
# SANET: Semantically Annotated Workflow Net

Luca Sabatucci, Antonella Cavaleri and Massimo Cossentino

*Technical Report N.:* *Date*

RT-ICAR-PA-16-06 October 2016

Consiglio Nazionale delle Ricerche, Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR )
-Sede di Cosenza, Via P.Bucci 41C, 87036 Rende, Italy, URL: *www.icar.cnr.it*
-Sede di Napoli, Via P. Castellino 111, 80131 Napoli, Italy, URL: *www.na.icar.cnr.it*
-Sede di Palermo, Via Ugo La Malfa 153, 90146 Palermo, Italy, URL: *www.pa.icar.cnr.it*


# SANET: Semantically Annotated Workflow Net

Luca Sabatucci, Antonella Cavaleri and Massimo Cossentino

*Technical Report N.:* *Date:*

RT-ICAR-PA-16-06 October 2016



[1]Istituto di Calcolo e Reti ad Alte Prestazioni, ICAR-CNR, Sede di Palermo, Via Ugo La Malfa 153, 90146 Palermo.

# SANET: Semantically Annotated Workflow Net

Luca Sabatucci, Antonella Cavaleri and Massimo Cossentino
ICAR-CNR,Palermo, Italy
Email: {name.surname}@icar.cnr.it

**Abstract**

This work proposes a general approach for checking the semantic consistency of a Workflow Model. Industries can have many benefits from being able to verify the fairness of their BPMN model. The adopeted strategy is driven by a Semantically Annotated workflow Net (SA-Net) generated from a BPMN well-formed model. We describe how to extend a WF-net with semantic annotations for inspecting behavioral properties of a workflow. The algorithm illustrated is based on Petri net analysis techniques and information on the state of the world associate to each main element in a BPMN.

## 1 Introduction

The traditional way to model and execute workflow is based on the petri net theory: tokens and activation. In order to enable the engine to self-adapt we want to relax rigid constraints due to this mechanism. We believe that a goal-based model of the workflow could improve the ability of the engine to search for alternatives on the run. In particular we adopt 1) an ontological view of the domain and 2) a goal definition based on triggering-condition/desired-final-state This paper aims at highlighting that the most information required for building an equivalent goal model are already contained in the BPMN description of the workflow. We focus on importance of logical information related to different state of the world produced at run-time to a workflow model. These semantic data are mapped on correspondent transition for a WF-net. Unsing analisys thecniques of petri nets and a the developed alghoritm it is possible identify bugs for the BPMN model to analize. The approach emphasizes that it is possible that a well-formed model may have hidden errors related to incorrect logical considerations.The remainder of this article is organized as follows: Section 3 presents the main concepts used for the scope of this paper. The Semantically Annotated Workflow Net (SA-net) is presented, it permits to describe the evolution of the state of the world related to a the execution of the corresponding workflow. Section 4 desscribes how a SA-Net ca be automatically produced from a BPMN. It is illustrated the impact of each BMPN element to determine the global state of a workflow. For this purpose it is necessary to evaluate internal and external factors related to BPMN element. Therefore, the adopted strategy is presented and some exampleas are illustrated in the Section 5. Section 2provides a summary of the current state of art, it focus on use of Petri nets for workflow static analysis and on inspect the workflow behavior. Finally, Section 4 gives the conclusions.

## 2 Related Work

BPMN t is mainly intended for modeling business processes at a conceptual level. However more and more it is required to move from an analysis process model towards executable error-free version.

In last decades, academic research has spent great effort in defining formal techniques for validation and verification of executable business processes. In this context, checking the correctness of workflow models has reached a good level of maturity [1–4].

A range of very different approaches are currently used for identifying structural inconsistencies and behavioral anomalies. A cornerstone of this research area was the idea to exploit the similarity of workflow with Petri-Nets, thus gaining a strong and consolidated formal theory.

### 2.1 Workflow Static Analysis through Petri Nets

Van der Aast, in [1], indicated the use of Petri Nets for the specification of complex workflows, and introduced the concept of WorkFlowNet (WF-net), a specific form of Petri net adapted for modeling a workflow definition.

He also shown how to operate a straightforward transformations from BPMN to Petri nets. The key for this transformation is based on a mapping of the BPMN routing constructs (AND-split, AND-join, OR-split, OR-join) into patterns of petri nets.

A WorkFlowNet represents a formal model of the behavior that may appear in the corresponding BPMN, but, more important, it may be used as proxy for verifying some properties of the BPMN. It was proved that the necessary condition for the structural correctness of a workflow is the soundness of the WorkFlowNet, that, in turn, is associated to the property of liveness and bound of the petri net. This is a great result, but conditioned by the limit that verifying the liveness of a petri-net is a NP problem. This opened the field to two mainstreams of sub-goals: i) to define an efficient transformation from a BPMN into a petri net and ii) to conduct static analysis in scalable and efficient way.

In its paper, Van der Aast simplified the scope by focusing on free-choice, well-structured and S-coverable petri-nets: A WorkFlowNet satisfying at least one of these properties can be analyzed in linear time.

Cao at al. in [2] proposes a Petri net based on Grid workflow modeling toolkit to help users describe their workflow and carry out analysis and verification. Grid workflow can be seen as a collection of task that are processed on distributed resource in a well-defined order to accomplish a specific goal. The toolkit is composed by two layers: a GUI BPEL Designer (through which users can describe Grid workflow and generate the definition of bpel file) and the workflow analyzer layer (BPELToPNML)that converts BPEL scripts into PNML (Petri Net Markup Language) format and it implements the verification algorithms and an optimization mechanism. The proposed toolkit aims defining the composition of processes, the analysis and verification of the workflow is related to its composition but it does not concern the definition of a single process.

Dijkman at al. in [5] proposed a mapping from BPMN to Petri nets, they have implemented this mapping as a tool for statically check semantic correctness of a models BPMN. The tool translates the XML serialization of BPMN models in the Petri Net Markup Language (PNML), the proposed mapping is designed for achieving static analysis. They introduces three restrictions of BPMN for simplify the presentation of the mapping. They use a graphical editor (ILog BPMN Modeller) to create BPMN models, and implement a simple pre-processor to transform the tool's output into XMI, finally the transformation tool loads this XMI, applies the transformation and export the resulting Petri net by a PNML file. This generated file can serve as input to Petri net-based verification tool (e.g. ProM [6]), by this tool they check two properties: Absence of dead task and Proper completion. In their work they show by some example how the semantic correctness can be detected by this mapping, but they test the soundness of BPMN models using the ProM framework, so do not think there is the possibility of logical error check.

BI at al. in [3] propose a theoretical framework for workflow verification applying propositional logic. Their workflow verification is based on process inference, which reduces the logical representation of a workflow model to its simplest form possible. They propose a new concept called 'constrained truth table' to derive new inferences rules that are applicable to workflow verification, this table excludes the truth values of impossible situations in a workflow model. In their work we can see how if a workflow model is reduced to the conclusion, then the workflow model is free from process anomalies. They compare three existing workflow verification approaches, Petri nets, graph reduction and matrix-based and compare them with their logic-based verification method. The limitation of their verification algorithm is that it is not able to handle certain overlapping workflow patterns.

Yu at al. in [7] present the Extended object Petri Net (EOPN) to model and verify BPMN process formally, author focuses on modeling and verifying by addressing the temporal constraint with Petri net. They note as each component process in BPMN model has its own time constraints, then they focuses on an non functional property(temporal). Their study is devoted to mapping a BPMN diagram into EOPN and to verify it in a formal way; in their work is presented an analysis approach (timestamp state class)and the details of rules mapping. Only a subset of BPMN metal objects is considered in this study, mapping into rules for execution handling, compensation, looping are not presented.

Reeijers at al in [8] adapt the concept of syntax highlighting to workflow nets. Their important contributions are : establish a theoretical argument and formalize a concept for syntax highlighting in workflow nets and present a prototypical implementation with the WoPeD modeling tool (a java-based open source tool supporting the modeling of Petri nets) [9]. Authors focus on the highlighting of syntactical elements in process models to improve their understability, they propose the use of color to highlight process model elements that relate to one another in a way that is comparable.They show the algorithm at the base of their work and they prove with experiments that the highlighting was of greatest benefit to the accuracy, as consequence the performance increase. This work show the potential of secondary notation to improve process model understanding.

Van der Aalst at all. in [10] show how verification of a typical process control specification can benefit from Petri -net based analysis techniques; a verification tool has been developed to illustrate the applicability of the approach. Their work focus on Task Structures [11] (a powerful language for the specification of process control). Authors translate Task Structures into Petri nets, they emphasize the major difference between Task Structures and Workflow nets (a special class of Petri nets): task structures do not have a unique final task then it is difficult to identify the point where a Task Structures terminates. The mapping Task Structures to classical Petri nets is straightforward and they show that for the verification of Task Structures we can benefit from Petri-net theory and tools. They start from a process specified in Staffware (a workflow management system), this process is automatically translated into a format readable by Woflan [12](an analysis tool wich can be used to verify the correctness of a workflow process definition). Finally they show that soundness property for Task Structures corresponds to liveness of the corresponding extended WF-net.

## 2.2 Analyzing the Workflow Behavior through Simulation

An alternative way that allows to perform the analysis of workflow models is through the use of simulation systems. Traditional simulation is focused on examining abstract steady-state situations.

Rozinat at all. in [13] present a new way of creating a simulation model, in whitch they integrate design, historic and state information. Their purpose is use simulation for answering strategic questions but also for tactical and even operational decision making. The structure of the simulation model is construct by the design information. The historic information is used to set parameters of the model (e.g., fit distributions). Finally the simulation model is initialize with the state information. Following this, process mining techniques to be used to view the real-world processes and the simulated processes in a unified way. The realization is based by integration of YAWL [14] , ProM [6] and CPN Tools [15]: YAWL is the workflow environment, ProM the process mining framework and CPN Tools is used for the simulation. They show how a workflow state can be exported from the YAWL engine using a generic workflow state XML schema format and how this could be translated into a CPN input file for simulation purposes.

Kovcs at all in [16] check correctness properties of workflows by a framework for the simulation and formal analysis of workflow models. Authors transform a workflow models implemented in the BPEL language in dataflow networks (a low level mathematical notation which can be verified automatically) and show the rules for to realize this mapping. Finally dataflow network constructs are mapped into PROMELA(a Process Meta Language) [17]. The transformations are property preserving in a sence that every execution path of the BPEL process can be found in the PROMELA program too. If a property is valid, concerning the PROMELA model, then it holds for the BPEL process as well.

## 2.3 Brief Overview of Tools for Automatic Verification of Correctness

In this section we analyze the tools which support workflow definition through the concepts of Petri nets. The main need is to identify Petri nets based workflow modelling software tools. WoPeD (Workflow Petrinet Designer) [9]is a tool open-source, it provide an environment for modelling, simulating and analyzing processes. In the WoPeD editor the user can to draw and manage Petri nets as workflow according to the van del Aalst notation. WoPeD contains a graphical resource model editor covering the organization view of BPM. WoPeD can export file format to standard PNML and support graphical formats (PNG,JPEG). User can check some properties of a Petri nets (soundness, free-choise, S-coverability ) through the analysis session. This tool contains algorithms to construct the coverability graph of a Petri net and it can be displayed for inspection and navigation of the associated Petri net. WoPeD supports quantitative simulations. The recent feature of WoPeD is an interface to access document stored in a AProMoRe repository (a open source software witch provide a set of conversion algorithms between most common process model formats). Yasper (Yet Another Smart Process Editor) [18] is a tool for modeling, analyzing and simulating workflow systems based on Petri nets. Relevant are animation and simulation, Yasper expose correctness and performance issues; by the simulation report we can see the completion of cases and standard performance indicators. The tool can be combined with other tools, for facilitates exchange yasper uses the PNML format. It support more popular Petri net extensions. Simulations are based on exact execution semantics and are very easy to set up and run. It is possible estimate overall throughput and efficiency. Yasper program is free to use ma the code is not free.

# 3 Definitions

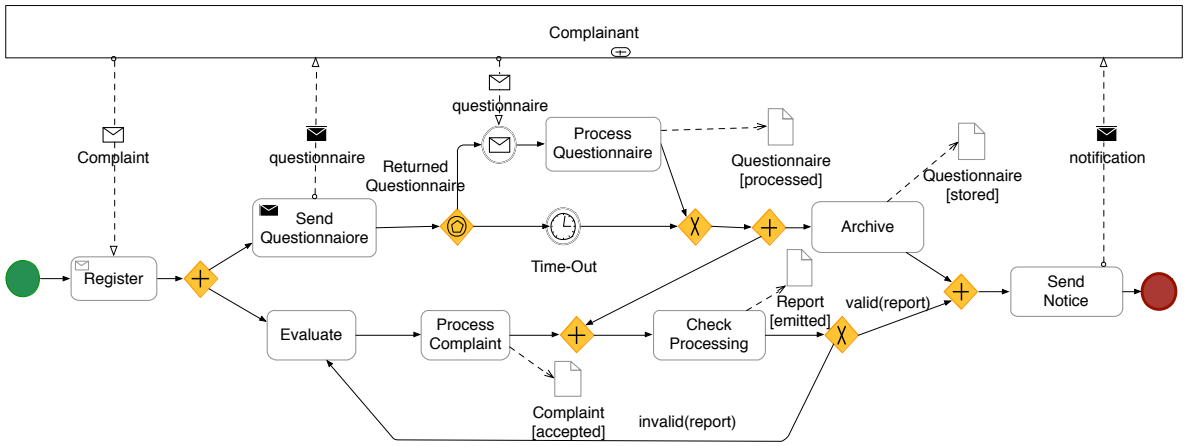



Figure 1: BPMN for the customer complaint resolution business process

The Business Process Modelling Notation (BPMN) is a OMG standard for capturing workflow models [19]. The variety of models spreads from simple workflow consisting of only a few sequential tasks to very complex models including data modeling and calls to external services or software systems in order to be executed by BPMN compliant process engines. An example of BPMN is shown in Figure 1 representing a typical workflow for customer complaint resolution.

Checking the correctness of a BPMN model assumes a fundamental importance for the industry. Whereas well-formedness may be checked by inspecting static properties of single components, there are more useful techniques that focuses on behavioral properties, able for instance of predicting deadlocks or the proper completion of the workflow.

This work focuses on the use of Petri-Nets in the context of workflow management as an established tool for modeling and analyzing processes and it introduces the SA-Net (Semantically Annotated workflow Net) , an extension of a WF-Net with semantic annotations.

## 3.1 Workflows and WF-Net

A Petri Net is a directed graph with two types of nodes called places and transitions, connected via directed arcs. Connections between two nodes of the same type are not allowed. The common notation graphically represents places by circles and transitions by rectangles.

**Definition 1** (Petri Net)**.** *A Petri Net is a tuple* $(P, T, F)$*:*

- $P$ *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,*

*A place* $p$ *is called an input place of a transition* $t$ *iff there exists a directed arc from* $p$ *to* $t$*. We use* $\bullet t$ *to denote the set of input places for a transition* $t$*. Place* $p$ *is called an output place of transition* $t$ *iff there exists a directed arc from* $t$ *to* $p$*. We use* $t\bullet$ *to denote the set of output places for a transition* $t$*. A transition* $t$ *is said to be enabled iff each input place* $p$ *of* $t$ *contains at least one token. An enabled transition* $t$ *may fire by consuming one token from each input place of* $t$ *and producing one token in each output place of* $t$*.*

*The global state at time* $\tau$ *of the petri net is given by the position of all its token, described through* $M$ *that is a mapping from* $P$ *to* $\mathbb{N}$*. This function associate to each place the number of token conatined in it.* $M_o \in M$ *is the initial marking of places.*

A Petri Net used to model a workflow is called Workflow Net (WF-net).

**Definition 2** (WF-Net)**.** *A Petri net* $PN = (P, T, F)$ *is a WF-net if and only if:*

- *PN has a special source place i ($\bullet i = \emptyset$) and a special sink place o ($o\bullet = \emptyset$);*
- *if we add a transition $t^*$ to PN which connects place o with i then the resulting Petri net is strongly connected.*

*A WF-net is said sound iff (i) starting from the initial state i it is always possible to reach the final state o, (ii) in the moment a token is put in place o all other places must be empty and (iii) there are no dead transitions in the state i.*

Intuitively, a WF-net models the execution of one instance of a workflow from its creation up to its completion, where tasks and events are modeled with transitions, input/output conditions are modeled with places and cases are modeled with tokens. The initial marking of a workflow net contains a single token located in the source place, and in principle, at least one token should reach the end place.

A precise description of the mapping is described in [1,5]. For instance, a BPMN gateway is generally mapped onto network with a transition for each alternative. A workflow process specified in terms of a Petri net has a clear and precise definition based on a firm mathematical foundation. Van der Aalst [1] proofed that Petri-net-based analysis techniques can be used to determine the correctness of a workflow process definition. For instance, the *absence of dead tasks* and the *proper completion* of a workflow can be verified by checking the soundness property of the corresponding WF-Net.

In particular a WF-Net $PN$ is sound if and only if $\overline{PN} = (P, T \cup \{t^*\}, F \cup \{\langle o, t^* \rangle, \langle t^*, o \rangle\})$ is live and bounded (where $t^*$ is a imaginary transition that connects $o$ with $i$). This can be checked in polynomial time for *free-choice WF-net*, a subset of WF-nets in which it is not allowed to mix choice and synchronization. This is a realistic assumption for most of the workflow management systems available at the moment.

## 3.2 Semantically Annotated Workflow Net

We consider the workflow management engine owns a (partial) knowledge about the environment in which the workflow is executed. We use the concept of State of the World for modeling the global state of execution for a specific workflow case during its lifecycle.

**Definition 3** (State of the World)**.** *The state of the world in a given time $\tau$ is a set $W^\tau \subset S$ where $S$ is the set of all the (non-negated) first order variable-free statements (facts) $s_1, s_2 \dots s_n$ that can be used in a given domain.*

*$W^\tau$ has the following characteristics:*

$$W^\tau = \{s_i \in S | (Bel\ a\ s_i)\} \tag{1}$$

*where a is the subjective point of view (i.e. the execution engine) that believes all facts in $W^\tau$ are true at time $\tau$; a fact is a statement to which it is possible to assign a truth value (i.e. $tall(john)$ or $likes(john, music)$).*

During the workflow execution, each BPMN element produces changes of the workflow state. Let us denote with $evolution(t)$ the change of state due to the corresponding transition $t$:

$$evolution(t) = W^{in} \xrightarrow{t} W^{out} \tag{2}$$

A State of the World is said to be consistent when $\forall s_i, s_j \in S$

$$\text{if } \{s_i, s_j\} \models \perp \text{ then } \begin{cases} s_i \in W^\tau \Rightarrow s_j \notin W^\tau \\ s_j \in W^\tau \Rightarrow s_i \notin W^\tau \end{cases} \tag{3}$$

i.e.: it contains only facts with no (semantics) contradictions.

Since facts are enough expressive for representing an object of the environment, a particular property of an object or a relationship among two ore more objects, $W^\tau$ describes a closed-world in which everything that is not explicitly declared is assumed to be false. An example of $W^\tau$ is shown in Figure 2, whereas, for instance the set $\{tall(john), small(john)\}$ is not a valid state of world since the two facts produce a semantic contradiction.

A semantic error is a run-time property of a workflow that occurs when the global state of the workflow contains inconsistencies (i.e. it violates Property 3).

**Definition 4** (Workflow Consistence)**.** *A workflow is said semantically consistent if all the states of execution can be expressed as consistent states of the world.*

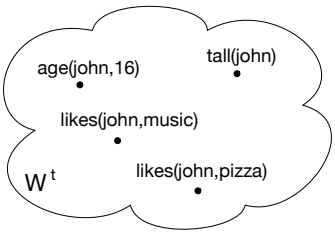



Figure 2: Example of a State of the World configuration at time $t$.

A Condition $\varphi : W^\tau \longrightarrow \{true, false\}$ of a state of the world is a logic formula composed by predicates or variables, through the standard set of logic connectives $(\neg, \wedge, \vee)$. A condition may be tested against a given $W^\tau$ through the operator of unification. For instance, the condition $\varphi = likes(Someone, music) \wedge age(Someone, 16)$ is true in the state of world of Figure 2 through the binding $Someone \mapsto john$ that realizes the syntactic equality.

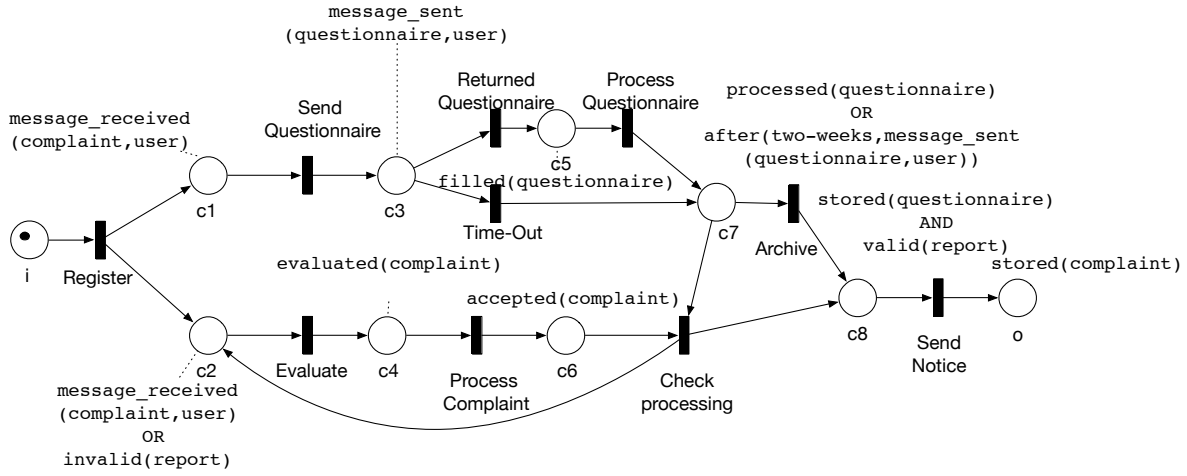



Figure 3: SA-Net for the customer complaint resolution business process

A SA-Net explicitly describes the evolution of the state of the world according to the execution of the corresponding workflow.

**Definition 5** (Semantically Annotated Workflow Net)**.** *A SA-Net $PN = (P, T, F)$ is a WF-Net in which each place $p$ is associated to a condition $\varphi_p$ to be tested over the state of the world:*

- *if $p = \bullet t$ then the condition $\varphi_p$ must be true in order the subsequent transition can fire: $\varphi_p(W^{in}) = true$,*
- *if $p = t\bullet$ then the condition $\varphi_p$ must be true after predecessor transition is terminated: $\varphi_p(W^{out}) = true$.*

**Example 1.** *Figure 3 shows an instance of SA-Net built for illustrating a business process frequently used in literature [1, 8]. First the complaint is registered, then in parallel a questionnaire is sent to the customer, and the complaint is evaluated. The questionnaire must return within two weeks, otherwise it is not used. Based on the result of the evaluation the complaint is processed or not. The processing of a complaint is synchronized with the result of the incoming questionnaire (or the time-out). The result is a report that must be supervised. If the supervision returns ok, then the report is stored and sent to the customer. In figure each place is associated to a variable-free logical sentence. This does not variate the normal activation rules for a petri-net but it rather enrich transitions with explicit meaning. Indeed, for instance, focusing on the Process Complaint activity, it fires when [accepted(complaint) OR rejected(complaint) OR invalid(report) ] is true and it produces a new state where [processed(complaint)] is true.*

# 4 From BPMN to SA-Net

A SA-Net can be automatically produced from a BPMN well-formed model. The only assumption we make is that the BPMN model include input/output data objects and incoming/outgoing messages for each task or event of the workflow. The underlying idea is to take into account the effect that each BPMN element produces in the execution contest.

We studied the impact in terms of state change of each activity, event and gateway with respect to the global state of workflow. This gives us the opportunity for identifying the set of facts that correspond to each place in a WF-net.

Our focus is on the three main elements of BPMN, i.e. Activities, Events and Gateways. Let us begin observing that gateways do not actually affect the state of the world, they act letting or blocking the passage of the control flow from some of their incoming to some of their outgoing sequenceFlows. Conversely, event elements, by definition, capture an exogenous modification of the current state of the world, whereas activity elements produce an endogenous change of the state of the world, observed immediately before their execution.

**Example 2.** *Let us consider the activity send_questionnaire of the complaint process: it produces an outgoing message (directed to the user): the corresponding state is described by the fact* $sent(questionnaire, user)$*. On the other hand the event returned_questionnaire generates a new data object* $questionnaire[filled]$*: the corresponding state is described by the fact* $filled(questionnaire)$*.*

The above conditions reveal to be not enough to form the State of World we aspire since these are independent from the element position inside the workflow, that is they do not take account of the mutual incidence of neighbors BPMN elements.

Therefore, observing with more attention a transition $t$ we can distinguish a couple of factors (one is internal to $e$ and one is external due to $e$ position) that contribute to the $evolution(e)$.

The ***internal factors*** describe how the element $e$ contributes to the evolution of the state of the world. Formally we introduce the waited state and the generated state.

**Definition 6** (Waited State)**.** *The Waited State for a BPMN element* $e$*, denoted by* $ws(e)$*, is the condition over* $W^{in}$ *whose satisfaction is necessary in order for the element to be internal ready for activation.*

**Definition 7** (Generated State)**.** *The Generated State (GC) for a BPMN element* $e$*, denoted by* $gs(e)$*, is the condition that must be true over* $W^{out}$ *as a consequence of the execution of* $e$*.*

The waited state internal factor covers all the element requirements whose fulfillment is determinant for it to be executed (availability of input data, incoming messages, etc.) assuming it was already reached by a token. On the other hand, the generated state internal factor takes into account the result of all the element actions: output data, outgoing messages, etc.

Conversely, the ***external factors*** describe mutual interactions of an element $e$ with its neighborhoods in the network. Formally, we introduce the predecessors influence and the successor influence.

**Definition 8** (Predecessors Influence)**.** *The Predecessors Influence for a BPMN activity or event* $e$*, denoted by* $pre\_inf(e)$*, is the condition that must be true over* $W^{in}$ *in order to connect* $e$ *with its predecessor elements in the network.*

**Definition 9** (Successors Influence)**.** *The Successors Influence for a BPMN activity or event* $e$*, denoted by* $succ\_inf(e)$*, is the condition that must be true over* $W^{out}$ *in order to connect* $e$ *with its successor elements in the network.*

According to all the previous definitions we can state that:

$$\begin{aligned} \varphi_{\bullet e} &= ws(e) \wedge pre\_inf(e) \\ \varphi_{e\bullet} &= gs(e) \wedge succ\_inf(e) \end{aligned} \tag{4}$$

## 4.1 How to Calculate Internal Factors

The waited state and the generated state are individual characteristic of each BPMN element and they are constructed by composing logical expressions as described in this subsection.

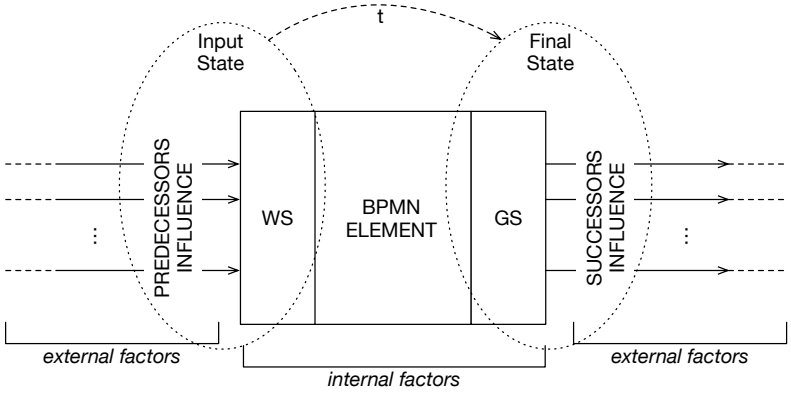



Figure 4: A generalized model for BPMN elements semantic

#### 4.1.1 Data Input/Output Condition

Activities and events activation is subject to the availability of the their input and it produces a definite output, thus attaining a diverse state of the world.

BPMN provides several constructs to model data within the workflow, primarily DataStores, DataObjects, DataInputs/DataOutputs. These BPMN elements could be associated to activities and events via DataInputAssociations /DataOutputAssociations to specify their data requirements and results.

We indicate the condition of availability of a data with a fact in the form $available(\langle Data \rangle)$ where available is a functor and $\langle Data \rangle$ is a placeholder for the name of the input/output artifact. If a state is specified for a data object then we use a predicate in which the functor is the string corresponding to object state (i.e $filled(questionnaire)$).

Depending if the data object appears as input or output for a BPMN element $e$, we refer to the condition either as Data Input Conditions, denoted by $data\_in(e)$, or as Data Output Condition, indicated by $data\_out(e)$.

**Example 3.** *Let us consider the activity process questionnaire of the compliant process: it produce as output a DataObject; the corresponding state is described by the predicate processed(questionnaire).*

#### 4.1.2 Message Received/Sent Condition

Workflows might be composed of several collaborating parties, each modeled as a separate process but communicating each other via messages exchange.

We point out that an activity or event which is destination of messageFlows is unable to activate until it receives all the prescribed messages. This condition is represented by a fact in the form $message\_received(\langle Message \rangle, \langle Actor \rangle)$ where $\langle Message \rangle$ is a placeholder for the name of the BPMN message element arriving to $e$ and $\langle Actor \rangle$ is the name of the collaborator.

Conversely an activity or event which is source of messageFlows produces a new state of the world in the moment messages are sent. This condition is represented by a fact in the form $message\_sent(\langle Message \rangle, \langle Actor \rangle)$ where $\langle Message \rangle$ is a placeholder for the name of the BPMN message element departing from $e$ and $\langle Actor \rangle$ is the name of the collaborator.

Depending if the message arrives to or departs from the BPMN element $e$, we refer to the condition either as Message Received Condition, denoted by $mess\_in(e)$, or as Message Sent Condition, denoted by $mess\_out(e)$.

#### 4.1.3 Waited/Generated Event Condition

BPMN events are extremely numerous and varied. Each of them catches or throws a specific 'state of affair' within the workflow execution. Generalizing, the relative conditions are expressed by facts in the form $caught(\langle event \rangle)$ or $thrown(\langle event \rangle)$, where $\langle event \rangle$ is a placeholder for specific category of events.

Depending if the event is catched or thrown by the BPMN element $e$, we refer to the condition either as Waited Event Conditon, denoted by $event\_in(e)$ or Generated Event Condition $event\_out(e)$.

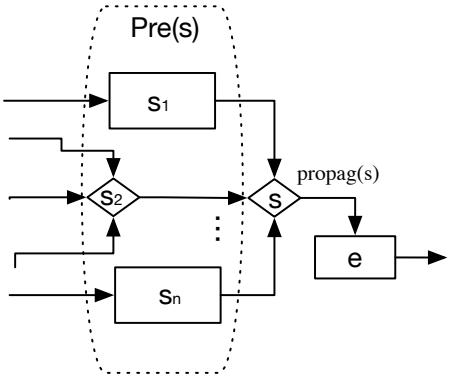



Figure 5: Example of Back to Forward Propagation in which $propag(s) = gs(s_1) \vee propag(s_2) \vee \ldots \vee gs(s_n)$.

#### 4.1.4 Boundary Condition

BPMN activities may fail, be terminated or compensated before its completion. Indeed BPMN provides a way to model, via the boundary interrupting events, these exceptional situations, thus allowing to make the workflow able to react in some way.

In order to distinguish the normal termination of the activity we introduce the Termination Condition, denoted by $termination(e)$. When the activity terminates correctly then it assumes the form $done(\langle e \rangle)$ where $\langle e \rangle$ is the label of the activity. Otherwise, if the activity is interrupted by the triggering of a boundary event, then $termination(e)$ assumes the form of $\langle boundary\_event \rangle(\langle e \rangle)$, where $\langle boundary\_event \rangle$ is a placeholder for the kind of boundary event has triggered.

Summarizing, if $e$ is an activity:

$$\begin{cases} ws(e) = data\_in(e) \wedge mess\_in(e) \\ gs(e) = data\_out(e) \wedge mess\_out(e) \wedge termination(e) \end{cases} \tag{5}$$

If $e$ is an event:

$$\begin{cases} ws(e) = data\_in(e) \wedge mess\_in(e) \wedge event\_in(e) \\ gs(e) = data\_out(e) \wedge mess\_out(e) \wedge event\_out(e) \end{cases} \tag{6}$$

Otherwise $ws(e) = gs(e) = true$ if $e$ is a gateway.

**Example 4.** *Let us consider a BPMN activity called "Flight Booking", which takes in input a data object "Client Credential [verified]" and the "Flight Id", and it produces an outgoing message "Receipt" for the client. A boundary error event is also specified for capturing the unavailability of the resource. This activity has the following ws and gs:*

$$\begin{aligned} ws(flight\_booking) =& verified(client\_credential) \\ gs(flight\_booking) =& message\_sent(receipt, user) \\ & \vee error(flight\_booking) \end{aligned}$$

### 4.2 How to Calculate External Factors

The predecessors/successors influence are 'social' characteristic of a BPMN element that depends on its position inside the whole network. They are constructed by composing logical expressions as described in this subsection.

This external factor is built by observing that whereas activities and events act by changing the state of the world as the result of internal activities (see Section[4.1]) conversely gateways rather propagate and combine the state from their inputs to outputs. For calculating the predecessors influence ($pre\_inf(e)$) in a generic BPMN element $e$ it is necessary to inspect the state propagated (forward) from all the elements $s \in Pre(e)$.

**Definition 10** (Back to Forward Propagation)**.** *Given $e$, a generic BPMN element, $propag(e)$ describes the state observed from a generic output sequence flow with respect to the state observed at all the input flows.*

therefore :

$$pre\_inf(e) = \bigvee_{\forall s \in Pre(e)} (propag(s)) \tag{7}$$

Similarly, for calculating the successors influence ($succ\_inf$) in a generic BPMN element $e$, it is necessary to inspect the state required (backward) by all the elements in $s \in Succ(e)$.

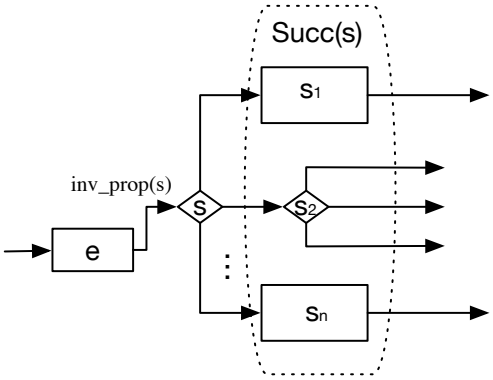



Figure 6: Example of Inverse Propagation in which $inv\_prop(s) = ws(s_1) \vee inv\_prop(s_2) \vee \ldots \vee ws(s_n)$.

**Definition 11** (Inverse Propagation)**.** *Given e, a generic BPMN element, inv_prop(e) describes the state observed from a generic input sequence flow with respect to the state observed at all the output flows.*

therefore:

$$succ\_inf(e) = \bigvee_{\forall s \in Succ(e))} (inv\_prop(s)) \tag{8}$$

Gateways act as backward/forward state pass-through (propagators of state). In other words the state measured at one of the output sequence flow of the gateway depends on the state at all the inputs, and conversely the state measured at one of the input sequence flows depends on the state at all the outputs.

If $e$ is a XOR gateway:

$$\begin{cases} propag(e) = \bigvee_{\forall s \in Pre(e)} (propag(s) \wedge flow(e,s)) \\ inv\_prop(e) = \bigvee_{\forall s \in Succ(e)} (inv\_prop(s) \wedge flow(e,s)) \end{cases} \tag{9}$$

where $flow(e, s)$ is the condition associated to the sequence flow that connects $e$ to $s$.

If $e$ is a parallel gateway:

$$\begin{cases} propag(e) = \bigvee_{\forall s \in Pre(e)} (propag(s) \wedge flow(e,s)) \\ inv\_prop(e) = \bigwedge_{\forall s \in Succ(e)} (inv\_prop(s) \wedge flow(e,s)) \end{cases} \tag{10}$$

If $e$ is an activity or an event, $propag(e) = gs(e)$, whereas $inv\_prop(e) = ws(e)$. A special note concerns BPMN timer event entities which act as state pass-through only for inverse propagation:

$$\begin{aligned} inv\_prop(e) = after(&\langle time \rangle, \\ &\bigwedge_{\forall s \in Succ(e)} (inv\_prop(s) \wedge flow(e,s)) \end{aligned} \tag{11}$$

where $\langle time \rangle$ is a placeholder for the relative absolute time specification of the time event.

# 5 Global Consistency Check

In this section we show one of the practical application of SA-Net (Semantically Annotated workflow Net) . In particular let us assume to have as input a SA-Net obtained (as illustrated in Section 4) starting from a sound WF-Net as defined by Van der Aast in [1] and enriched with information about states of the world.

## 5.1 Inconsistencies and Logical Errors

A WF-Net is a great instrument to check at design time the correctness of a workflow in terms of properties such as the presence of deadlock and the proper completion. This technique does not allow to connect the global behavior of the network of elements with the concrete job done by each single activity. Indeed it considers all the transitions of the petri net are (semantically) equal.

Conversely, for testing the logical consistence of a workflow, the analyst should inspect all its possible states of world. Nevertheless $\{W^1, W^2, \ldots, W^m\}$ are accessible only at run-time.

We follow a different strategy for testing the workflow consistence at design-time. In place of states of world, we consider the logical conditions associated to each place. In a SA-Net , when a place $p$ contains a token then the associated condition $\varphi_p$ must be true. This consideration allows for constraining the admissible states of the world and therefore deducting when there are inconsistencies.

Intuitively, given a condition $\varphi$, there may exist many states of the world in which the condition may be true. In particular, by considering that i) logical conditions associated to places are logical formula, and ii) these are built as disjunctive normal form of positive terms or conjunctions of positive terms (see Section 4):

$$\varphi = \bigvee_{i=1}^{k} f_i$$

we can build the minimal states of the world $B_i$ in which $\varphi = true$ as follows:

$$\forall i, B_i = \begin{cases} \{f_i\} & \text{if } f_i \text{ is a term} \\ \{f_i^A, f_i^B\}, & \text{if } f_i = f_i^A \wedge f_i^B \end{cases}$$

We can say that given a place $p$, $\varphi_p$ generates a base for all the admissible states of the world:

$$\forall W_j : \varphi_p(W_j) = true, \exists B_i : W_j \supset B_i$$

**Definition 12** (Condition Consistence)**.** *Given a couple of logical condition $\varphi_1$ and $\varphi_2$ expressed in DNF (disjunctive normal form)*

$$\varphi_1 = \bigvee_{i=1}^{k} f_i, \varphi_2 = \bigvee_{j=1}^{r} g_i$$

*we say they are semantically consistent when*

$$\forall f_i \in \{f_1, f_2, \ldots, f_k\}, g_j \in \{g_1, g_2, \ldots, g_r\}, f_i \wedge g_j \vdash \top$$

*A set $\Phi = \{\varphi_1, \varphi_2, \ldots, \varphi_k\}$ is semantically consistent if $\forall \varphi_i, \varphi_j \in \Phi (i \neq j)$, $\varphi_i$ is consistent with $\varphi_j$.*

**Example 5.** *Let us suppose the two conditions $\varphi_1 = red(car) \vee running(car)$ and $\varphi_1 = blu(car) \wedge sporty(car)$. These are not consistent because $red(car) \wedge blu(car) \vdash \bot$. In order to illustrate better this point, let us assign these two conditions to a couple of input places of the same transition (Figure 7.a), we can observe 2 equivalent SA-Net (Figure 7.b-c) in which the conditions have been unfold. The rationale of the inconsistency relies on the SA-Net in Figure 7.b in which the two tokens generate a condition that require a state of the world $W^\tau \supseteq \{red(car), blu(car)\}$ that violates Property 3 of Definition 3. Therefore a valid state of world in which $red(car) \wedge blu(car) = true$ can not exist.*

This intuition is formalized and generalized in the following theorem:

THEOREM 1. Let $\Phi = \{\varphi_1, \varphi_2, \ldots, \varphi_k\}$ be a set of conditions in DNF where clauses are only positive terms or conjunctions of positive terms. Let be $G$ the set of states of the world in which all conditions of $\Phi$ are true at the same time.
A necessary condition that $G$ contains only consistent states of the world is that $\Phi$ is semantically consistent. □

*Necessary condition.* All the $W_i^\tau \in G$ are consistent, therefore $\forall s_i, s_j \in W_i^\tau \Rightarrow s_i \wedge s_j \nvdash \bot$.

$$\forall \varphi_1, \varphi_2 \in \Phi, \varphi_1 = \bigvee_{i=1}^{k} f_i, \varphi_2 = \bigvee_{j=1}^{r} g_i$$

If $\varphi_1 = true$ and $\varphi_2 = true$ at time $\tau$, then for each couple of positive terms $\langle f_i, g_j \rangle \exists W_{ij}^\tau : f_i, g_j \in W_{ij}^\tau$ therefore $f_i \wedge g_j \nvdash \bot$. As conclusion we can deduct that $\Phi$ is semantically consistent. □

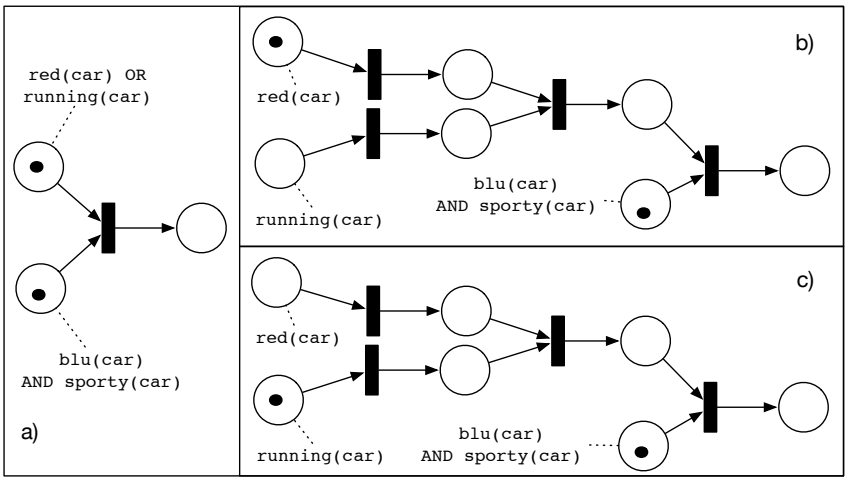



Figure 7: Equivalence in a SA-Net . On the left side (a) the starting SA-Net in which a place is associated to an OR condition. On the right (b-c) two equivalent SA-Net in which the OR condition has been unfold.

The importance of Theorem 1 is illustrated in the following two corollaries for testing local and global consistency of a workflow at design-time.

**Corollary 2** (Local Consistency)**.** *Given a transition $t$ of a SA-Net , in which $\bullet t = \{p_1, p_2, \ldots p_n\}$ associated to a $\Phi = \{\varphi_1, \varphi_2, \ldots, \varphi_k\}$. When enabled, the transition $t$ may fire only if $\Phi$ is semantically consistent.*

**Corollary 3** (Global Consistency)**.** *Given a SA-Net associated to a workflow in a global state $M^\tau$. Let $\Phi = \{\varphi_1, \varphi_2, \ldots, \varphi_k\}$ be the set of conditions associated to places with (at least) a token according to $M^\tau$. The workflow is logically consistent only if $\Phi$ is semantically consistent.*

Identifying logical errors present in a workflow in the design time can be useful in order to prevent any problems at run-time and to make the appropriate corrections before the process is executed. This is important for increasing the value of a business system. In the business analysis activity, the evaluation of the logical correctness of a workflow process at design time can be useful to prevent economic losses due to the malfunction of processes erroneously executed.

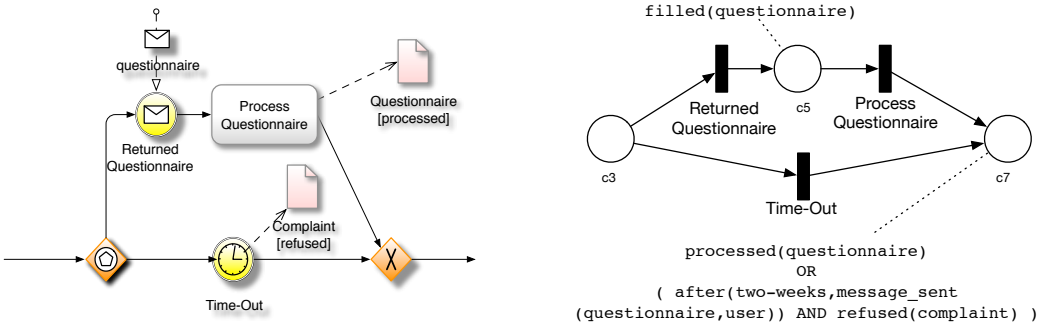
**Example 6.** *Figure 1 shows the BPMN diagramm for the process of Complaint; for such process has been defined the corresponding Petri net obtaining the SA-Net shown in Figure 3, in this network for each transition corresponding to a specific task have been defined the respective trigger conditions and final states for its input and output places.*

*The process shown in Figure 1 is correct from the point of view logic, then the corresponding WF-net defined is valid. However, we intend to demonstrate the importance of the information on goals to add to a WF-net. To do this we have introduced a logic error in the BPMN diagram on the process complaint, in particular, it was assumed that the entire complaint is rejected if attended by a time out and the questionnaire has not yet been received by the customer(see Figure 8a); however, when you run the task process complaint the complaint is accepted.*

*The example shown in Figure 3 has been obtained by mapping the trigger condition and final states in the appropriate places corresponding to the transitions, in the generated WF-net it is easily identified a logic problem in correspondence with the transition "check processing", such transition is in fact activated when there are tokens in places respectively c6 and c7.*

*In such transitions as you can see the states correspond respectively processed (Quest) OR refused (Complaint) for the post c7 and accepted (Complaint) for the post c6, which are mutually conflicting, the transition check processing can not therefore ever be activated and at runtime the process should be in deadlok; this problem is not immediately detectable in the corresponding BPMN diagram (Figure 8a). A further problem that can be easily identified in a WF-net extended with information about the goals is that relating to the state of inconsistency of data (Figure 3).*

*To demonstrate this we appropriately modified the original complaint process by introducing a task of verification(Verify existing tipology complaint) of the various types of complaint previously managed; it was assumed that the sending of a new questionnaire to a user is subject to the control of the same type in the past managed, if so the complaint is immediately accepted and start chek phase of the process,*




(a) BPMN (b) SA-Net

Figure 8: On the left a particular of the BPMN for the complaint process in which the process waits for a filled questionnaire from the user. With respect to Figure 1, in this fragment, the *time-out* event changes the state of the *Complaint* to 'refused'. On the right a particular of the corresponding SA-Net .

*otherwise the process is running in the previous mode. This case is shown in Figure 10), as is easy to see the places c5 and c6, corresponding respectively to the states refused (Complaint) and accepted (Complaint) can be reached by a token in the same time because the transitions are executed in parallel. If such event occurs the system would be in a state of inconsistency because the dataObject Complaint would simultaneously with two opposite states.*

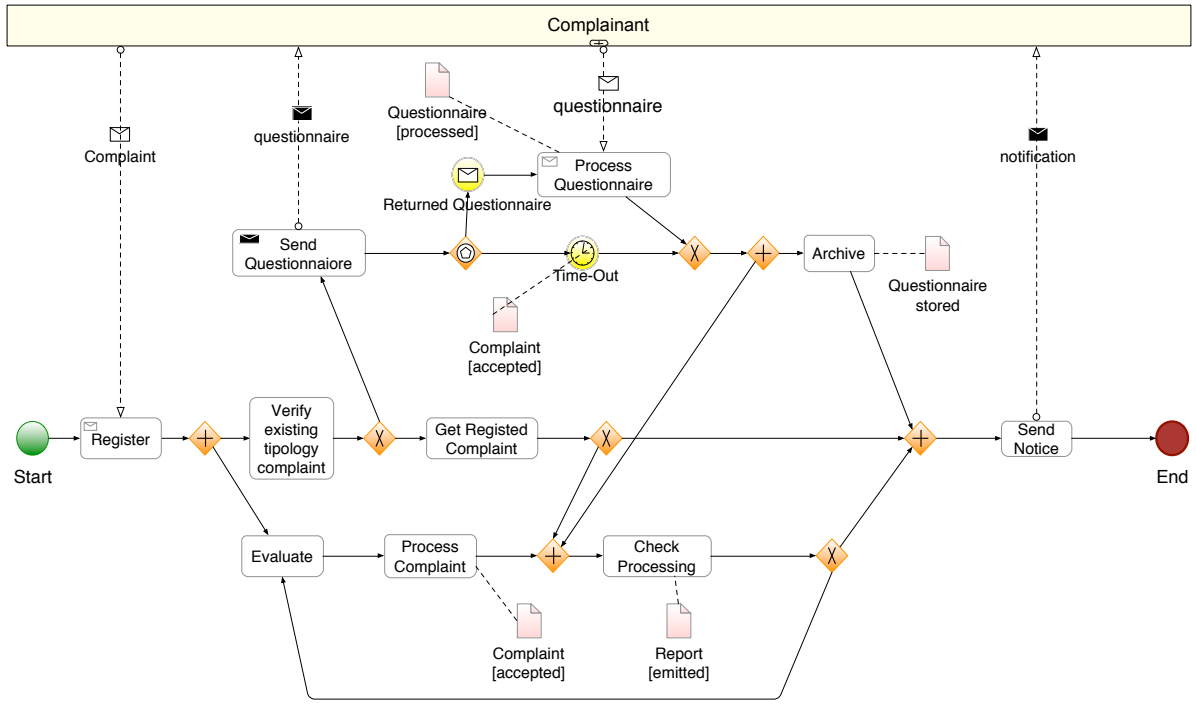



Figure 9: BPMN: Process Complaint with introduction verification task

## 5.2 The General Technique for Global Consistency Check

We propose an algorithm for checking whether a workflow contains logical errors.

***Global Consistency Check Algorithm***: given as input a SA-Net,network built from its workflow. Build the tree coverage of the network, which is a two-dimensional array with dimensions corresponding respectively to the states and to the places allowed in the net. For each places (Pn) of the network if it can possess a token at state m sets the correspondent value in matrix C with true, false otherwise;

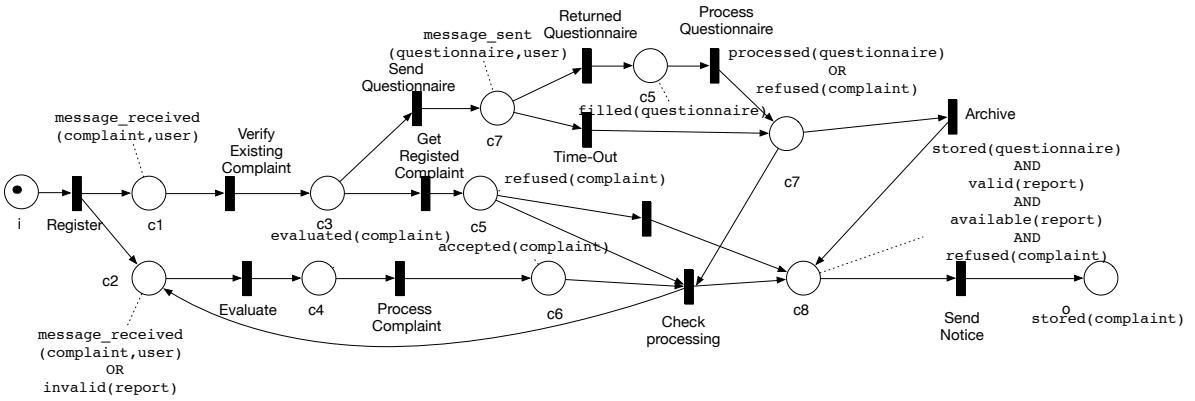



Figure 10: SA-Net for the Process Complaint with introduction verification task

create an array (SnF) to store all the logic functions (F) for each state of the workflow. Build each logic function as the linear combination of the values associated with places with true value for each state. Finally check the logical correctness of the workflow by analyzing all the functions generated, the defined workflow is valid if all the functions are logically correct; otherwise it is possible to identify the block which is a source of error.

**Algorithm 1** Semantic correctness SA-Net Algorithm

**Require:** a SA-Net $SN$
**Ensure:** validity
1. given a SA-Net $SN$
2. $C = coverabilityTree(SA - Net)$
3. $SnF = array\{m\}$
4. **for all** $m \in C$ **do**
5. $F_m = logicalFunction(m)$
6. add $F$ to $SnF$
7. **end for**
8. **for all** $F_m \in SnF$ **do**
9. **if** $F_m$ is not logically correct **then**
10. **return** The workflow model is invalid: the error is in $F_m$
11. **end if**
12. **end for**
13. **return** The workflow model is valid

In the follows the execution of the algorithm is explained in details, using as example the BPMN represented in Figure 9.

Let us consider the workflow show in Figure 8a and the relative SA-Net as defined in Figure 8b we can construct its coverabilty tree as well as defined in Algorithm 1 which is compoded as follows

$$C_{m,n} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ \vdots & \dots & \dots & \dots & \dots & \dots & \dots & \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{0} \\ \vdots & \dots & \dots & \dots & \dots & \dots & \dots & \end{bmatrix} \tag{12}$$

Now we create the logic functions corresponding to the various states of our workflow, or a function for each row of the matrix coverage. The highlighted row in the matrix corresponds to the state of activation of the transition *check processing*. If we verify the correctness of the corresponding logic function which is thus defined:

$F_m$ = *true* AND *true* AND *true* AND *true* AND *true* AND
**accepted(Complaint)** AND *processed(Complaint)* OR **refused(Complaint)**

We note that the $F_m$ function is not logically correct because the two prepositions **accepted(Complaint)** and **refused(Complaint)** are opposite to each other so their logical AND leads the system in a state of deadlock.

# 6 Conclusions

The possibility to prove validity to a workflow model is a key for ensuring success and saving cost for industries. Finding at design time logical errors is more difficult. In this paper we propose an approach for checking the semantic consistency to a workflow model. BPMN is the main standard to define workflow models. The Petri net theory can support the management of workflows. For this purpose the WF-net(Workflow Net) is used for modeling the execution of one instance of a workflow. Van der Aalst [1]shows how Petri-net-based analysis techniques can be used to determine the correctness of a workflow process definition. The soundness of a WF-net ensures for instance absence of dead task and the correct termination of the entire process. In the proposed approach the global state of execution, for a specific workflow ,is defined as the State of the World. At run time each BPMN element produces change of this state, if an inadmissible state is reached then the workflow can have a semantic error. We define the SA-Net construct for describing the evolution of the state of the world during a workflow execution. It is illustrated how a SA-Net can be automatically produced from a BPMN well-formed model that includes data object and message for each task or event in the workflow. This allows to identify logical condition, in place of state of the world, for each place of the corresponding WF-net. We propose an algoritm for identifying bugs in the design time in order to prevent any problems at run-time. For this purpose we use a two-dimensional matrix, derived by the petri net coverability tree, for buinding the logical function of each state of the considered workflow. The correctness of this logical function ensure that the workflow is free of semantic errors.

# References


[1] W. M. Van der Aalst, "The application of petri nets to workflow management," *Journal of circuits, systems, and computers*, vol. 8, no. 01, pp. 21–66, 1998.

[2] H. Cao, H. Jin, S. Wu, and S. Ibrahim, "Petri net based grid workflow verification and optimization," *The Journal of Supercomputing*, vol. 66, no. 3, pp. 1215–1230, 2013.

[3] H. H. Bi and J. L. Zhao, "Applying propositional logic to workflow verification," *Information Technology and Management*, vol. 5, no. 3-4, pp. 293–318, 2004.

[4] A. Awad, G. Decker, and M. Weske, "Efficient compliance checking using bpmn-q and temporal logic," in *Business Process Management*. Springer, 2008, pp. 326–341.

[5] R. M. Dijkman, M. Dumas, and C. Ouyang, "Semantics and analysis of business process models in bpmn," *Information and Software Technology*, vol. 50, no. 12, pp. 1281–1294, 2008.

[6] W. M. Van der Aalst, B. F. van Dongen, C. W. Günther, A. Rozinat, E. Verbeek, and T. Weijters, "Prom: The process mining toolkit." *BPM (Demos)*, vol. 489, p. 31, 2009.

[7] R. Yu, Z. Huang, L. Wang, and H. Zhang, "Analyzing bpmn with extended object petri net." *Journal of Software Engineering*, vol. 8, no. 2, 2014.

[8] H. A. Reijers, T. Freytag, J. Mendling, and A. Eckleder, "Syntax highlighting in business process models," *Decision Support Systems*, vol. 51, no. 3, pp. 339–349, 2011.

[9] A. Eckleder and T. Freytag, "Woped 2.0 goes bpel 2.0." *AWPN*, 2008.

[10] W. M. Van Der Aalst and A. H. Ter Hofstede, "Verification of workflow task structures: A petri-net-baset approach," *Information systems*, 2000.

[11] A. Ter Hofstede and E. Nieuwland, "Task structure semantics through process algebra," *Software Engineering Journal*, 1993.

[12] E. Verbeek and W. M. Van Der Aalst, "Woflan 2.0 a petri-net-based workflow diagnosis tool," in *International Conference on Application and Theory of Petri Nets*. Springer, 2000, pp. 475–484.

[13] A. Rozinat, M. T. Wynn, W. M. van der Aalst, A. H. ter Hofstede, and C. J. Fidge, "Workflow simulation for operational decision support," *Data & Knowledge Engineering*, 2009.

[14] W. M. Van Der Aalst and A. H. Ter Hofstede, "Yawl: yet another workflow language," *Information systems*, vol. 30, no. 4, pp. 245–275, 2005.

[15] A. V. Ratzer, L. Wells, H. M. Lassen, M. Laursen, J. F. Qvortrup, M. S. Stissing, M. Westergaard, S. Christensen, and K. Jensen, "Cpn tools for editing, simulating, and analysing coloured petri nets," in *International Conference on Application and Theory of Petri Nets*. Springer, 2003, pp. 450–462.

[16] M. Kovács and L. Gönczy, "Simulation and formal analysis of workflow models," *Electronic Notes in Theoretical Computer Science*, 2008.

[17] E. Mikk, Y. Lakhnech, M. Siegel, and G. J. Holzmann, "Implementing statecharts in promela/spin," in *Industrial Strength Formal Specification Techniques, 1998. Proceedings. 2nd IEEE Workshop on*. IEEE, 1998, pp. 90–101.

[18] K. van Hee, O. Oanea, R. Post, L. Somers, and J. M. van der Werf, "Yasper: a tool for workflow modeling and analysis," *Application of Concurrency to System Design, 2006. ACSD 2006. Sixth International Conference on*, 2006.

[19] B. P. M. OMG, "Notation (BPMN) Version 2.0 (2011)," *Available on: http://www. omg. org/spec/BPMN/2.0*, 2011.