $p1 \rightarrow p6 \rightarrow p7$, (2) $p1 \rightarrow p2+p3 \rightarrow p4+p5 \rightarrow p7$, (3) $p1 \rightarrow p2+p3 \rightarrow p2+p5 \rightarrow p4+p5 \rightarrow p7$, and (4) $p1 \rightarrow p2+p3 \rightarrow p3+p4 \rightarrow p4+p5 \rightarrow p7$. These are not equivalent to the three “ordinary” paths of the net. When examining the four D-paths, it is clear that three of them (D-paths 2, 3, and 4) relate to different orderings in which the concurrent transitions can be performed, while D-path 1 specifies a different way for reaching p7. This can be formalized in the following definition:

Definition 21: Let A_c be the set of places that become active in D-path C. Two D-paths C_1 and C_2 are termed *distinct* iff $A_{c_1} - A_{c_2} \neq \emptyset$ and $A_{c_2} - A_{c_1} \neq \emptyset$.

One of the basic properties of Petri nets is free-choice, which is associated with “desirable” mathematical properties of nets. Non free-choice Petri nets are associated with situations of confusion between choice and parallelism, and are considered

improper (albeit syntactically possible). We do not require that Petri nets would be free choice to be well-mapped domain representations. Instead, we have a more relaxed requirement, which we term *relaxed free-choice*, specified as Necessary condition 5. In a free-choice Petri net, if two transitions share input places, then all their input places must be the same (namely, they have exactly the same triggering conditions). In contrast, the relaxed free-choice requirement is for the two transitions to share the same set of inputs (sub-domain state variables), but not necessarily at the same values (places). In other words, each of them may trigger on different combinations of values of the same set of state variables. We denote by $D_{\bullet t}$ the domain of all input places of a transition t : $X^{\bullet t} = \cup \{X^{pk} | p_k \in \bullet t\}$.

Necessary Condition 5 (Relaxed free choice): For every two transitions t_1, t_2 , if $\bullet t_1 \cap \bullet t_2 \neq \emptyset$ then $D_{\bullet t_1} = D_{\bullet t_2}$.

The relaxed free choice requirement is a result of Necessary conditions 2 and 3. Condition 2 requires D_t to be included in $D_{\bullet t}$, while condition 3 requires D_{t_1} and D_{t_2} to be independent. If D_{t_1} and D_{t_2} partly overlap (since $\bullet t_1 \cap \bullet t_2 \neq \emptyset$), then they cannot be independent of each other. In other words, at a given place a sub-domain can either transform independently or dependently of other sub-domains, both are not possible.

One of the basic concepts in GPM is the goal of a process, which we relate to the sink place o of a WF-net.

Necessary condition 6: The sink place in a WF net o marks a set of stable states of the entire process domain.

Two notes should be made. First, this set of stable states must be in the process goal. Second, a well mapped WF net is constructed so that once o is reached no other part of the domain can still be active. A particularly interesting case is when concurrent paths are merged by a simple merge. This structure is in violation of the well-structuredness property, yet we allow it. When concurrent paths are joined by a simple merge, the merge place may hold more than one token. The next transition may have other input places, so it may not be fired even when the merge place has tokens. For example, consider a process where two teams work concurrently to find a solution to a problem. When one team finds a solution, the process can continue. The solution found by the other team is not used. In this process model, the place representing that a solution exists may have two tokens, but only one token is used by the following transition. In the final state of the process, the entire domain is in a stable state. Nevertheless, for one sub-domain there might still be a solution “waiting” to be used, namely, to trigger additional action. For the entire domain to be stable with certainty, some action is required to “notify” the unstable sub-domain that no further transitions will take place. Technically, this can be accomplished by adding a transition from such nodes to the final place (in our example such transition may stand for archiving or discarding the “losing” solution).

Another result of Necessary condition 6 is that the net cannot include loops which action continues in parallel to the continuation of the process, namely, loops whose exit point is a parallel split, as formulated in Lemma 3.

Lemma 3: Let a WF-net be a well-mapped domain representation, and consider a transition t and three places p_1 , p_2 , and p_3 , such that $p_1 = \bullet t$, $\bullet p_2 = \bullet p_3 = t$. Then p_1 is not reachable from p_2 or from p_3 .

Proof: Assume p_1 is reachable from p_2 . When t fires, p_2 and p_3 become active, and while p_3 may lead to o , p_2 will lead infinitely to the sequence that activates t . This means that o may be reached while the loop sub-domain is unstable, in contradiction to Necessary condition 6.

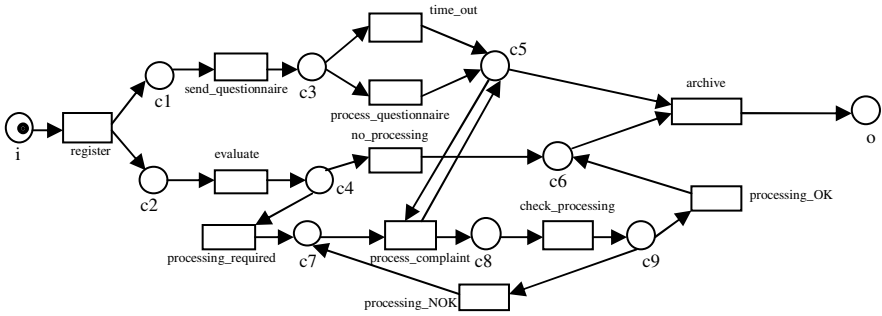


Fig. 2. A complaint processing process

To demonstrate how the necessary conditions can be used, let us examine the example given in Fig. 2 (taken from [1]). The WF-net representing complaint processing is not a well-mapped domain representation, for the following reason. The predicate that can be assigned to c_5 is $(Questionnaire_status=time_out \text{ Xor } Questionnaire_status=processed) \text{ Or } Complaint_status=processed$. It follows that c_5 is defined over a sub-domain which includes at least the *Questionnaire_status* and the *Complaint_status* state variables. c_5 and c_7 are input places of the transition *process_complaint* (a synchronizing merge), which transforms the status of a complaint. Such merge violates Necessary condition 4, which requires the sub-domains joined by a synchronizing merge to be disjoint. The modified model (Fig. 3) defines c_5 over a sub-domain which does not include the complaint status, thus it is a well-mapped domain representation.

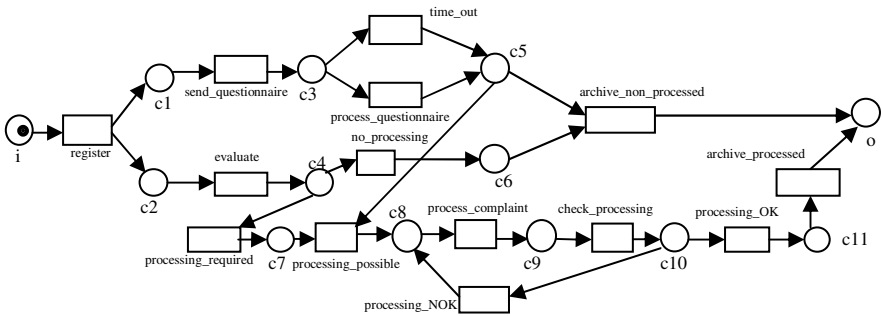


Fig. 3. A modified model of the complaint processing process

4.2 Process Validity Considerations

Validity of a process model can only be assessed with respect to a set of expected external events and to a defined goal [11]. How these are determined is outside the scope of the current analysis. We consider only the reachability of the process termination state, assuming that it represents the process goal. Assuming that all the expected external events occur, validity relates to completeness of the internal law definition (which relates to internal events) and to its consistency with the goal definition.

Incompleteness reflects potential deadlock situations. Following [6], a process instance is in deadlock iff it is not in the goal and no transition is enabled. GPM also allows for a process execution “hanging” when external events fail to occur, but we assume here no such failure happens. Thus, deadlock means that the process is in a state for which the law is not defined. In a well mapped domain representation this may occur when a transition has more than one required input place (i.e., it is a synchronizing merge which joins different sub-domains) and not all of them are enabled. Two situations are possible:

- (1) Not all sub-domains have been activated at the split point.
- (2) At least one of the sub-domains took a D-path which does not lead to the input place of the merge transition.

We will specify modeling rules to avoid each of these cases. Case (1) is possible if an exclusive choice is followed by a synchronizing merge. According to [12], such structure should not appear in a valid process. It also does not appear in a well-structured WF-net [1], where an exclusive choice is matched by a simple merge and a parallel split is matched by a synchronizing merge. As discussed above, we do not require well-structuredness. Instead, we only require that every synchronizing merge be preceded by a parallel split, leaving the simple merge unconstrained as to the type of split it should be preceded by. Since in Petri nets a synchronizing merge is in a transition, it should correspond to a transition in the split point (parallel split). This is formalized in Modeling rule 1.

Modeling rule (MR)1: Let x and y be two elements (i.e., places or transitions) in a well-mapped domain representation WF-net, connected by two different elementary paths leading from x to y . If x is a place then y should be a place too. If y is a transition then x should be a transition too.

To avoid the second case, the modeler needs to make sure that if two sub-domains that have alternative distinct D-paths need to synchronize, then every possible combination of these D-paths has a merging transition defined for it. To illustrate the idea, consider the examples of Fig. 4.

In Fig. 4(a), the process domain is split in $t1$ to two concurrently active sub-domains, and both these sub-domains have different D-paths that can be selected. The process may clearly deadlock, if one sub-domain takes a D-path leading to $p7$ while the other reaches $p10$, or if one sub-domain reaches $p8$ while the other takes a D-path that leads to $p9$. There are more combinations of D-paths that can be taken than combinations that lead to the goal of the process. The sub-domain on the left side has two distinct D-paths: (1) $p2 \rightarrow p7$, and (2) $p2 \rightarrow p8$. The sub-domain on the right

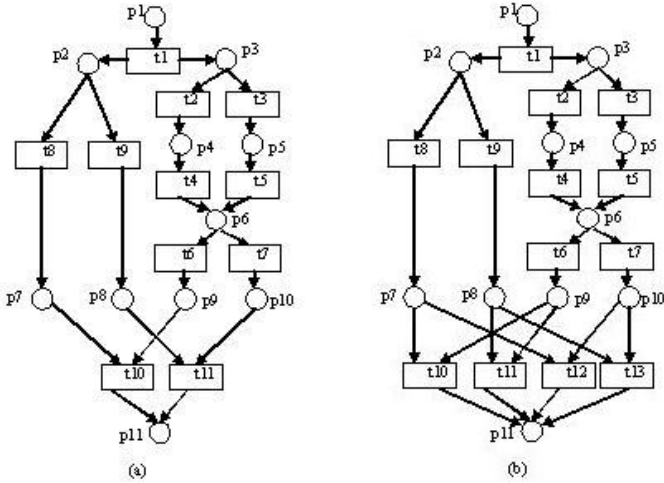


Fig. 4. D-path combinations

side has four distinct D-paths: (1) $p_3 \rightarrow p_4 \rightarrow p_6 \rightarrow p_9$, (2) $p_3 \rightarrow p_4 \rightarrow p_6 \rightarrow p_{10}$, (3) $p_3 \rightarrow p_5 \rightarrow p_6 \rightarrow p_9$, and (4) $p_3 \rightarrow p_5 \rightarrow p_6 \rightarrow p_{10}$. To eliminate the possible deadlock, we need to define action in every possible situation the process may reach. We may look for a place which is reached from all D-paths. Considering the right side sub-domain, p_6 is reachable in all the distinct D-paths. Hence, it is guaranteed to be reached. Let us examine a possible correction, where p_6 is connected to t_{11} . Then t_6 , t_7 , and t_{11} would share p_6 as an input place, while t_{11} is also preceded by p_8 (whose domain is different). This is in contradiction to Necessary condition 5 (relaxed free choice), which proscribes a sub-domain from being both independently transforming and merging at a given place. Following this analysis, p_9 and p_{10} , which are reachable in two distinct D-paths each and together “cover” all the four D-paths, do not represent states where the sub-domain is independent (since they lead to a merge). A complete solution, addressing every possible situation, requires the net to include a transition defined for every possible combination of non-independent places in the two sub-domains, namely, $(p_7, p_8) \times (p_9, p_{10})$, as shown in Fig. 4(b).

The above analysis is formalized in Modeling rule (MR) 2 that requires that if a transition depends on a combination of states of two sub-domains, and that combination is not guaranteed to happen, there must be other transitions specified for every other possible state combination.

Modeling rule (MR) 2: Let D_1, D_2 be two sub-domains, and t_1 a transition such that $\bullet t_1 = \{p_1, p_2\}$, where $p_1 \in D_1$ and $p_2 \in D_2$. Let PM_1 and PM_2 be sets of places such that the domain of each place in PM_1 or in PM_2 is not independent and for every distinct D-path in D_k there is a place in $PM_k, k=1,2$. Then for every pair of places $p_j \in PM_1$ and $p_k \in PM_2$, there must be a transition t such that $\bullet t = \{p_j, p_k\}$.

Note: D_1 and D_2 can be identified by backtracking paths from p_1 and p_2 until the first transition which is included in both paths. For example, it is easily seen that the model in Fig. 3 is in compliance to MR2.

Inconsistency between the law and the goal definition relates to infinite loops. In WF-nets, since every element must be on a path from i to o , loops must have (at least one) exit points. These may be parallel splits or exclusive choice splits. According to Lemma 3, parallel splits cannot be exit points of loops in a well-mapped domain representation WF-net. We shall hence examine the possible structures in which loops whose exit point is an exclusive choice may become infinite. When a loop has an exclusive choice as an exit point at place p , two cases are possible:

- (1) The next transition has only one input place ($\bullet t = \{p\}$). Then the exit depends on one sub-domain only. Structurally, this is not an infinite loop, and the exit from the loop depends on the decision criteria defined by the analyst.
- (2) The next transition has more than one input place (i.e., it merges a number of concurrent sub-domains). In this case, if the merge deadlocks, the loop will continue infinitely. However, merging deadlocks can be eliminated by using modeling rules 1 and 2. Hence, if the modeling rules are used in a well-mapped domain representation WF-net, it does not include infinite loops.

Theorem 1: A well-mapped domain representation WF-net which satisfies Modeling rules 1 and 2 is sound.

Proof: soundness has three requirements. (ii) proper termination – follows directly from Necessary condition 6, and (iii) no dead transitions – follows from Necessary condition 2. To prove (i), namely that o is reachable from every state reachable from i , we will show that for any state M reachable from i there is a transition t that can be fired. Since all the elements in a WF-net are on a path from i to o and no infinite loops are possible, if any arbitrary state M transforms, o will be reached.

We will show that in a given state M every transition is either (a) within an independent sub-domain, or (b) a result of a merge between two (or more) sub-domains. In the first case, a transition will be fired with certainty. In the second case, by MR1, all the required sub-domains should be active, and by MR2 there is a transition defined for every possible combination of D-paths of the sub-domains. Hence a transition will be fired.

Formally: Let $P(M)$ be the set of active places in M , and consider a place $p_1 \in P(M)$ and a transition $t_1 \in p_1 \bullet$. Two cases are possible: (1) $\bullet t_1 \subseteq P(M)$, then t_1 fires at M . (2) $\bullet t_1 \cap P(M) \neq \emptyset$, $\bullet t_1 \not\subseteq P(M)$. Then t_1 cannot fire at M , but we will show that there exists a transition t_2 that can fire at M . Having more than one input place, t_1 merges two or more independent sub-domains D_i . Assume $\bullet t_1 = \{p_1, p_2\}$, where p_1 is the projection of M over D_j , and $p_2 \notin P(M)$. We assume p_2 is the projection of some state M' over a sub-domain D_k ($k \neq j$). Since, by MR1, D_k is active at M , we denote the projection of M over D_k by p_3 ($p_3 \in P(M)$). According to MR2 there exist a transition t_2 and a place p_4 , such that $\bullet t_2 = \{p_1, p_4\}$, $D_{p_4} \subseteq D_k$, and p_4 is on a D-path that includes M/D_k . Three cases should be checked: (1) $p_3 \rightarrow p_4$. Then $M \rightarrow M'$, thus t_2 can fire. (2) $p_3 = p_4$. Then t_2 can fire. (3) $p_4 \rightarrow p_3$. This is impossible due to Necessary condition 5 (relaxed free choice).

In summary, following the necessary conditions and modeling rules, it is possible to construct a sound WF-net.

5 Conclusion

This paper proposed to use the ontologically-based GPM semantics for existing constructs of process modeling languages. We demonstrated how this can be done for WF-nets. We also showed how modeling guidelines, based on this semantics, can assist in avoiding process modeling problems that traditionally could only be detected by verification of the completed models. Existing verification algorithms for WF-nets can analyze in polynomial time only specific classes of models (free-choice or well-structured). The modeling rules suggested here can lead to sound WF-nets which are not necessarily free-choice or well-structured. Note that the modeling rules do not constitute a verification approach. Rather, they form a construction approach, which yields sound models when applied.

The essence of the analysis is in mapping common situations that can occur when a domain undergoes state transitions, into a WF-net representation. For a process to be guaranteed to reach its goal, its definition should fulfill three conditions: (1) no situations should arise where it “hangs”, (2) completeness: all possible states should have defined transitions, and (3) no infinite loops. Process “hanging” can happen when several conditions need to be fulfilled for the process to continue – i.e. in merge situations. Merges occur because a split has occurred earlier in the process. By choosing only appropriate combinations of splits and merges, the process can be guaranteed to proceed. This was the purpose of Modeling Rule 1. Completeness requires that the process model will specify continuation for all possible states – this was the purpose of Modeling Rule 2. Both rules and the goal definition ensure the absence of infinite loops. Constructing models that conform to these two rules, therefore assures that the process, when executed, can always complete (in the sense of reaching its goal). It is important to note that the rules guide the actual construction of process model, rather than being applicable only to complete models.

As GPM concepts are generic, they can be applied to other modeling languages. We intend to do this in future research. This application would require mapping of these languages to GPM and deriving appropriate restrictions and modeling rules. In addition, we plan to empirically investigate the effectiveness of the propositions made here in contributing to the quality of models produced by modelers. Finally, we will develop a modeling tool to support the application of the modeling rules when constructing a model.

References

- [1] van der Aalst, W.M.P.: Formalization and Verification of Event-Driven Process Chains. *Information and Software Technology* 41(10), 639–650 (1999)
- [2] van der Aalst, W.M.P.: Workflow Verification: Finding Control-Flow Errors Using Petri-Net-Based Techniques. In: van der Aalst, W.M.P., Desel, J., Oberweis, A. (eds.) *Business Process Management*. LNCS, vol. 1806, pp. 161–183. Springer, Heidelberg (2000)
- [3] van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: *Workflow Patterns*. *Distributed and Parallel Databases* 14(1), 5–51 (2003)
- [4] Bunge, M.: *Treatise on Basic Philosophy*. In: *Ontology I: The Furniture of the World*, vol. 3. Reidel, Boston (1977)

- [5] Bunge, M.: *Treatise on Basic Philosophy*. In: *Ontology II: A World of Systems*, vol. 4, Reidel, Boston (1979)
- [6] Kiepuszewski, B., ter Hofstede, A.H.M., van der Aalst, W.M.P.: *Fundamentals of control flow in workflows*. *Acta Informatica* 39(3), 143–209 (2003)
- [7] Mendling, J.: *Detection and Prediction of Errors in EPC Business Process Models*, PhD thesis, Vienna University of Economics and Business Administration (2007)
- [8] Object Management Group (OMG), *Business Process Modeling Notation Specification* (2006), <http://www.bpmn.org>
- [9] Rosemann, M., Recker, J., Indulska, M., Green, P.: *A Study of the Evolution of the Representational Capabilities of Process Modeling Grammars*. In: Dubois, E., Pohl, K. (eds.) *CAiSE 2006*. LNCS, vol. 4001, pp. 447–461. Springer, Heidelberg (2006)
- [10] Sadiq, W., Orłowska, M.E.: *On Correctness Issues in Conceptual Modeling of Workflows*. In: *Proceedings of the 5th European Conference on Information Systems*, Cork, Ireland, pp. 943–964 (1997)
- [11] Soffer, P., Wand, Y.: *Goal-Driven Multi-Process Analysis*. *Journal of the Association of Information Systems* 8(3), 175–203 (2007)
- [12] Soffer, P., Wand, Y., Kaner, M.: *Semantic Analysis of Flow Patterns in Business Process Modeling*. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) *BPM 2007*. LNCS, vol. 4714, pp. 400–407. Springer, Heidelberg (2007)
- [13] Wand, Y., Weber, R.: *On the Ontological Expressiveness of Information Systems Analysis and Design Grammars*. *Journal of Information Systems* (3), 217–237 (1993)
- [14] Wand, Y., Weber, R.: *Towards a Theory of Deep Structure of Information Systems*. *Journal of Information Systems* 5(3), 203–223 (1995)