Analysis of BPMN Models

Addis Gebremichael

addisalemayehu.gebremichael@student.uantwerpen.be

Abstract

The Business Process Modeling Notation (BPMN) is a standard notation for capturing business processes, especially at the level of domain analysis and high-level systems design. The diverse constructs found in BPMN and the lack of an unambiguous definition of the notation makes it possible to create models with semantic errors. Such errors can lead to significant system development associated costs and can also be difficult to recover. Hence, it is quite essential to verify the semantic correctness of business process models designed with BPMN. This paper, thus, discusses a mapping from BPMN to a formal language, namely Petri nets, for which semantic analysis of BPMN models can be made.

Keywords: BPMN, business process modeling, petri nets, model transformations, denotational semantics

1. Introduction

The Object Management Group (OMG) has developed a standard Business Process Model and Notation (BPMN)[1]. The primary goal of BPMN is to provide a notation that is readily understandable by all business users, from the business analysts that create the initial drafts of the processes, to the technical developers responsible for implementing the technology that will perform those processes, and finally, to the business people who will manage and monitor those processes[1].Thus, BPMN creates a standardized bridge for the gap between the business process design and process implementation [2].

It is known that models defined in graph-oriented process definition family of languages may exhibit a range of semantic errors, including deadlocks and livelocks[3]. BPMNs follow a similar paradigm, and such errors are especially problematic at the levels of domain analysis and high-level systems design, because errors at these levels are among the hardest and most costly to correct.

This paper takes on the challenge of providing a denotational semantics for a subset of BPMN. The semantics is defined as a mapping between BPMN and Petri nets. The choice of using plain Petri nets as a target for the mapping is motivated by the availability of efficient static analysis techniques and the shared properties inherent to both formalisms. Thus, the proposed mapping not only serves the purpose of disambiguating the core constructs of BPMN,

but it also provides a foundation to statically check the semantic correctness of BPMN models.

To support this claim, as discussed in the experiment section of this paper, the first task was to develop a BPMN formalism. By exploiting an existing Petri Nets formalism in the working tool set AToMPM, it was also possible to perform model transformation for mapping BMPN onto Petri nets.The following section thoroughly highlights the semantic analysis conducted on sample models that support BMPNs, followed by a conclusion to the overall experiment conducted.

2. Experiment

This section introduces the BPMN formalism built in AToMPM,an open-source framework for designing domain-specific modeling environments, performing model transformations, manipulating and managing models[4]. Furthermore, this section also discusses the mapping of BPMN onto Perti Nets which utilizes another essential feature of the same tool, i.e. model transformations.

2.1. BPMN formalism

For building a BPMN formalism, an abstract syntax meta-model and a visual concrete syntax had to be developed. The meta model represented in figure 1 depicts the classes and their associations for a subset of BPMN. This experiment focuses on the control-flow perspective of BPMN, that is, the subset of the notation that deals with the order in which activities and events are allowed to occur. The inclusion of pools, which are organizational features of BPMN, was intended to demonstrate the semantic behavior of message flow between two or more atomic processes contained in pools. Also to note is the multiple constraints imposed on many of the classes. The constraints, discussed below, arise due to facilitating the definition of the mapping by introducing a notion, "well-formed BPMN process".
Such a process has the following characteristics:

- A start event has just one outgoing sequence flow but no incoming flow;
- An end event has just one incoming sequence flow but no outgoing sequence flow;
- Activities and intermediate events have exactly one incoming sequence flows and one outgoing sequence flow;
- Fork or decision gateways have one incoming sequence flow and more than one outgoing sequence flows;
- Join or merge gateways have one outgoing sequence flow and more than one incoming sequence flow; And

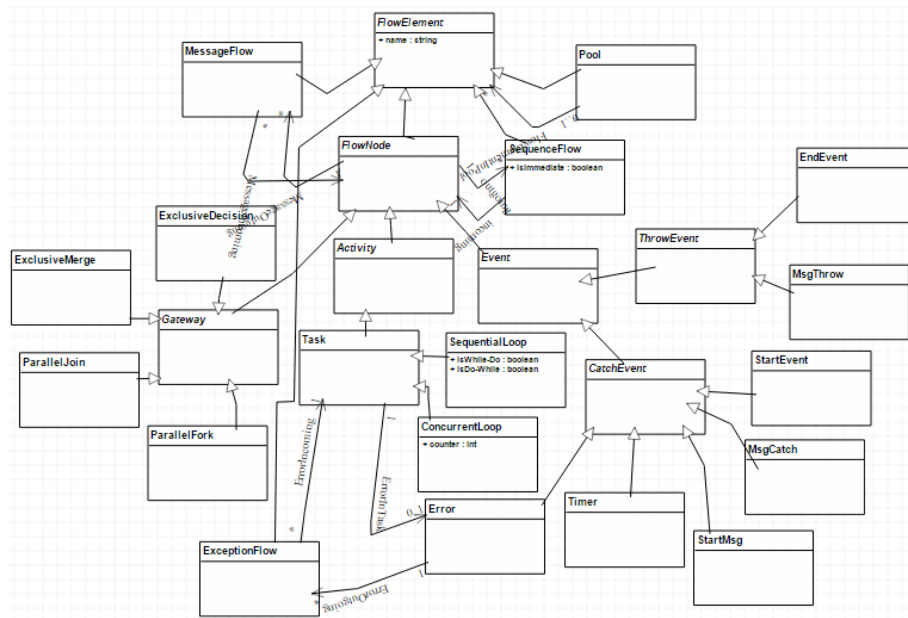


Figure 1: BPMN Abstract Syntax Meta Model

- An error event has no incoming and outgoing sequence flow, however, has one outgoing exception flow and an incoming containment association with task.

The first two of the above constraints are syntactic restrictions inherent to the definition of start and end events in BPMN[1]. The latter three conditions are not part of the syntactic restrictions of BPMN, but are introduced in order to simplify the presentation of the mapping. Thus, it is important to note that these three constraints do not affect the generality of the mapping and that it was possible to have achieved similar "well formed BPMN models" by applying transformation rules, which can be tedious and trivial. The last one is there because a subset of the BPM notation i selected for this experiment has an error event that is a catch event that responds to an exception thrown while a certain task is executing.

The concrete syntax presented in figure 2, depicts visual notations that more or less resemble BPMN as per version 2.0.[1]. Coloring and some textual information has been applied on some of the notations to promote perceptual discriminability.

2.2. Mapping BPMN onto Petri Nets

The mapping of BPMN onto petri nets was done using model transformations in AToMPM. The multi-paradignm modelling environment now has both

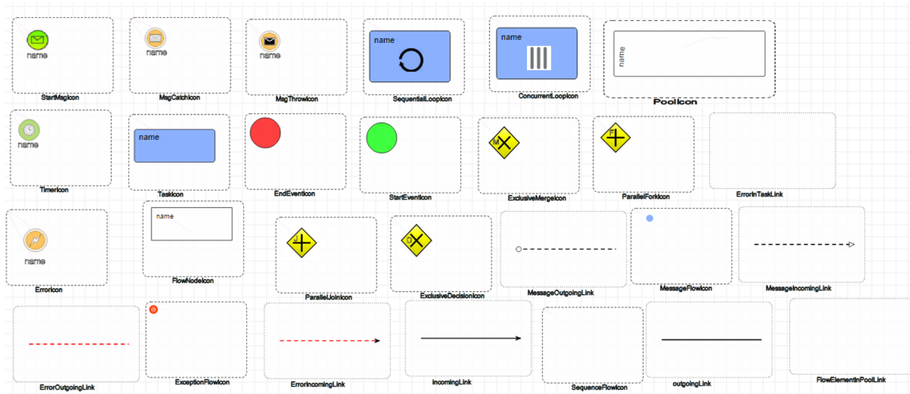


Figure 2: BPMN Visual Concrete Syntax

formalisms and thus enables for transformation. The following sub-sections illustrates the mapping of BPMN objects on to Petri nets.

2.2.1. Sequence flows

Figure 3 indicates the mapping of sequence flow objects to petri net places. The Left Hand Side (LHS) rule queries for a sequence flow object connecting two generic flow nodes connected via an incoming and outgoing association and mapping the sequence flow onto petri net place, as indicated on the Right Hand Side (RHS) rule. The Negative Application Condition (NAC) also makes sure this does not apply to sequence flow objects that have already been mapped.

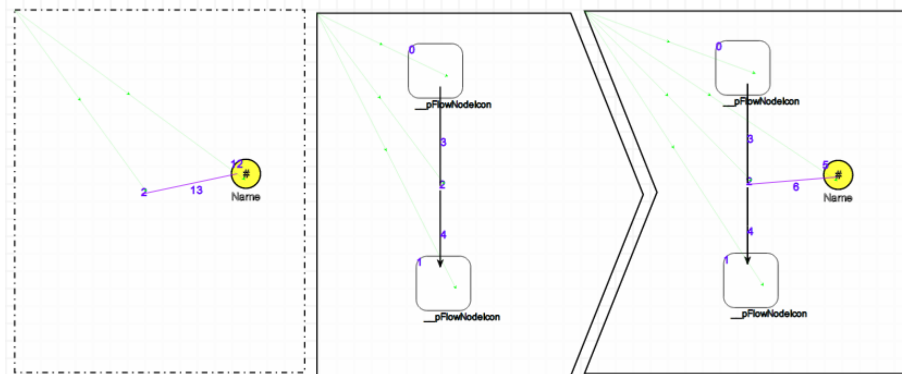


Figure 3: Sequence flow transformation rule

2.2.2. Events

Events such as start, message start and an end event simply queries for a sequence flow object connected to the event and based on the connection type, i.e. either outgoing or incoming, it simply maps the event to a place and also puts a transition in between the place mapped to a sequence flow and the place with the corresponding event as indicated on the example for a start event on figure 4. Moreover, intermediate events such as timer and message events are mapped onto transitions as indicated on figure 5.

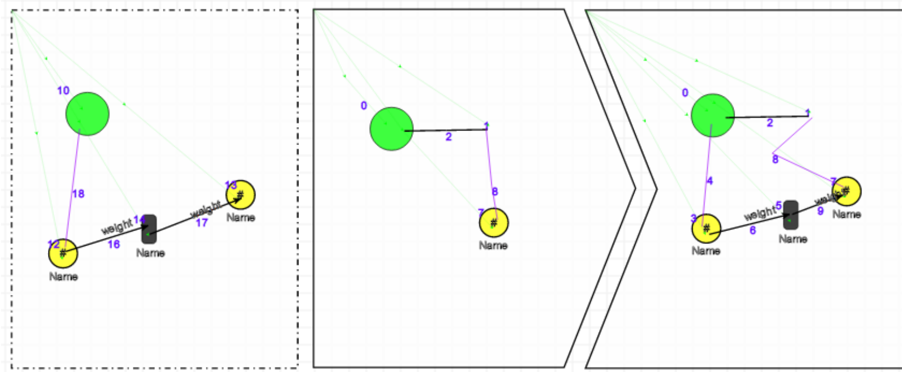


Figure 4: Start event transformation rule

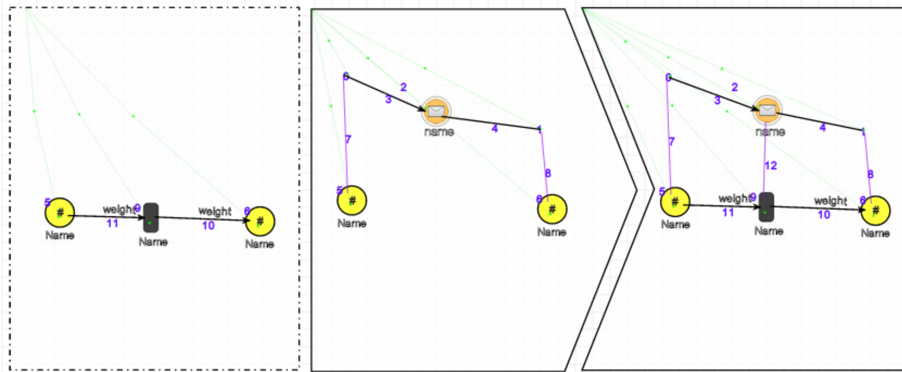
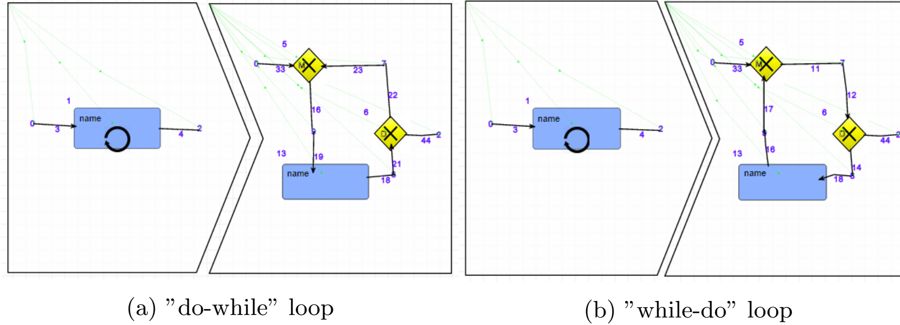


Figure 5: Intermediate message event transformation rule

2.2.3. Activities

The activity node also has sub processes as a type. However, since a sub process is viewed as a stand alone process by which BPMN elements are contained, it would not add much to the control flow perspective and thus not



(a) "do-while" loop

(b) "while-do" loop

Figure 6: Two variants of sequential activity Looping

included in a subset of the notations. A task is on one hand an atomic activity and hence can be mapped onto petri net transitions similar to intermediate events discussed earlier. Also to note is the graph link connection between some intermediate flow nodes and transitions to imply these objects can use message flows for communication and thus will be used later during the construction of message flow mapping rules.

An activity may also have attributes that specify special behaviour such as repetition (i.e. the activity is executed multiple times sequentially) and multiple instantiation (i.e. the activity is executed multiple times concurrently)[1][3]. There are two variants of sequential activity repetition: one corresponding to a "while" loop and the other corresponding to a "repeat-until" loop as depicted on figure 6. Note that the value of attribute "TestTime" determines whether the repeated activity corresponds to a "while" loop or a "repeat-until" loop.
Activities with a "multiple instantiation", are executed in multiple instances (i.e. copies) with each of these instances running concurrently and independently of the others. The number of instances n may be determined at design time or at run time. If n is known at design time, the "multiple instantiation" construct can be replaced by n identical copies of the activity enclosed between an AND-fork and an AND-join. However, figure 7 shows a mapping that is equivalent to $n = 2$, since this project is undergone for demonstration purpose.

2.2.4. Gateways

There are four gateways as part of the subset BPMN formalism built. For parallel gateway (AND),the transformation rule checks for an AND gateway with one incoming and two outgoing sequence flows. If the conditions are met, the fork object is mapped to a transition.This way a token in a place mapped to incoming sequence flow can be distributed to two places mapped to outgoing sequence flows when the corresponding parallel fork transition fires, showing concurrency.
In a similar fashion for parallel joins, the transformation rule checks for an AND gateway with two incoming and one outgoing sequence flows. If the conditions are met, the join object is mapped to a transition.This way the two tokens in the

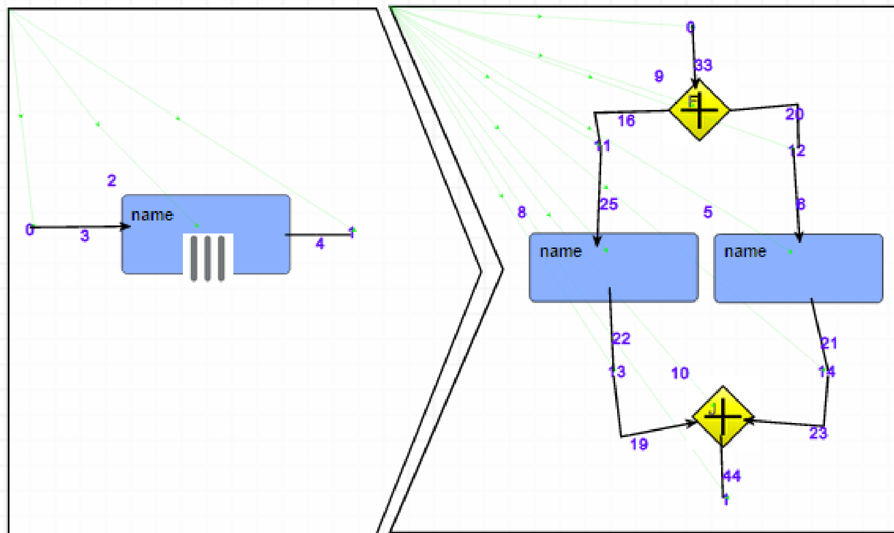


Figure 7: Mapping Activity with dual Instantiation

places mapped to incoming sequence flows can be joined to one place mapped to outgoing sequence flow when the corresponding parallel join transition fires, showing synchronization.
Moreover, for exclusive or gateway (XOR), the transformation rule again queries for an XOR gateway with one incoming and two outgoing sequence flows. If conditions are met, the transformation will result in mapping the XOR decision gateway to two transitions.These transformations will compete for a single token found in a place mapped to the incoming sequence flow, and the choice as to which one will fire will be non-deterministic.

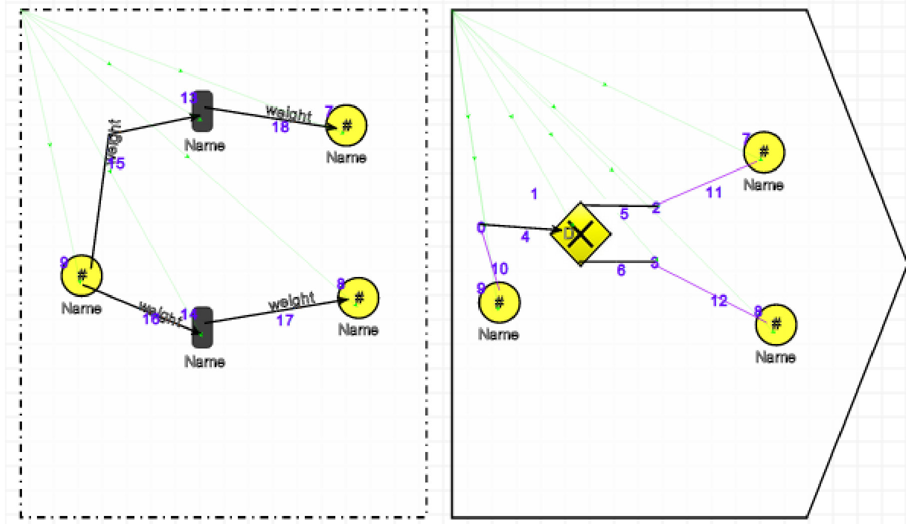


Figure 8: Search for XOR decision gateway

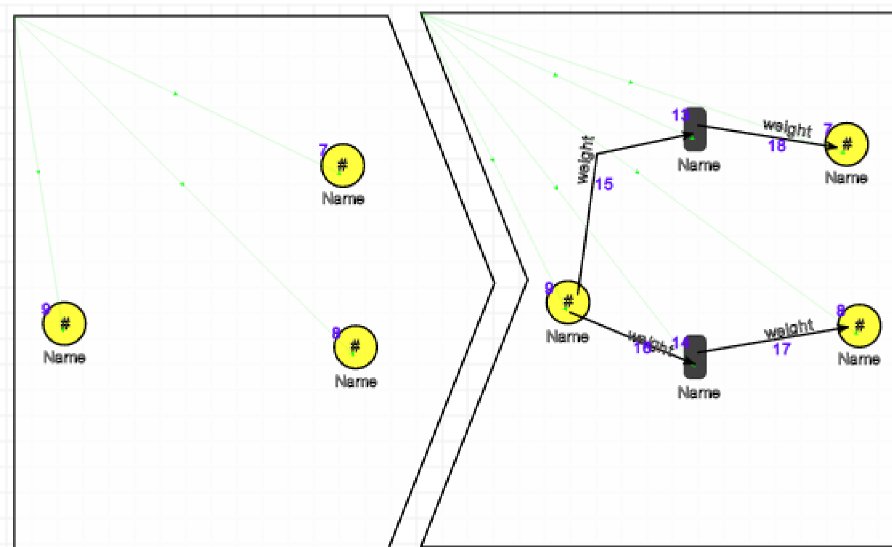


Figure 9: XOR decision gateway transformation rule

Likewise, for an XOR merge gateway with two incoming places mapped to sequence flows, the transformation rule will map the merge gateway to two transitions in which case either one of them is sure to be enabled as a consequence of the decision gateway mapping.
The XOR decision gateway transformation rule can be seen in figure 8 and figure 9. Also to note in these transformations is that the mapping is performed only for two incoming and outgoing sequence flows while for example a parallel fork can have multiple outgoing sequence flow. This is to simplify the mapping process as it has no effect in it's generality.

2.2.5. Exception Handling

In BPMN, exception handling is captured by exception flows [1]. An exception flow originates from an error event attached to the boundary of an activity. For presentation purposes, it is convenient to distinguish the case where the activity is a single task, from the case where it is a sub-process. Figure 10 shows the mapping of an error event associated with a task. Given that the execution

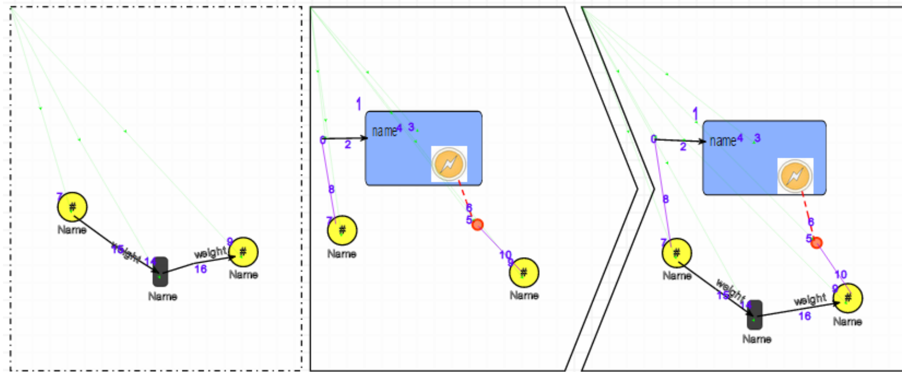


Figure 10: Exception handling transformation rule

of task **T** is atomic, the occurrence of exception **Ex** may only interrupt **T** when **T** is enabled and has not yet completed. In Petri net terms, this means that the occurrence of exception **Ex** can steal the input token that would normally be consumed by the transition corresponding to task **T**.

2.2.6. Message Flow

A message flow describes the interaction between processes contained in pools[1]. It can be mapped to a place with an incoming arc from the transition modelling a send action and an outgoing arc to the transition modelling a receive action. A special case is the mapping of a message flow to a start event where the process is instantiated each time a message is received. In this case, the message flow is directly mapped to an arc linking the transition that models sending the message to the place that signals triggering the start event.

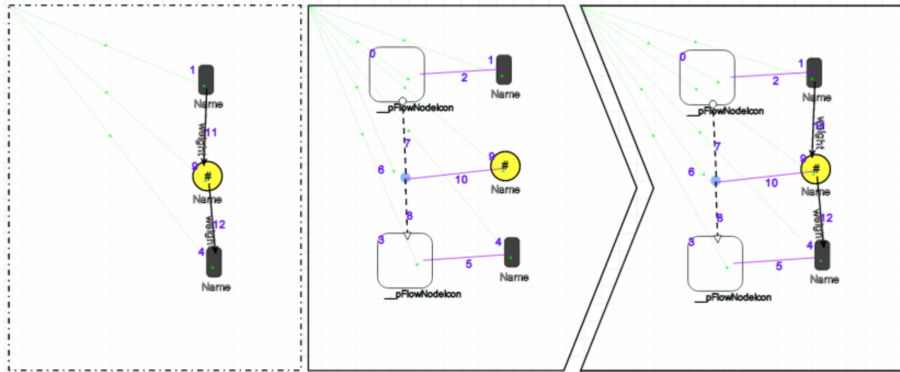


Figure 11: Generic message flow transformation rule

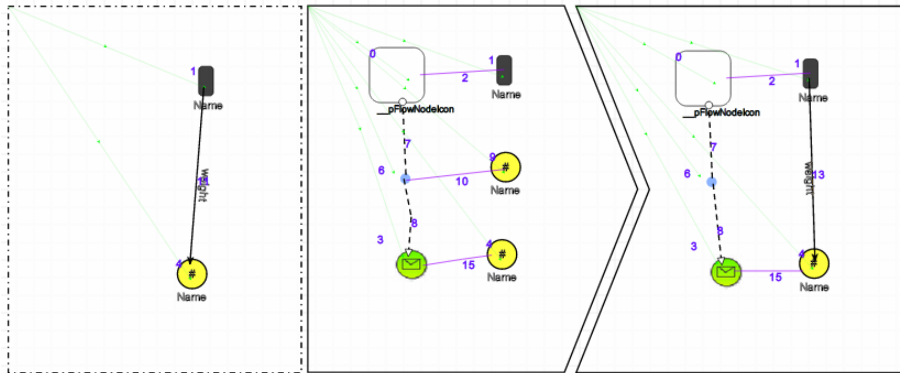


Figure 12: A start event associated to a message flow

Figure 11 shows the case where a message is not associated to a start event and figure 12 illustrates the case where it is. Also to note is that when using the generic flow node, the transformation rule only queries for nodes with an actual graph link to transitions, thus, indicating that these nodes are the only ones that can participate in messaging.

2.2.7. Initial Marking configuration

The initial state of a BPMN model can be specified by the initial marking of the corresponding Petri net model. The basic idea for configuring the initial marking is to mark the trigger places for each of the start events that do not have any incoming message flows and that the processes they belong to are top-level processes[3]. However, in this experiment sub processes are not part of a subset of the BPMN built which result in having hierarchical processes. A message flow that has as a target the start event of a process, will create an instance

of the process upon message delivery. So, the mapping should ensure that the trigger place of each start event with an incoming message flow does not contain a token in the initial marking, because the process can only be instantiated as a consequence of this event when a message has arrived. We can clearly see in figure 13 the LHS rule queries for a start event while the NAC imposes a rule that the start event does not have an incoming message attached to it.

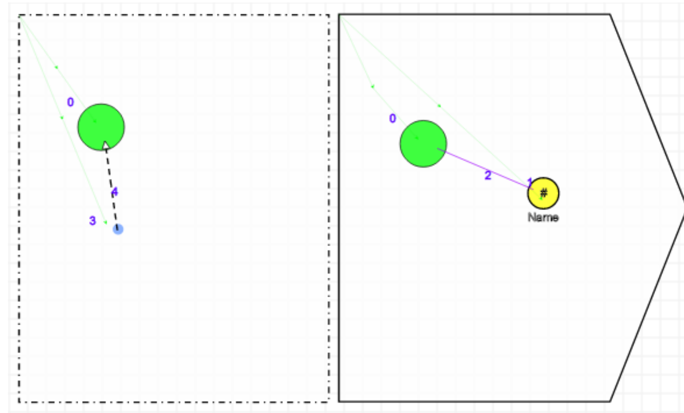


Figure 13: Search for start event with no incoming message flow

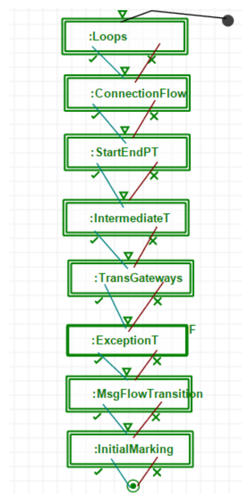


Figure 14: Schedule for BPMN to PN transformation

2.2.8. Scheduling transformation rules

Figure 14 shows the schedule for executing the transformation rules discussed so far. We can see there are six composite rules each with a sub transformation to execute. The order is defined to allow for some transformation rules to be operated prior to others. For example, activities with loops and multiple instantiation takes the most precedence because the transformation rules map these objects to other semantically equivalent notations which will be used in the following transformation rules for further mapping them onto petri nets.

3. BPMN Model Analysis

Petri nets are particularly suited to model behavior of systems in terms of *flow*, be it the flow of control or flow of objects or information[3]. This feature makes Petri nets a good candidate for formally defining the semantics of BPMN models, since BPMN is also flow-oriented.

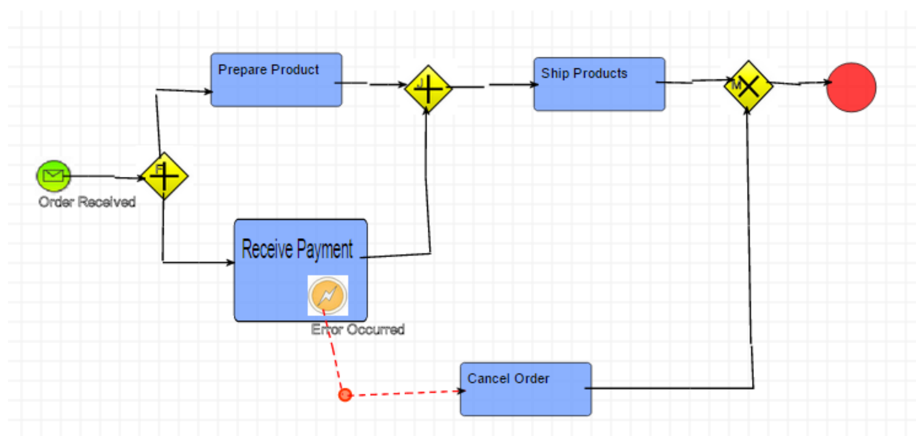


Figure 15: A BPMN order process model

This paper, hence, uses the petri net models derived from mapping BPMN to perti nets for model analysis. We have a simple order process model as depicted on figure 15 and a model with two entities showing collaboration via message flows in figure 16. The following sub-sections discuss a reachability analysis based on *"Proper completion"* and *"Absence of dead tasks"*, for the petri net models obtained from the mapping.

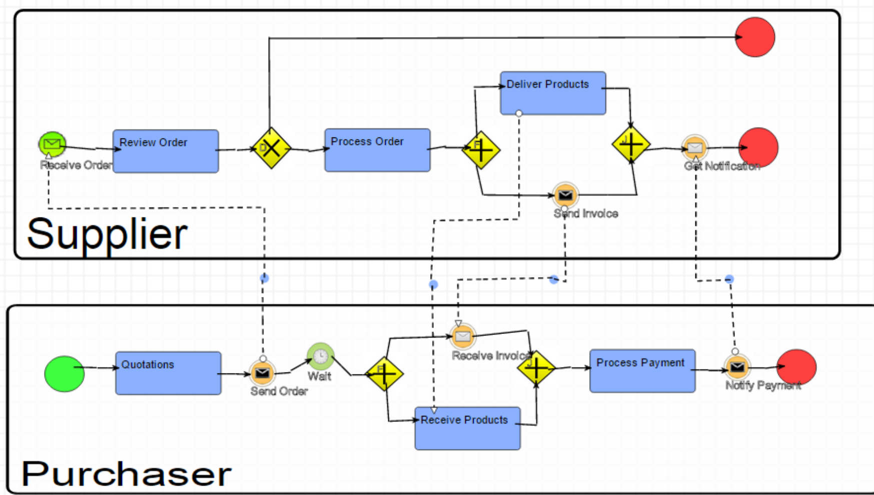


Figure 16: BPMN model with collaboration and message flows

3.0.9. Proper completion

We have the notion that an instance of a process model is completed when at least one of its end tasks has been executed at least once, and there is no other enabled task for that process instance. Hence,in Petri net terms, it implies any reachable state having a token in the sink place does not have a token in any of the other places.

Figure 17 shows a petri net model of an order process model. We can clearly

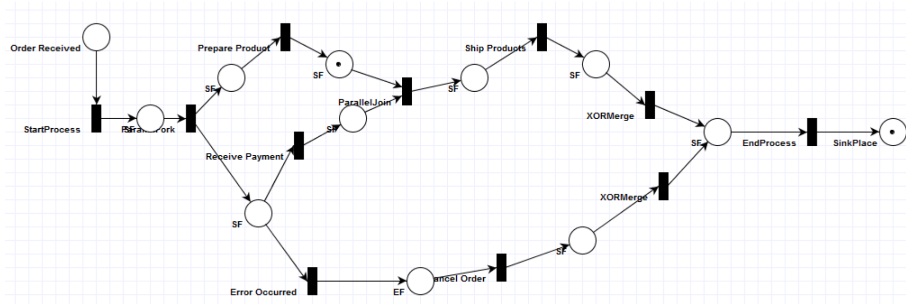


Figure 17: Order process petri net model with improper completion

see in the reachability graph, i.e. figure 18, obtained from this model that there is a reachable state S15 having an end process and prepare products as it's last enabled transitions. This means the order process may not complete properly. If receiving payment fails with an error, the process will complete but a token

is left in-between task "preparation of products" and "ship products". Practically, this means that the products are packed but not shipped because of payment issues. The process would need to be corrected to properly withdraw this remaining token and to undo any product preparations that may have been performed.

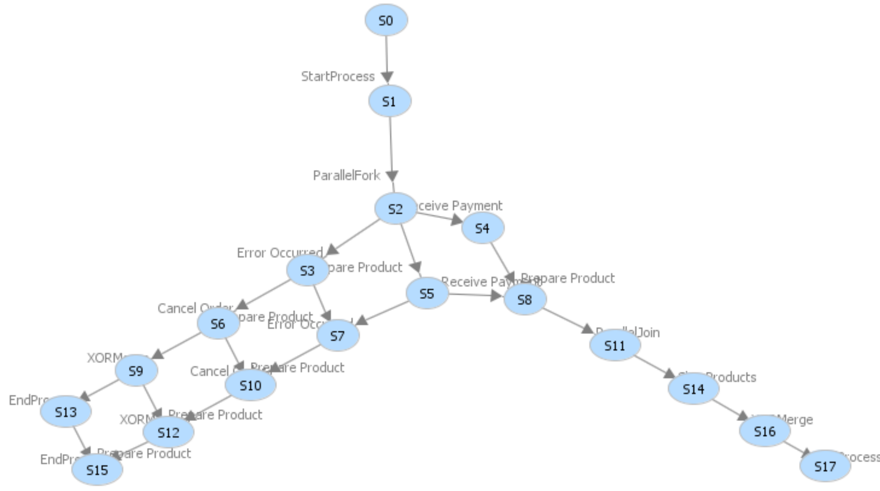


Figure 18: Reachability graph of order process model

3.0.10. Absence of Dead Tasks

In a process instance, absence of dead tasks means that there are no tasks that can never be performed within a model. It can be checked through the absence of dead transitions within the corresponding net. In other words, for any transition there is a reachable state enabling it, except for the end process transition where we get into a deadlock implying the process has completed.

In figure 19, is a corresponding petri net model for the purchaser-supplier collaboration work flow. The petri net model in the figure shows a reachable state where the purchaser process instance has a dead task that can never execute. If the supplier's process ends in declining to further process the purchaser's order request, the purchaser will wait indefinitely without further execution of tasks. Hence, the process would need to be corrected possibly with a message from the supplier notifying the rejection of the order request. This way the purchaser would require an alternative course of work flow to carry out resulting in a process that completes.

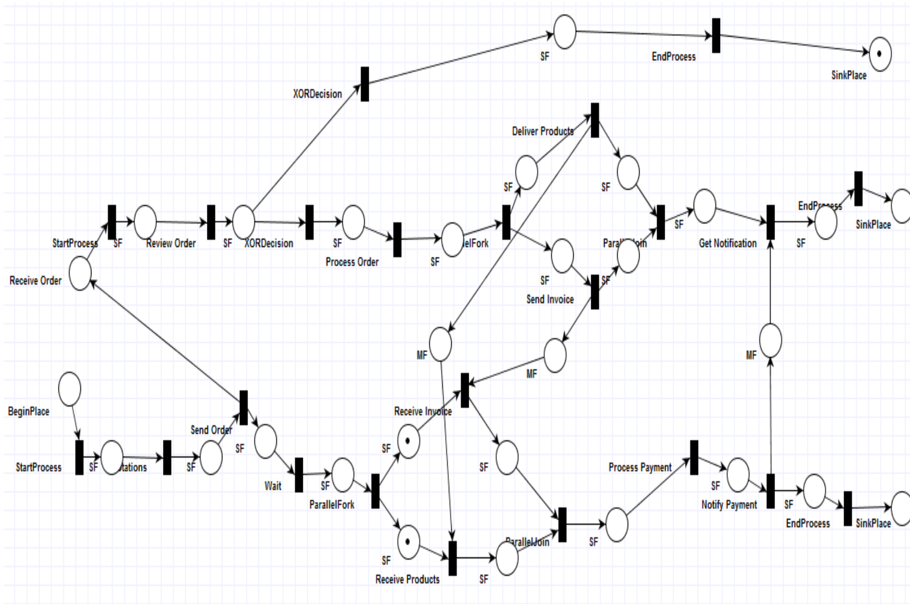


Figure 19: Petri net model of a collaboration BPMN diagram with deadlock

3.1. Conclusion

This paper has taken a model driven engineering approach to disambiguate the core constructs of BPMN by providing a denotational semantics, and also verify the semantic correctness of BPMN models. The use of AToMPM as a working tool set has greatly benefited the project to be more efficient in performing the mapping using model transformations, as opposed to additional efforts involved in tool design and implementation that supports the mapping. BPMN specification has a range of detailed syntactic constraint specifications as we have seen with, for instance, the connection flows. In addition, BPMN also provides multiple visual concrete syntax for the diverse elements it has making them more distinguishable perceptually.

However, it is worth to note that the BPMN specification lacks a systematic and unambiguous semantic definition. For instance, a BPMN process model may have multiple start events but the meaning of BPMN process models with multiple start events is not fully specified, similar to when a process is deemed complete.The BPMN specification states that each Start Event is an independent event. That is, a process instance *SHALL* be generated when the Start Event is triggered[3]. Though ambiguous, this statement suggests that it is enough for one of the start events to occur for a process instance to be generated. However, once a process instance is generated by the occurrence of a start event, it is unclear whether the other start events may, must or may not occur as part of the execution of that process instance.

Moreover, the BPMN specification does not clearly state when should an execution of a process model be considered to be completed. This is particularly problematic for process models with multiple end events since many options are possible in this case, e.g. is it enough that one end event occurs (or is reached) for the process instance to be completed, or should we wait for all end events to be reached, or should we wait until there is no activity within the process instance that is enabled or active? As it is with my experiment and other related research, we can adopt to a completion policy similar to the one discussed before.
In addition, the fact that BPMNs are highly diverse makes a designer prone to create models with errors and or ambiguity. Hence why it is quite essential to verify BPMN models before such errors potentially lead to significant system development associated costs which are also difficult to recover.

References

[1] Object Management Group, http://www.omg.org/spec/BPMN, Business Process Model and Notation (BPMN).

[2] W. P. van der Aalst, Business process management: A comprehensive survey, ISRN Software Engineering 2013 (2012) 37. doi:w.m.p.v.d.aalst@tue.nl.

[3] R. M. Dijkman, M. Dumas, C. Ouyang, Semantics and analysis of business process models in BPMN, Information and Software Technology 50 (12) (2008) 1281–1294. doi:http://eprints.qut.edu.au/7115/.

[4] E. Syriani, H. Vangheluwe, R. Mannadiar, C. Hansen, S. V. Mierlo, H. Ergin, AToMPM: A web-based modeling environment, Tech. rep., University of Alabama, University of Antwerp, McGill University.