

# PN2Maude: An automatic tool to generate Maude specification for Petri net models

Ammar Boucherit      Messaoud Abbas  
Mohammed Lamine Lamouri      Osman Hasan

## Abstract

Currently, Model-Driven Engineering (MDE) plays a key role in the software development process as it aims to handle their increasing complexity and focuses on the automatic generation of code and/or specifications from system models. This paper presents a very useful tool for the automatic generation of Maude specifications from both Petri net PNML (Petri Net Markup Language) descriptions or incidence matrices. At the end of this paper, a simple but complete Petri net example will be presented to demonstrate the usefulness of the developed tool.

**Keywords:** Maude, Rewriting Logic, Petri Nets, Code Generation.

**MSC 2020:** 68N30, 68Q60.

**ACM CCS 2020:** Grammars and Other Rewriting Systems, Formal Languages.

## 1 Introduction

Nowadays, computer systems have become more and more indispensable not only in the industrial field, telecommunication, and energy production but in almost all areas of our daily life. On the other hand, because of the increasing complexity and the involvement of several heterogeneous and distributed components interactions in these systems, they are also responsible for an ever-increasing number of errors of varying severity. Therefore, it becomes necessary that the development of software systems should be based on powerful modeling formalisms facilitating the analysis and verification of these systems before

their actual implementation. Moreover, since graphical modeling techniques are of increasing interest, formal approaches accompanied with a graphical representation are becoming more interesting.

Petri nets [1] have been widely used as a graphical and semi-formal tool for specification and analysis of concurrent and complex systems [2], [3]. They are gaining more popularity in recent years since they allow an easy structural and behavioral description of studied systems [4], [5]. Due to their popularity, a large number of tools have been developed for editing, modeling, and analyzing Petri net model's properties [6]. However, the lack of interoperability between such tools, manual preparation, and ad hoc validation of system specifications represent some of their major shortcomings.

In this context, rewriting logic is a very expressive logic that is particularly suitable for formalizing concurrent, complex, and real-time systems [7]. In addition, it is a unifying framework for a wide range of Petri nets [8] and many other concurrency models [9]. Nevertheless, specifications based on the rewriting logic of Petri net-based systems are often voluminous and/or more difficult to refine or correct. Hence, it is necessary to automate such an operation to save time and avoid errors in the process of writing (preparing) specifications.

Model-Driven Engineering (MDE) is known as a promising solution to handle the increasing complexity of software architecture through the automatic generation of code and/or specification from system models. In fact, it helps the designer to integrate formal specification and verification techniques at earlier stages in the software development life-cycle. Therefore, looking for a solution integrating MDE technologies and allowing system designers to quickly, easily, and automatically generate formal specifications of Petri nets are expected to be very advantageous. In addition, we believe that the use of standard input format like the Petri Net Markup Language (PNML) and/or incidence matrices will enlarge the usefulness of this solution and allows designers to use Petri net tools while preparing their system models.

The main objective of this work is to present a tool (PN2Maude) that offers a simple and quick way for Maude practitioners to automatically generate the specification for their Petri net models from both PNML description and incidence matrices. Once the Maude specifi-

cation of a Petri net is generated, designers can then use the Maude LTL model checker or Maude reachability tool to analyze and verify its behavioral properties. This can greatly simplify the process of modeling and verifying concurrent and distributed systems and can help to ensure their correctness and reliability. In fact, this tool is an enhancement and extension of our previous work [10], in which we have proposed an algorithm for the automatic generation of Maude specification based only on the existing rewriting logic semantics by using incidence matrices for pure Petri nets without inhibitor arcs.

The remainder of this paper is structured as follows. In Section 2, we give a brief introduction to the Petri nets descriptions and the classical and improved semantics based on the rewriting logic for Petri nets. Section 3 discusses related works and is followed by Section 4 that presents a brief description of the process of translating the PNML Petri net description into rewriting logic as well as the structure of the tool PN2Maude and its main modules. In Section 5, we present the PN2Maude tool with a simple illustrative example. Finally, Section 6 concludes the paper and draws some perspectives.

## 2 Preliminaries

### 2.1 Petri nets Descriptions

Petri nets are typically regarded as one of the widely used modeling tools for the specification and analysis of concurrent and distributed systems. A Petri net can be viewed as a directed bipartite graph, in which arcs are labeled with their corresponding weights and connect nodes from different types. While the first type of nodes represents events, and they are depicted by rectangles or bars (called, Transitions), the second ones represent conditions or objects, and they are depicted by circles (called, Places). Places may contain a discrete number of dots (called, Tokens). Figure 1 below shows a simple Petri net consisting of four places and four transitions.

The distribution of tokens over the Petri net places represents its configuration and is called a marking. A particular transition  $t$  is enabled if all its input places (places leading to that transition) contain

sufficient tokens. Thereafter, an enabled transition  $t$  may be unconditionally fired, and, therefore, it changes the Petri net marking.

Practically, one can distinguish two sets of places for a transition  $t$ :

- $\text{pre-set}(t)$ : is composed of the input places of  $t$ , also referred to as  $\bullet t$ . For instance, in our example of Petri net, the  $\text{pre-set}(T_4) = \{P_3, P_4\}$ .
- $\text{post-set}(t)$ : is composed of the output places of  $t$ , also denoted by  $t\bullet$ . Consequently, the  $\text{post-set}(T_4) = \{P_1\}$ .

Accordingly, a tuple  $(p, t)$  is called a self-loop if  $p$  belongs to both the  $\text{pre-set}(t)$  and  $\text{post-set}(t)$ . Accordingly, a Petri net that does not contain any self-loop is called pure. In addition, a transition that does not have any input place is called a source transition, and a transition that does not have any output place is called a sink transition. Moreover, a particular kind of arcs is called inhibitor arcs which is denoted by an arc with a small circle attached to a transition. Such kind of arcs is generally used for tests and does not consume tokens after firing. A transition connected by an inhibitor arc with a place will only be enabled if such a place contains fewer tokens than the weight of the inhibitor arc. Figure 1 gives an example of a Petri net.

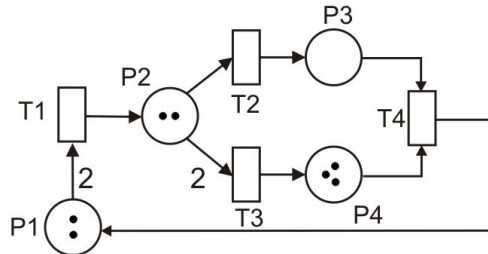


Figure 1. Example of Petri net

### 2.1.1 Matrix Representation

The easiest mathematical way to represent the structure of a Petri net with  $N$  places and  $M$  transitions is by using matrix representation. Besides the fact that matrices can provide an alternative way to describe

Petri nets from conventional graphical representation, one of the main motivations for this representation is that it facilitates mathematical analysis of the behavior and properties of Petri nets, such as deadlock and liveness. The three main types of matrices used for Petri net representation are given as follows:

- **PRE-Matrix (Input matrix):** It is a matrix (N,M) that captures all the input places to all transitions. An element of matrix (N,M) equals the weight of the arc linking a place  $p_i$  to the transition  $t_j$  if it exists and 0 otherwise. The PRE-Matrix may contain negative values in some particular cases, such as the Petri net with inhibitor arcs, to reflect the weights of the inhibitor arcs and to distinguish them from ordinary arcs.
- **POST-Matrix (Output matrix):** It is a matrix (N,M) that captures all the output places to all transitions. An element of matrix (N,M) equals the weight of the arc linking the transition  $t_j$  to a place  $p_i$  if it exists and 0 otherwise.
- **INC-Matrix (Incidence matrix):** It basically represents POST-Matrix - PRE-Matrix. It can only be used for pure Petri nets. Otherwise, the static structure of a non-pure Petri net will not be properly described, and, in that case, POST-Matrix and PRE-Matrix have to be used instead.

For example, the Petri net in Figure 1 can be specified in matrix representation as follows:

$$\begin{matrix} PRE \\ \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{matrix} \quad \begin{matrix} POST \\ \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \end{matrix} \quad \begin{matrix} INC \\ \begin{bmatrix} -2 & 0 & 0 & 1 \\ 1 & -1 & -2 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & -1 \end{bmatrix} \end{matrix} \quad (1)$$

In these matrices, each row represents a place, and each column represents an input or output of a transition. The values in the matrix represent the number of tokens that are either consumed or produced by transitions, i.e., a positive value in the matrix represents tokens being produced, while a negative value represents tokens being consumed.

### 2.1.2 PNML Description

The Petri Net Markup Language (PNML) is a standardized XML-based description of Petri net models. Indeed, a PNML description (see Figure 2) is generally made up of a set of main nodes such as “Place”, “Transition” and “Arcs” to describe the structure of a Petri net in the form of a labeled directed graph. In addition, PNML provides a universal interchange file format between various Petri net tools and a common language that allows users and developers to interchange Petri nets, which enables them to reuse models and integrate tools in the analysis of complex systems. Practically, there are many tools, such as WoPeD [11], P3 [12], and PIPE [13], that are capable of exporting the Petri net model in the PNML format.

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<pnml>
  <net id="Net-One" type="P/T net">
    <place id="P0">
      <name> <value>P0</value> </name>
      <initialMarking> <value>Default,0</value> </initialMarking>
    </place>
    .....
    <transition id="T0">
      <name> <value>T0</value> </name>
      <timed> <value> false </value> </timed>
    </transition>
    .....
    <arc id="P0 to T1" source="P0" target="T1">
      <inscription> <value>Default,1</value> </inscription>
      <type value="normal"/>
    </arc>
    .....
  </net>
</pnml>

```

Figure 2. Example of the structure of a PNML description

As can be seen, the Petri net components are specified in detail with three nodes. For instance, the three attributes `id` (identifier), `name`, and `initial marking` are used to define a place (see node `<place id = ... > ... </place>`). A transition is also described with three attributes, `id` (identifier), `name`, and `timed`, a boolean attribute that determines whether the transition is timed or not (see

node `<transition id = ... > ... </transition>`). Lastly, the attributes `id`, `inscription`, and `type` are used to describe an arc (see node `<arc id = ... > ... </arc>`). The source and target of the arc are identified by the `id` attribute. The weight of the arc is written in the inscription. The type indicates whether the arc is normal or inhibitor.

## 2.2 Petri nets and Rewriting Logic

Since 1990, rewriting logic [14] has appeared as a promising logical and semantic framework within which Petri nets [8] and many other different concurrent systems and logics can be naturally specified [15]. Maude system is an implementation of rewriting logic that offers a powerful verification toolkit so that Maude's reachability analysis and model-checking can be used to formally analyze and verify the Petri net models with respect to different LTL properties [16].

Usually, a Maude specification is composed of a functional module and/or a system module. While the functional module describes the static part of a system by an equational theory, the dynamic part is described by a set of rewrite rules and, possibly, equations in a system module.

In the first proposed rewriting logic-based semantics for Petri nets [8], there are two main types (`Sorts`, `Place` and `Marking`). The `Marking` is a multiset of `Place` as it represents the distributions of tokens over the Petri net. In addition, the names of places are used to represent token instances. Moreover, each Petri net transition is expressed with a rewrite rule as follows:  $[t] \Rightarrow [t']$ , where:

- $[t]$  and  $[t']$ : are terms that represent a part of the global marking of the Petri net and describing the input and output of a transition, respectively.
- $\Rightarrow$ : describes the change to be made while firing a transition ( $[t]$  and  $[t']$  are the parts to be consumed and produced, respectively).

Therefore, the Maude specification of Petri net in Figure 1 is given in Listing 1.

```

fmod FUNC_PETRINET is
sorts Place Marking .
subsorts Place < Marking .
op empty : -> Marking .
ops P1 P2 P3 P4 : -> Place .
op _ _ : Marking Marking -> Marking
    [assoc comm id: empty] .
endfm

mod SYSTEM_PETRINET is
including FUNC_PETRINET .
rl[T1]: P1 P1 => P2 .
rl[T2]: P2 => P3 .
rl[T3]: P2 P2 => P4 .
rl[T4]: P3 P4 => P1 .
endm
    
```

Listing 1. First Maude specification

In this context, it is worth noting that there is another enhanced rewriting logic-based semantics [17],[18] for Petri nets, and, therefore, the corresponding specification of our Petri net is given in Listing 2.

```

fmod NEW_FUNC_PETRINET is
protecting INT .
sorts PlaceName Place Marking .
subsorts Place < Marking .
op empty : -> Marking .
ops P1 P2 P3 P4 : -> PlaceName .
op <_,_> : PlaceName Int -> Place [ctor] .
op _ _ : Marking Marking -> Marking [ctor assoc comm id: empty] .
endfm

mod NEW_SYSTEM_PETRINET is
inc NEW_FUNC_PETRINET .
vars x1 x2 x3 x4 : Int .
crl[T1]: < P1,x1 > < P2,x2 > => < P1,x1 - 2 > < P2,x2 + 1 > if
    ( x1 >= 2 ) .
crl[T2]: < P2,x1 > < P3,x2 > => < P2,x1 - 1 > < P3,x2 + 1 > if
    ( x1 >= 1 ) .
crl[T3]: < P2,x1 > < P4,x2 > => < P2,x1 - 2 > < P4,x2 + 1 > if
    ( x1 >= 2 ) .
crl[T4]: < P1,x1 > < P3,x2 > < P4,x3 > => < P1,x1 + 1 > < P3,x2 -
    1 > < P4,x3 - 1 > if ( x2 >= 1 ) and ( x3 >= 1 ) .
endm
    
```

Listing 2. Second Maude specification

As can be seen, our Petri net has four places, P1, P2, P3, P4 and four transitions, T1, T2, T3, and T4. In addition, in Listing 1 (respectively, Listing 2), there are two modules.

While the first is a functional module that describes the static part of the Petri net, the second is a system module that describes the dynamic part of the Petri net.



However, the new semantics differs from the existing one in that it encodes the sets of tokens in a place by their cardinality, which allows for the natural expression of various high-level Petri nets such as Petri nets with inhibitor arcs, variable arc weights, colored Petri nets, etc. Additionally, the new semantics facilitates the expression and then checking of behavioral properties such as boundedness, Deadlock, and conservation by using the Maude LTL model checker.

## 2.3 XSLT Transformation

XSLT is a technology that transforms information from an XML document to another type of document, such as XML, HTML, XHTML, WML, PDF, etc. The process of transformation is mainly based on an XSLT document, called stylesheet, which is an XML format file that contains the information needed by the processor to perform the transformation. Practically, an XSLT Stylesheet is associated with an XML-based document in order to create a resulting document of a different or identical format. This principle is illustrated in Figure 3.

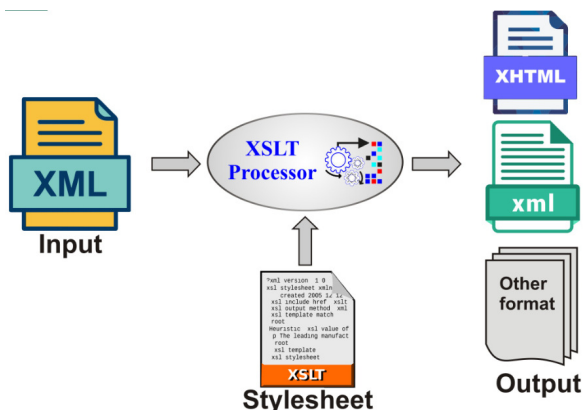


Figure 3. XSLT transformation principle

In the present work, XSLT functions and operators are used to navigate, select nodes from an PNML document, and then generate a purified file (see Figure 5(b)) that contains five blocks describing the set of Petri net places, transitions, and arcs. The purified file is thereafter

used for the generation of Petri net matrix description.

### 3 Related Works

Petri nets have been widely used for modeling and simulation of distributed and concurrent systems, including communication protocols and synchronization between system components. Unfortunately, most Petri net tools are rarely accompanied by model-checking algorithms. Therefore, Petri net models are often translated into other specific languages for formal analysis purposes [19], [20]. For instance, in [21], authors presented a transcription tool from Petri net to PLC programming languages. The tool makes the translation process more efficient and less error-prone and allows for greater flexibility in system design. Besides the translation process, the program supports stepwise simulation of the Petri net model, allowing for error-checking during the development. According to the authors, the proposed tool can be useful for dynamic integration projects by providing a more efficient and reliable process of design and implementation. In addition, in [22], PetriNet2NuSMV, a tool for the automatic translation of reachability graphs for colored Petri nets and place-transition Petri nets into the NuSMV language, was introduced by the authors. The reachability graphs used as input for this tool are those generated by the TINA and CPN Tools software. This tool allows the formal verification of Petri nets designed with these environments using model-checking techniques for LTL and CTL temporal logics.

In this context, Maude is a very powerful system for which some works have been realized for the transformation of Petri nets [8], [23]. However, very few works have been reported on the automatic translation of Petri nets into Maude. For instance, a tool for the editing, simulation, and analysis of a special extension of Petri nets, so-called ECATNets (Extended Concurrent Algebraic Terms Nets), is presented in [24]. Similarly, a graphic tool allowing a bidirectional translation of colored Petri nets to Maude and vice versa is reported in [25]. Then, a graph transformation-based approach is proposed in [26] for the automatic generation of ECATNets specification in Maude for simulation and analysis purposes.

Moreover, to the best of our knowledge, all the existing works are related to some extension of Petri nets (ECATNets and Colored Petri nets) and their authors did not give any details about the generation algorithm for their approaches, and none of them had made the presented tool available for users, which made the use of such tools too limited.

In addition, the most closely related work is that presented in [10], where authors have proposed an algorithm for the automatic generation of Maude specification based on the existing semantics of Petri nets. However, this work is limited to pure Petri nets without supporting inhibitor arcs.

Finally, the main advantage of the present work over the latter one is that now we generate Maude specification based on both existing and improved semantics to provide a large and solid theoretical basis and thus facilitate the simulation and formal analysis of Petri nets with inhibitor arcs using the Maude system analysis tools.

## 4 The Translation Approach

This section provides an overview about the proposed approach for translating PNML description into the Maude specification. The tool PN2Maude, developed based on this approach is accessible for users <sup>1</sup>.

### 4.1 Overview of the Process

The following Figure 4 shows a general description of the developed tool.

The main idea behind the PN2Maude tool is to generate Maude specification in six steps as follows:

1. Create a Petri net model and export it as a PNML file, or use incidence matrices and pass it to Step 5.
2. Translate the importation of the PNML file. . The current version of PNML2Maude supports Petri nets with inhibitor arcs and

---

<sup>1</sup>PN2Maude is available at: [https://drive.google.com/file/d/1I2DeysLPZzK5i-0sWtFd1PCuShc\\_0DqV](https://drive.google.com/file/d/1I2DeysLPZzK5i-0sWtFd1PCuShc_0DqV)

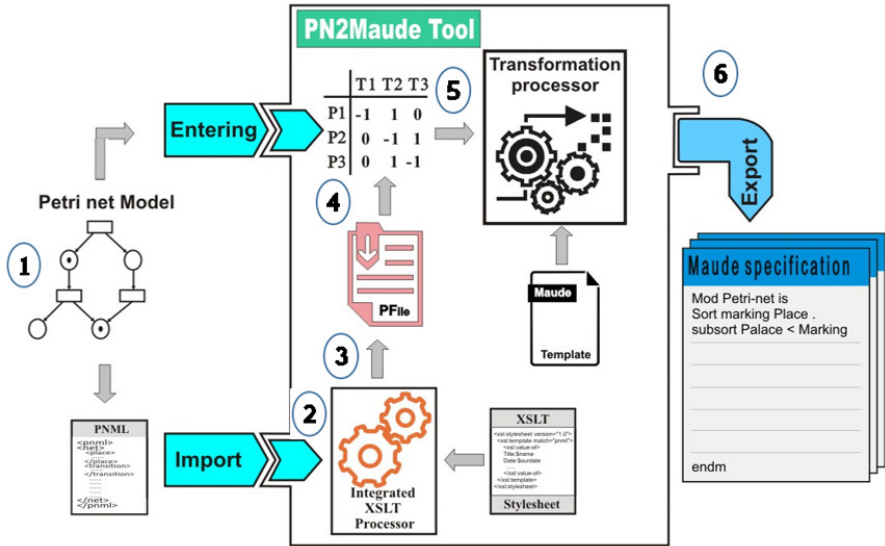


Figure 4. General Description of PN2Maude Tool  
is exported by PIPE or P3 tools.

3. Extract data and values from the PNML file and create a purified file (PF) using XSLT.
4. Create incidence matrices from PF.
5. Start translation by using incidence matrices and Maude specification templates.
6. Export the generated specification into the Maude file.

## 4.2 Details of the transformation Process

There are two primary steps in the process of generating a Maude specification from a Petri net description. The first one deals with creating the incidence matrix from a PNML file, while the second one focuses on creating a Maude specification for a Petri net using the incidence matrices. The subsections that follow provide more details on these two steps.

#### 4.2.1 From the PNML file to the incidence matrices

As we have previously noted, a PNML file typically contains information about the places, transitions, arcs, and other elements of the Petri net, which can be used to construct the incidence matrix. Therefore, the process of obtaining matrices from a PNML file (exported from the P3 or PIPE tools) is based on the use of XSLT language and goes through the following steps:

1. Purification: It allows extracting more or less essential information, such as Petri net places, transitions, arcs as well as the initial marking in order to facilitate the operation of creating the incidence matrices. Figure 5(a)) shows the XSLT code developed to navigate and select the essential nodes describing the Petri net from the input PNML file. Subsequently and in order to facilitate the creation of PRE and POST matrices, these nodes (see Figure 5(b))) are organized into five blocks describing lists of places with their initial markings, transitions, input arcs, inhibiting arcs, and out arcs, respectively.
2. Creation of matrices: This operation is based on the previous operation, where the purified file is manipulated to create and fill in the incidence matrices. This step can be summarized as follows:
  - i. The number of lines (N) of Block 1 (list of places) represents the number of lines for the incidence matrices, and the number of lines (M) of Block 2 (list of transitions) represents the number of columns for the incidence matrices.
  - ii. The elements of the incidence matrices are filled from Blocks 3, 4, and 5.

```

<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="text" encoding="utf-8" />
<xsl:strip-space elements="*" /> <xsl:template match="pnml/net">
<xsl:for-each select="place"> <xsl:value-of select="@id"/>|<xsl:value-
of select="name"/>|<xsl:value-of select="translate(initialMarking,
translate(initialMarking,'0123456789',''),'')"/> <xsl:text>|&#10;
</xsl:text>
</xsl:for-each>
<xsl:for-each select="transition"> <xsl:value-of select="@id"/>|<xsl:
value-of select="name"/> <xsl:text>|&#10;</xsl:text> <xsl:for-each>
<xsl:for-each select="arc[starts-with(@target,'t') or starts with
(@target,'T')]>
<xsl:if test="type/@value = 'normal'">inp_<xsl:value-of select=
"@target"/>:<xsl:value-of select="@source"/>(<xsl:value-of
select="translate(inscription, translate(inscription,
'0123456789', ''), '')"/>&#10;</xsl:text>
</xsl:if>
</xsl:for-each>
<xsl:for-each select="arc[starts-with(@target,'t') or starts-with(@
target,'T')]>
<xsl:if test="type/@value='inhibitor'">h_inp_<xsl:value-of select=
"@target"/>:<xsl:value-of select="@source"/>
<xsl:choose>
<xsl:when test="inscription = ''">
<xsl:text>|</xsl:text><xsl:text>&#10;</xsl:text>
</xsl:when>
<xsl:otherwise>
<xsl:text>(</xsl:text><xsl:value-of select="translate
(inscription,translate(inscription,'0123456789', ''), '')"/>
<xsl:text>&#10;</xsl:text>
</xsl:otherwise>
</xsl:choose>
</xsl:if>
</xsl:for-each>
<xsl:for-each select="arc[starts-with(@source,'t') or starts with
(@source,'T')]> out_<xsl:value-of select="@source"/>:<xsl:value-of
select="@target"/>(<xsl:value-of select="translate(inscription,
translate(inscription,'0123456789', ''), '')"/> <xsl:text>&#10;
</xsl:text>
</xsl:for-each>
</xsl:template>
</xsl:stylesheet>
    
```

(a) : XSLT File

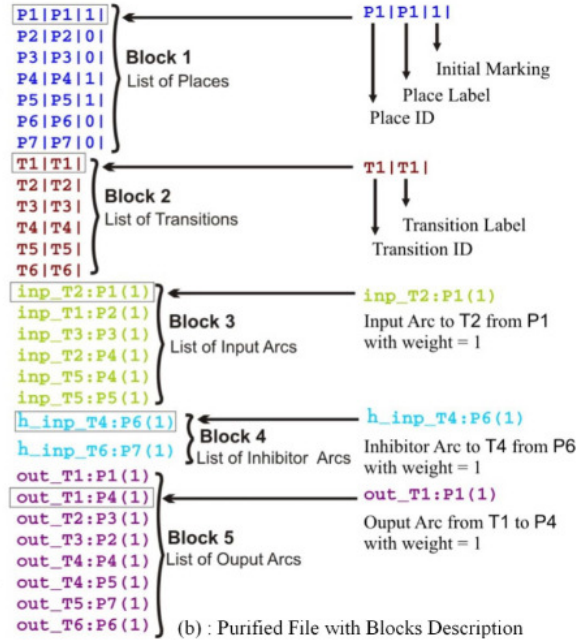


Figure 5. XSLT Code and Purified File with Blocs Description

#### 4.2.2 From incidence matrices to Maude Specification

The principle of generating Maude specifications from incidence matrices can be summarized in the following steps:

- i. The number of rewrite rules is equal to the number of transitions of the Petri net (number of columns in the incidence matrices).
- ii. A rewrite rule is made to describe the firing of a transition. This rule may often be conditional to describe the enabling condition for such a transition, especially in the enhanced semantics. Therefore, such a rule is given as follows:

```
cr1[Label] : <Left_hand_side> => <Right_hand_side> if Cond .
```

- **Label**: a given name to represent the transition.
  - **Left-Hand-Side**: the left part of the rule (LHS), which describes the marking of the Petri net before firing.
  - **Right-Hand-Side**: the right part of the rule (RHS), which describes the marking of the Petri net after firing.
  - **Cond**: a logical expression represents the enabling condition of the transition and is picked up from the PRE-Matrix.
- iii. Each transition is represented by one column in both PRE-Matrix and POST-Matrix. Therefore, some special cases can be determined as follows:
    - **Source Transition**: It is a transition that has columns with null values in the PRE-Matrix.
    - **Sink Transition**: It is a transition that has columns with null values in the POST-Matrix.
    - **Transition with inhibitor arcs**: This transition must have negative values in the PRE-Matrix.
    - **Normal Transition**: It is a transition that is not source, sink, or with inhibitor arcs.

We recapitulate the Maude specification — by using the existing and improved semantics — of the different possible cases of a Petri net transition in Table 1.

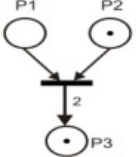
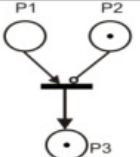
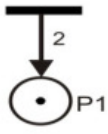
Type of Transition		Rewrite Rule	
		Existing Semantics	Improved Semantics
1	<b>Ordinary Transition</b> 	LHS: P1 P2 RHS: P3	LHS: $\langle P1, X1 \rangle \langle P2, X2 \rangle \langle P3, X3 \rangle$ RHS: $\langle P1, X1 - 1 \rangle \langle P2, X2 - 1 \rangle \langle P3, X3 + 2 \rangle$
			<b>Condition</b> if $(X1 \geq 1)$ and $(X2 \geq 1)$ .
2	<b>Ordinary Transition with Inhibitor Arc</b> 	it is not expressible without additional operators	LHS: $\langle P1, X1 \rangle \langle P2, X2 \rangle \langle P3, X3 \rangle$ RHS: $\langle P1, X1 - 1 \rangle \langle P2, X2 \rangle \langle P3, X3 + 1 \rangle$
			<b>Condition</b> if $(X1 \geq 1)$ and $(X2 < 1)$ .
3	<b>Source Transition</b> 	LHS: M RHS: M P1 P1	LHS: $\langle P1, X1 \rangle$ RHS: $\langle P1, X1 + 2 \rangle$
			<b>Condition</b> this is an unconditional rule

Table 1. Maude specification of different Petri net Transitions Cases (Continue)



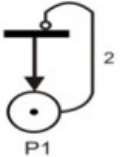
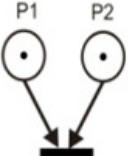
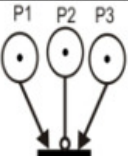
Type of Transition		Rewrite Rule	
		Existing Semantics	Improved Semantics
4	<b>Source Transition with Inhibitor Arc</b> 	it is not expressible without additional operators	LHS: $\langle P1, X1 \rangle$ RHS: $\langle P1, X1 + 1 \rangle$
			<b>Condition</b> if $(X1 < 2)$ .
5	<b>Sink Transition</b> 	LHS: $M \ P1 \ P2$ RHS: $M$	LHS: $\langle P1, X1 \rangle \langle P2, X2 \rangle$ RHS: $\langle P1, X1 - 1 \rangle \langle P2, X2 - 1 \rangle$
			<b>Condition</b> if $(X1 \geq 1)$ and $(X2 \geq 1)$ .
6	<b>Sink Transition with Inhibitor Arc</b> 	it is not expressible without additional operators	LHS: $\langle P1, X1 \rangle \langle P2, X2 \rangle \langle P3, X3 \rangle$ RHS: $\langle P1, X1 - 1 \rangle \langle P2, X2 \rangle \langle P3, X3 - 1 \rangle$
			<b>Condition</b> if $(X1 \geq 1)$ and $(X2 < 1)$ and $(X3 \geq 1)$ .

Table 1. Maude specification of different Petri net Transitions Cases

### 4.3 Generation Algorithm

As stated previously, the process of generating a Maude specification of a Petri net is based on its mathematical description or PNML. The general algorithm illustrating the generation process is given as follows in the Listing 3.

---

#### Algorithm 1 Automatic Generation of Maude Specifications (Existing and Improved semantics)

---

**Inputs:** XML File : Petri net PNML description  
 PRE(N,M) : Input incidence Matrix  
 POST(N,M) : Output incidence Matrix  
 Maude\_Spec : Maude specification Templates  
 XSLT\_Code : XSLT Stylesheet

**Outputs:** Maude Specification 1 : Existing semantics specification  
 Maude Specification 2 : Improved semantics specification

**Uses:** MSXSL : Microsoft Command Line XSL Transformation

**Begin**

- 1: **if** (User\_Choice = Import\_PNML\_File) **then**
- 2:     SELECT\_AND\_IMPORT\_PNML\_FILE
- 3:     CREATE\_PF\_IMPORT\_PNML\_FILE                     ▷ PF : is the purified file
- 4:     AUTOMATIC\_FILL\_MATRICES (PRE(N,M),POST(N,M))     ▷ filling  
       PRE-Matrix and POST-Matrix
- 5: **else**
- 6:     FILL\_PRE\_MATRIX(PRE(N,M))
- 7:     FILL\_POST\_MATRIX(POST(N,M))
- 8: **end if**
- 9: **NB\_Rules** ← M ▷ M : number of columns of incidence matrices = number of  
    transitions
- 10: **for** (each semantics\_template) **do**     ▷ templates for existing and improved  
    semantics
- 11:     **for** (j ← 1 to NB\_Rules) **do**
- 12:         TType ← Get\_Transition\_Type(j, PRE\_Matrix, POST\_Matrix) ▷  
        source, sink, inhibitor or normal transition
- 13:         RewriteRule ← Generate\_Rule(j, TType)
- 14:         Insert\_Rule(semantics\_template, RewriteRule)
- 15:     **end for**
- 16: **end for**

**End**

---

Listing 3. First Maude specification

## 5 Presentation of the PN2Maude tool

PN2Maude is developed using the Delphi 7 language, which is a strongly typed language that supports the structured and object-oriented design, allowing rapid application development (RAD) running on Windows. In practice, the declarative programming language XSLT has been used to purify a PNML file and extract only the information necessary for the transformation and save it in the file *purify* (PF). This file is then used to create the incidence matrices, which will be used in the process of generating the Maude specification. The latter can be saved in a Maude file. On the one hand, we have used MSXSL, which is Microsoft's free command-line XSLT processor for performing XSL transformations. In the next subsections, we will present the PN2Maude interface windows with a case study for the following Petri net (see Figure 6).

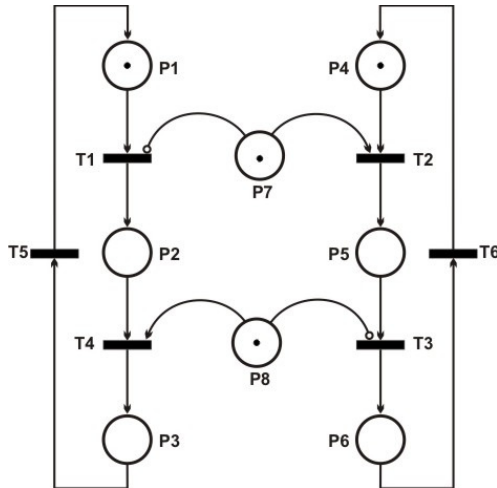


Figure 6. Example of Petri net (Case study)

### 5.1 Main Interface Window

The main interface window of PN2Maude tool is given in the following Figure 7).

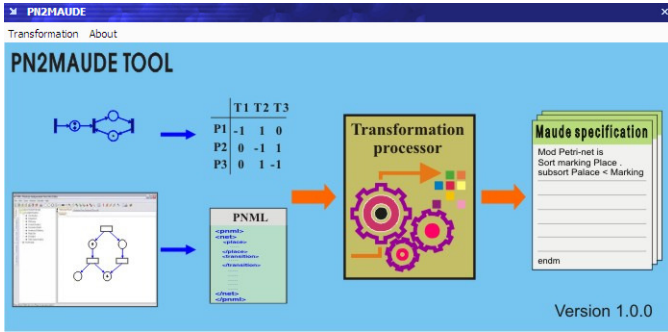


Figure 7. Main Interface Window of PN2Maude Tool

## 5.2 Using Incidence Matrices choice Interface Window

To introduce the Petri net, one can use the choice of “using incidence matrices” as shown in Figure 8.

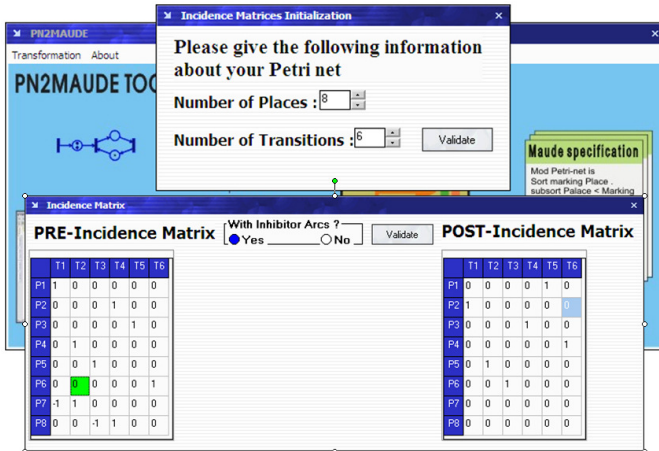


Figure 8. PN2Maude Interface Window for Using Incidence Matrices

## 5.3 Using PNML File choice Interface Window

The same Petri net may be introduced via a PNML file if it is edited by a Petri net tool, such as PIPE or P3. Therefore, the interface window

for importing a PNML file is given in Figure 9).

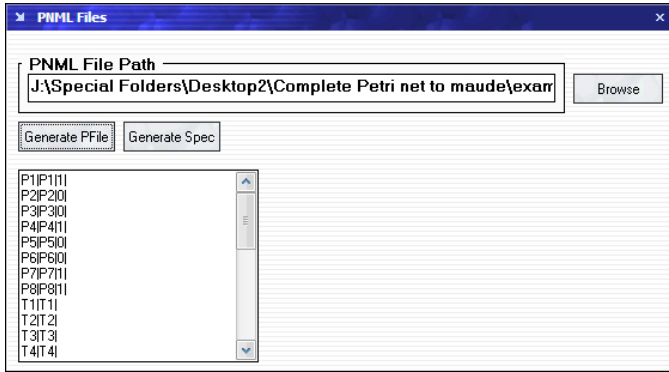
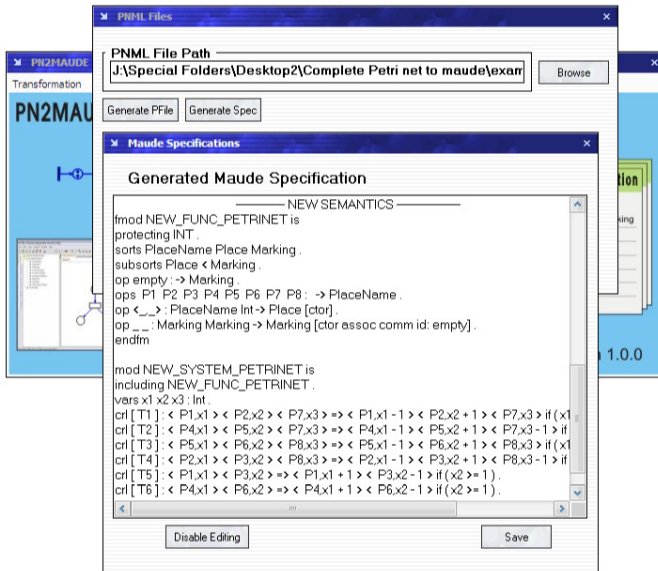


Figure 9. PN2Maude Interface Window for Importing PNML File

## 5.4 Generated Specification Interface Window

Figure 10 shows the interface window with the generated specification.



## 6 Conclusion and Future Work

In this paper, we presented PN2Maude, which is a developed tool for the automatic generation of the Maude specification for a Petri net following the existing and improved semantics based on its mathematical or PNML descriptions. We intend — in the near future — to include a Petri net editor within PN2Maude in order to facilitate the process of Petri net-based modeling for developers and users of our tool. In addition, we aim to develop a web version of PN2Maude and/or a plug-in for Eclipse in order to ensure its wide distribution for users.

## References

- [1] W. Reisig and G. Rozenberg, *Carl Adam Petri: Ideas, Personality, Impact*. Springer, 2019.
- [2] E. Huang, L. F. McGinnis, and S. W. Mitchell, “Verifying sysml activity diagrams using formal transformation to petri nets,” *Systems Engineering*, vol. 23, no. 1, pp. 118–135, 2020.
- [3] P. Singh and L. Singh, “Verification of safety critical and control systems of nuclear power plants using petri nets,” *Annals of Nuclear Energy*, vol. 132, pp. 584–592, 2019.
- [4] D. Buchs, S. Klikovits, and A. Linard, “Petri nets: A formal language to specify and verify concurrent non-deterministic event systems,” in *Foundations of Multi-Paradigm Modelling for Cyber-Physical Systems*. Springer, 2020, pp. 177–208.
- [5] J. Dong, J. Jiao, H. Xia, and J. Chu, “Safety simulation and analysis for complex systems concurrency based on petri net and stateflow model,” in *2019 Annual Reliability and Maintainability Symposium (RAMS)*. IEEE, 2019, pp. 1–7.
- [6] W. J. Thong and M. Ameen, “A survey of petri net tools,” in *Advanced Computer and Communication Engineering Technology*. Springer, 2015, pp. 537–551.

- [7] M. Alpuente, D. Ballis, F. Frechina, and J. Sapiña, “Exploring conditional rewriting logic computations,” *Journal of Symbolic Computation*, vol. 69, pp. 3–39, 2015.
- [8] M.-O. Stehr, J. Meseguer, and P. C. Ölveczky, “Rewriting logic as a unifying framework for petri nets,” in *Unifying Petri Nets*. Springer, 2001, pp. 250–303.
- [9] J. Meseguer, “Conditional rewriting logic as a unified model of concurrency,” *Theoretical computer science*, vol. 96, no. 1, pp. 73–155, 1992.
- [10] A. Boucherit, A. Khababa, and L. M. Castro, “Automatic generating algorithm of rewriting logic specification for multi-agent system models based on petri nets,” *Multiagent and Grid Systems*, vol. 14, no. 4, pp. 403–418, 2018.
- [11] T. Freytag, “Woped–workflow petri net designer,” *University of Cooperative Education*, pp. 279–282, 2005.
- [12] D. Gasevic and V. Devedzic, “Software support for teaching petri nets: P3,” in *Proceedings 3rd IEEE International Conference on Advanced Technologies*. IEEE, 2003, pp. 300–301.
- [13] P. Bonet, C. M. Lladó, R. Puijaner, and W. J. Knottenbelt, “Pipe v2. 5: A petri net tool for performance modelling,” in *Proc. 23rd Latin American Conference on Informatics (CLEI 2007)*, 2007.
- [14] J. Meseguer, “Rewriting as a unified model of concurrency,” in *International Conference on Concurrency Theory*. Springer, 1990, pp. 384–400.
- [15] —, “Twenty years of rewriting logic,” *The Journal of Logic and Algebraic Programming*, vol. 81, no. 7-8, pp. 721–781, 2012.
- [16] M. Clavel, F. Durán, S. Eker, S. Escobar, P. Lincoln, N. Martí-Oliet, J. Meseguer, R. Rubio, and C. Talcott, “Maude manual (version 3.0),” *SRI International–University of Illinois at Urbana-Champaign*. URL: <http://maude.cs.uiuc.edu>, 2019.

- [17] A. Boucherit, K. Barkaoui, and O. Hasan, “An enhanced rewriting logic based semantics for high-level petri nets,” in *The International Workshop on Petri Nets and Software Engineering 2021 co-located with the 42nd International Conference on Application and Theory of Petri Nets and Concurrency (PETRI NETS 2021)*, 2021.
- [18] A. Boucherit, L. M. Castro, A. Khababa, and O. Hasan, “Petri net and rewriting logic based formal analysis of multi-agent based safety-critical systems,” *Multiagent and Grid Systems*, vol. 16, no. 1, pp. 47–66, 2020.
- [19] K. Korenblat, O. Grumberg, and S. Katz, “Translations between textual transition systems and petri nets,” in *International Conference on Integrated Formal Methods*. Springer, 2002, pp. 339–359.
- [20] O. R. Ribeiro and J. M. Fernandes, “Translating synchronous petri nets into promela for verifying behavioural properties,” in *Industrial Embedded Systems, 2007. SIES’07. International Symposium on*. IEEE, 2007, pp. 266–273.
- [21] A. T. F. de Mello, M. C. Barbosa, D. J. dos Santos Filho, P. E. Miyagi, and F. Junqueira, “A transcription tool from petri net to clp programming languages,” in *ABCM Symposium Series in Mechatronics—Vol. 5, Section IV—Industrial Informatics, Discrete and Hybrid Systems*, 2012.
- [22] M. Szpyrka, A. Biernacka, and J. Biernacki, “Methods of translation of petri nets to nusmv language.” in *CS&P*, 2014, pp. 245–256.
- [23] L. J. Steggles, “Rewriting logic and elan: prototyping tools for petri nets with time,” in *International Conference on Application and Theory of Petri Nets*. Springer, 2001, pp. 363–381.
- [24] N. Boudiaf, A. Chaoui, and H. Bakha, “A rewriting logic based tool for ECATNet’s analysis: Edition and simulation steps description,” *European Journal of Scientific Research*, vol. 6, no. 2, pp. 16–27, 2005.



- [25] N. Boudiaf and A. Djebbar, “Towards an automatic translation of colored petri nets to maude language,” *International Journal of Computer Science & Engineering*, vol. 3, no. 1, pp. 1078–1083, 2009.
- [26] E. Kerkouche and A. Chaou, “A graphical tool support to process and simulate ecatnets models based on meta-modelling and graph grammars,” *INFOCOMP*, vol. 8, no. 4, pp. 37–44, 2009.

Ammar Boucherit, Messaoud Abbas,  
Mohammed Lamine Lamouri,  
Osman Hasan

Received April 17, 2023  
Revised June 15, 2023  
Accepted June 15, 2023

Ammar Boucherit<sup>1</sup>, Messaoud Abbas<sup>2</sup>, Mohammed Lamine Lamouri<sup>3</sup>  
<sup>1,2,3</sup>LIAP Laboratory, University of El Oued,  
PO Box 789, El Oued 39000, Algeria

<sup>1</sup>ORCID: <https://orcid.org/0000-0002-1617-0050>

E-mail: [ammam-boucherit@univ-eloued.dz](mailto:ammam-boucherit@univ-eloued.dz)

<sup>2</sup>ORCID: <https://orcid.org/0000-0002-7998-9020>

E-mail: [messaoud-abbas@univ-eloued.dz](mailto:messaoud-abbas@univ-eloued.dz)

<sup>3</sup>ORCID: <https://orcid.org/0000-0002-1074-624X>

E-mail: [lamouri-mohamedlamine@univ-eloued.dz](mailto:lamouri-mohamedlamine@univ-eloued.dz)

Osman Hasan

ORCID: <https://orcid.org/0000-0003-2562-2669>

SEECS, National University of Sciences and Technology (NUST),  
Islamabad, Pakistan

E-mail: [osman.hasan@seecs.nust.edu.pk](mailto:osman.hasan@seecs.nust.edu.pk)