International Journal of Innovative Computing, Information and Control
Volume **9**, Number **11**, November **2013**

ICIC International ©2013 ISSN 1349-4198
pp. 4285–4302

ON THE VERIFICATION OF G-NETS MODELS USING GRAPH TRANSFORMATIONS

Elhillali Kerkouche[1], Allaoua Chaoui[2], Khaled Khalfaoui[1]
and Raida Elmansouri[2]

[1]Department of Computer Science
University of Jijel
Jijel, Algeria
{ elhillalik; kh_khalfaoui }@yahoo.fr

[2]MISC Laboratory
Department of Computer Science and its Application
University of Constantine 2
Constantine, Algeria
{ a_chaoui2001; raidaelmansouri }@yahoo.fr

Received November 2012; revised April 2013

Abstract. *G-Nets are a kind of high level Petri Nets used for the modular design and specification of complex and distributed software systems. They integrate Petri net theory with the Object Oriented approach. The intention of this integration is to accumulate the advantages of both the formal treatment of Petri Nets and the modularity of the Object Oriented methodologies. However, G-Nets lack at present a formal analyzer tool enabling the verification of system properties. To overcome this limitation, a G-Nets specification can be translated (manually) to a semantically equivalent Predicate/Transition Nets (PrT-Nets). This transformation is performed in order to exploit existing analyzer tools for PrT-Nets to verify G-Nets specification. One of the most used analyzers for PrT-Nets is PROD reachability analyzer tool. In this paper, we propose a framework and a tool support to facilitate the design and verification of complex software systems using G-Nets formalism. The approach is based on the use of Graph transformation techniques that perform two automatic transformations: 1) the transformation of a G-Net model into an equivalent PrT-Net and 2) the translation of the obtained PrT-Net into PROD description language. Our framework is illustrated through an example.*

Keywords: G-Nets, PrT-Nets, PROD, Meta-modeling, Graph transformations, AToM^3

1. **Introduction.** In software engineering, the use of well-defined and loosely-coupled modules is a widely accepted design principle in practice [1]. Moreover, the need for formal techniques to support software development becomes even more essential over the last years [2]. Formal techniques are useful to precisely specify a system, and as a consequence to verify properties about the design.

G-Nets [3] are a kind of high level Petri Nets defined to provide the necessary support for these principles. They belong to an integration of Petri Nets theory with the Object Oriented software engineering approach. The motivation of this integration is to take benefits from the mathematical foundations of Petri Nets and at the same time to gain advantages of the modular, Object Oriented approach for the specification and prototyping of complex software systems. Furthermore, they reduce the gap between design and analysis.

The G-Nets notations include the concepts of module and system structure into Petri Nets, and promote abstraction, encapsulation and loose-coupling among the modules.

A system specified by G-Nets (called a *G-Nets specification*) consists of a collection of independent modules called G-Nets defined in terms of system structures. Similarly, an object in the object oriented concept, a G-Net satisfies the property of encapsulation. In this latter, no G-Net will be able to directly affect internal detail of another one. Rather, a G-Net can only access another G-Net through a well defined mechanism called *G-Net abstraction*. G-Net abstraction defines the interface between the G-Net and other G-Nets. It consists of a set of methods specifying the operations and services defined by the G-Net. The internal implementation of each method in the G-Net is described in terms of Petri nets notations.

Despite its well-defined semantics, G-Nets lack at present a formal analyzer tool enabling the analysis and verification of different system properties. To overcome this limitation, a formal transformation technique has been proposed in [3], which transforms a G-Nets specification to a set of Predicate/Transition Nets (PrT-Nets) models [2]. Each PrT-Nets model corresponds with a method in the G-Nets specification and has the equivalent semantics as the latter. This transformation aims at exploiting the existing formal analysis tools with a variety of analysis techniques for PrT-Nets on the obtained PrT-Nets models. PROD analyzer [4] is one of the most used tools for PrT-Nets. PROD is a text-based reachability analysis tool developed by Digital Systems Laboratory in Helsinki University of Technology. Currently, no graphical user-interface exists. The description language of PrT-Nets in PROD is the C pre-processor language extended by net description directives. A net description is compiled into an executable reachability graph generator program. From this graph, commands can be used to analyze PrT-Nets models.

To be effective, the G-Nets formalism should be supported by a tool requiring minimum user effort in modeling and analysis, thus allowing rapid development of systems. In this paper, we propose a formal framework and a tool support for the specification and verification of complex software systems using G-Nets formalism. To meet this objective, we propose to use Meta-Modeling [5] and Graph Transformation techniques [6] that support model evolution and manipulate models as instances of meta-models. The modeling process is implemented as a visual environment that allows G-Nets specifications according to the G-Nets meta-model. The verification process is implemented by graph transformation techniques that perform two automatic transformations: 1) the transformation of a G-Net model into an equivalent PrT-Nets, and 2) the translation of the obtained PrT-Nets into PROD description. These automatic transformations will help to avoid human errors and to minimize user efforts. The ideas presented above are implemented in AToM^3 (a tool for multi-formalism and meta-modeling). It is developed at the Modeling, Simulation and Design Lab in the School of Computer Science of McGill University [7].

In the present work, we have defined a meta-model for G-Nets formalism and another for PrT-Nets formalism. Then, the meta-modeling tool AToM^3 is used to automatically generate visual modeling tool for both formalisms according to their proposed meta-model. Also, we have proposed two graph transformation grammars. The first one performs the transformation of the specified G-Nets models to semantically equivalent PrT-Nets models according to the approach proposed in [3]. The second one translates the obtained PrT-Nets models into their equivalent descriptions in PROD input language.

This paper is organized as follows. Section 2 outlines some related works. We recall some basic notions about G-Nets formalism in Section 3, and their transformation procedure into PrT-Nets models in Section 4. In Section 5, we give an overview of the AToM^3 tool. In Section 6, we define a meta-model for G-Nets and another for PrT-Nets for generating visual tool for both formalisms. In Section 7, we propose two graph transformation grammars. The first one is to automatically transform G-Nets model into semantically equivalent PrT-Nets model. The second one is to translate the resulted PrT-Nets model into its description in PROD input language. In Section 8, we illustrate our framework through an example. Finally, Section 9 concludes the paper and gives some perspectives of this work.

2. **Related Works.**

When designing complex software systems, the need for structuring mechanisms which allow working with selected parts of the model and abstracting low level of other parts is crucial. On the other hand, the major classes of software systems introduce additional complexities into system design, and make it even harder to verify the design. Therefore, the need for formal techniques to support software development becomes even more essential. The combination of Petri net theory with the modular Object Oriented software engineering approach had emerged as a solution to tackle this problem. In addition to G-Nets, there are several formalisms in this direction, we may cite [8-14]. However, none of these works incorporates the encapsulation mechanism as defined in Object Oriented approach, where each module hides the internal details and communicates through well defined interfaces. The G-Nets notations provide a strong support to this principle.

It must be noted that these formalisms (including G-Nets) require more work in the analysis procedure due to primitives and notations used in communication between modules. Therefore, they should be supported by tools to help users in the analysis process.

Building a modeling tool from scratch is a hard task. Meta-modeling [5] approach is useful to deal with this problem since it allows the modeling of the formalisms themselves. In addition to AToM^3, there are several visual tools used to describe formalisms using meta-modeling like GME [15], MetaEdit+ [16] and other tools from the Eclipse Generative Modeling Tools (GMT) project such as Eclipse Modeling Framework (EMF) [17] and Graphical Modeling Framework (GMF) [18]. In most of these tools, model transformations have to be described in low-level textual languages. In AToM^3, the user expresses such transformations by means of graph transformation grammar models. Graph grammars are graphical, declarative and high-level way to express transformations.

There are also similar tools which manipulate models by means of graph grammars, such as PROGRES [19], GReAT [20], FUJABA [21] and AGG [22]. However, none of these have their own meta-modeling layer. Some of them are complemented with support for meta-modeling (for example, the GReAT model transformation engine is combined with GME).

The combined use of meta-modeling and graph grammars taken in AToM^3 allows users to benefit from the advantages of both meta-modeling and graph grammars [23]. The AToM^3 tool is proved to be very powerful, allowing the meta-modeling and the transformations of known formalisms. In [24], the authors presented a transformation between Statecharts (without hierarchy) and Petri Nets. In [25], the authors proposed an approach for transforming UML Statechart and collaboration diagrams to colored Petri nets models for analysis purposes.

In this paper, we propose an extended version of our paper published in [26]. It consists of a framework and a tool support to facilitate the design and analysis of complex software systems using G-Nets formalism.

3. **G-Nets Notations.**

G-Nets formalism is an object based high level Petri Nets introduced by [3], which extensively adopts object oriented structuring into Petri Nets. The intention is to profit from the strengths of both approaches. Petri nets offer a clean formalism for formal specification and verification of software, whereas object orientation offers formalism for abstraction, encapsulation and modular design [1]. As mentioned earlier, a G-Nets specification consists of a set of independent G-Nets modules representing the structure of a software system [3,27].

Structurally, a G-Net is composed of two parts: a special place called Generic Switch Place (*GSP*) and an Internal Structure (*IS*). The *GSP* provides the abstraction of the module. It contains the declaration of all G-Net attributes and methods that serve as an interface between the G-Net and the rest of the system. The Internal Structure (*IS*) is the hidden part of the G-Net. It represents the detailed design of the G-Net. The notation used for *IS* specification is an extended Petri Net notation. More precisely, G-Nets places represent computational primitives, and transitions with arcs represent connections among the primitives.

In a G-Net module, there exist three special kinds of places: Initial places, Goal places and Instantiated Switch Place (*ISP*). Firstly, Initial places are used to deposit new tokens inside a G-Net, denoting a new activation of some of its methods. Each method has its initial place defined in *GSP* place of the G-Net. Secondly, Goal places are used to indicate the termination of the methods. The presence of tokens inside a goal place is interpreted as results of the associated active method. Finally, *ISP* places are the element used to invoke methods in the same or in other G-Nets. An *ISP* place is a tuple (*G-Net_name, mtd*), where *G-Net_name* is the invoked G-Net and *mtd* is the specified method. When a token is deposited in an *ISP* place, the specified method in the invoked G-Net is activated by sending a token to its initial place. When the invoked G-Net terminates its activity, the resulting token is sent back to the invoker G-Net through the same *ISP* place.

The depth of recursion and the hierarchy of method calls are dynamically known due to the token structure in the G-Nets formalism. A token is defined as a triple (*seq, sc, msg*), where *seq* is called propagation sequence, *sc* is called the status color, and *msg* is called the message of the token. The propagation sequence (*seq*) of a token carries the history of G-Nets that have been traversed. The status color (*sc*) has two possible values (*before* or *after*), depending on whether the primitive action at the place is taken or not. The message field (*msg*) of a token is a list of application values. A formal definition of G-Nets can be found in [3].

The graphical notation for G-Nets is shown in Figure 1, and the method calls mechanism in G-Nets is illustrated in Figure 2.

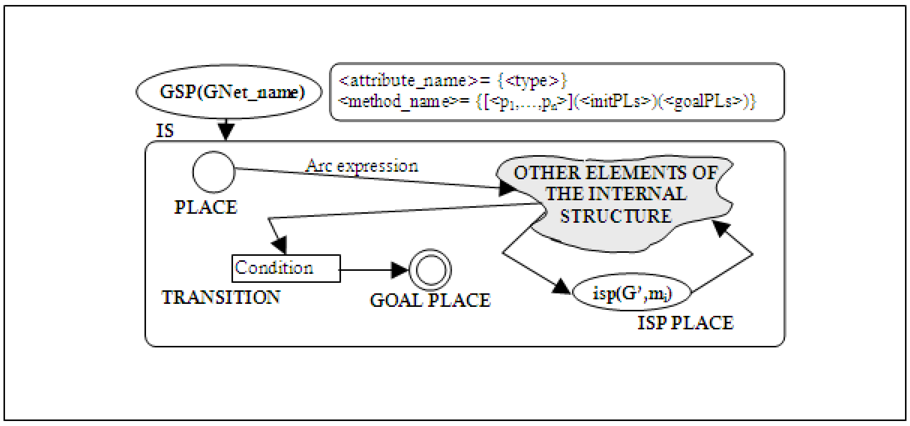

GSP(GNet_name)
<attribute_name>= {<type>}
<method_name>= {[<p1,...,pn>](<initPLs>)(<goalPLs>)}
IS
PLACE
Arc expression
OTHER ELEMENTS OF THE INTERNAL STRUCTURE
Condition
TRANSITION
GOAL PLACE
isp(G',mi)
ISP PLACE

FIGURE 1. Notations used to represent a G-Net

4. **Transformation from G-Nets to PrT-Nets.** G-Nets still lack a formal analyzer tool that allows proving properties about the system design. In order to overcome this limitation, a formal transformation technique has been proposed in [3]. This technique

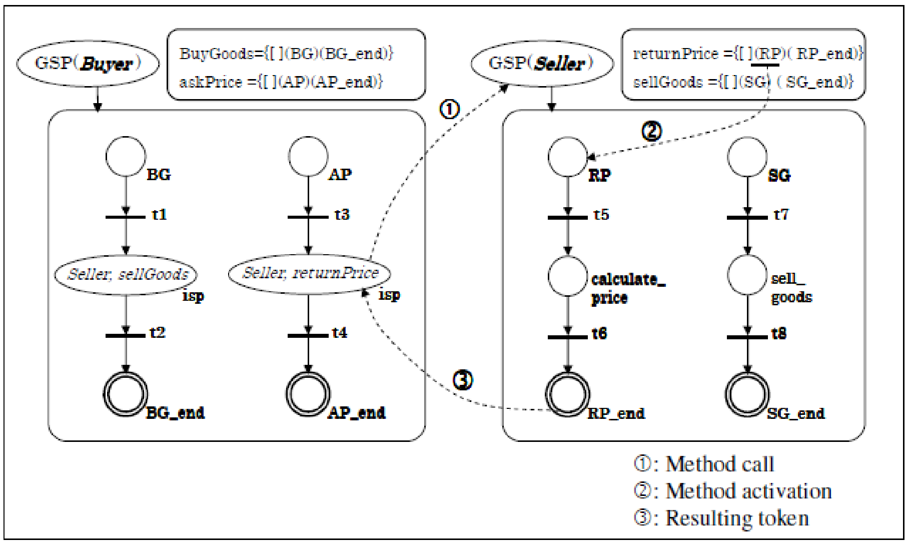


FIGURE 2. Invocation mechanism: buyer and seller G-Nets

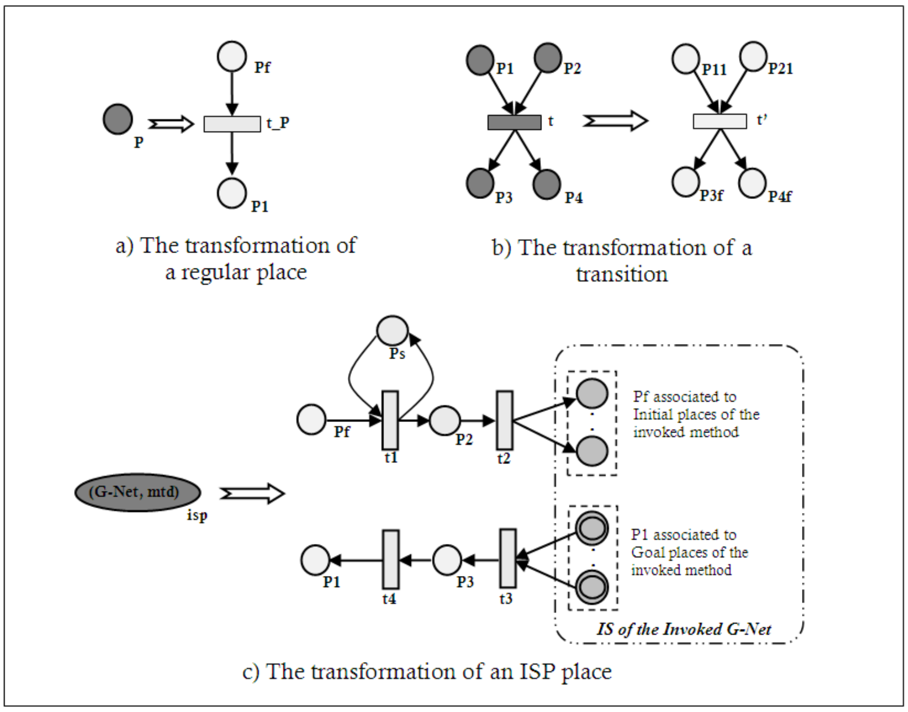

P
Pf
t_P
P1
a) The transformation of a regular place
P1
P2
t
P3
P4
P11
P21
t'
P3f
P4f
b) The transformation of a transition
(G-Net, mtd)
isp
Ps
Pf
t1
P2
t2
Pf associated to Initial places of the invoked method
P1
t4
P3
t3
P1 associated to Goal places of the invoked method
IS of the Invoked G-Net
c) The transformation of an ISP place

FIGURE 3. Transformation technique

translates a G-Nets specification to a set of PrT-Nets models, where each PrT-Nets model corresponds to a method in the G-Nets specification. This transformation aims at providing a basis for adapting PrT-Nets analysis techniques for use with G-Nets.

The transformation algorithm accepts a G-Nets specification and a method as input, and produces a semantically equivalent PrT-Nets model corresponding to the method as output. The basic idea of the transformation technique is illustrated in Figure 3.

The idea behind the transformation can be described as follows.

1. A regular place (including initial and goal places) can be transformed to PrT-Nets representation shown in Figure 3(a). The PrT-Nets transition represents the computational primitive associated to the G-Nets place. The input and output places

(Pf and P1) of this transition simulate the semantics of status color of the G-Nets token.

2. A transition in G-Nets can also be easily transformed to PrT-Nets notation as shown in Figure 3(b).
3. An *ISP* place denotes a G-Nets invocation. The transformation of an *ISP* place is shown in Figure 3(c), which represents the invocation interface between the G-Net containing the *ISP* place and the invoked G-Net in Prt-Nets model.
4. The PrT-Nets representation of the invoked (*G-Net, mtd*) is also created in the same way, and so forth recursively.

A formal presentation of the technique and associated proofs can be found in [28]. We note that the transformation in this approach is performed manually. In Section 7, we propose our graph transformation approach to perform the transformation automatically.

5. **AToM^3 and Graph Grammars.**

AToM^3 [7] is a visual tool for meta-modeling and model transformations. Being implemented in Python [29], AToM^3 is able to run without any change on all platforms for which an interpreter for Python is available.

By means of meta-modeling, we can describe or model the different kinds of formalisms needed in the specification and design of systems. The AToM^3 meta-layer allows a high-level description of models using Entity Relationship (ER) formalism or UML Class Diagram formalism extended with the ability to express constraints. Based on these descriptions, AToM^3 can automatically generate tools to manipulate (create and edit) models in the formalisms of interest [5].

AToM^3's capabilities are not restricted to these manipulations. AToM^3 also supports graph transformation, which uses graph grammars to visually guide the procedure of model transformation. Model transformation refers to the automatic process of converting, translating or modifying a model of a given formalism into another model that might or might not be in the same formalism.

Graph grammar [6] is a generalization of Chomsky grammar for graphs. It is a formalism in which the transformation of graph structures can be modeled and studied. The main idea of graph transformation is the rule-based modification of graphs as shown in Figure 4.

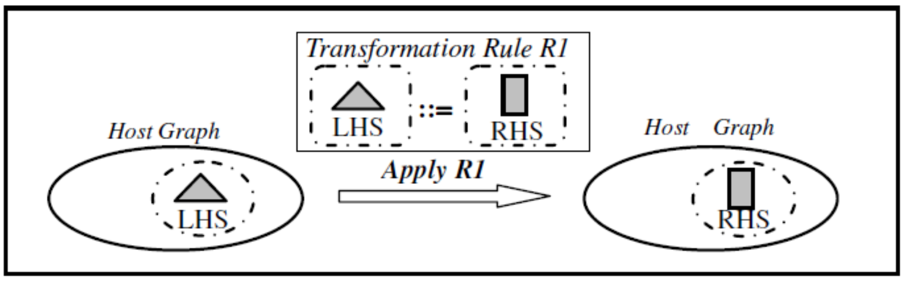


FIGURE 4. Rule-based modification of graphs

Graph grammars are composed of production rules, each having graphs in their left and right hand sides (LHS and RHS). In the transformation process, rules are evaluated against an input graph, called the host graph. If a matching is found between the LHS of a rule and a subgraph of the host graph, then the rule can be applied. When a rule is applied, the matching subgraph of the host graph is replaced by the RHS of the rule. Rules can have applicability conditions, as well as actions to be performed when the rule is applied. Also, rules are ordered according to a priority assigned by the users and are checked from the higher priority to the lower priority. After a rule matching and subsequent application, the graph transformation system starts again the search. The graph grammar execution ends when no more matching rules are found.

In the next sections, we will discuss how we use AToM^3 to create our formal framework for the specification and analysis of G-Nets models.

6. Meta-Modeling of G-Nets and PrT-Nets.

In this section, we will use the meta-modeling tool AToM^3 to define two meta-models, one for G-Nets formalism and the other for PrT-Nets formalism. To define a formalism, one has to provide abstract syntax (denoting constructs of the formalism, their attributes, relationships and constraints) as well as concrete graphical syntax information (the appearance of constructs, relationships, and possible graphical constraints in the visual tool). The meta-formalism used in our work is the UML Class Diagram and the constraints are expressed in Python code.

Following the description of G-Nets formalism given in Section 3, we have proposed meta-model G-Nets with four Classes and five Associations as shown in Figure 5.

The class “G*NetsGSP*” represents *Generic Switch Place (GSP)* and contains *name* attribute which must be unique. It has also two lists for G-Nets Attributes (*AS*) and G-Nets methods (*MS*). The class “G*NetsIS*” represents *Internal Structure* (*IS*). The third class “*GNetsPlace*” describes places; it has a *name*, a *type, an invokedGNets, a usingMethod* and *tokens* attributes. The *type* attribute indicates the type of the place

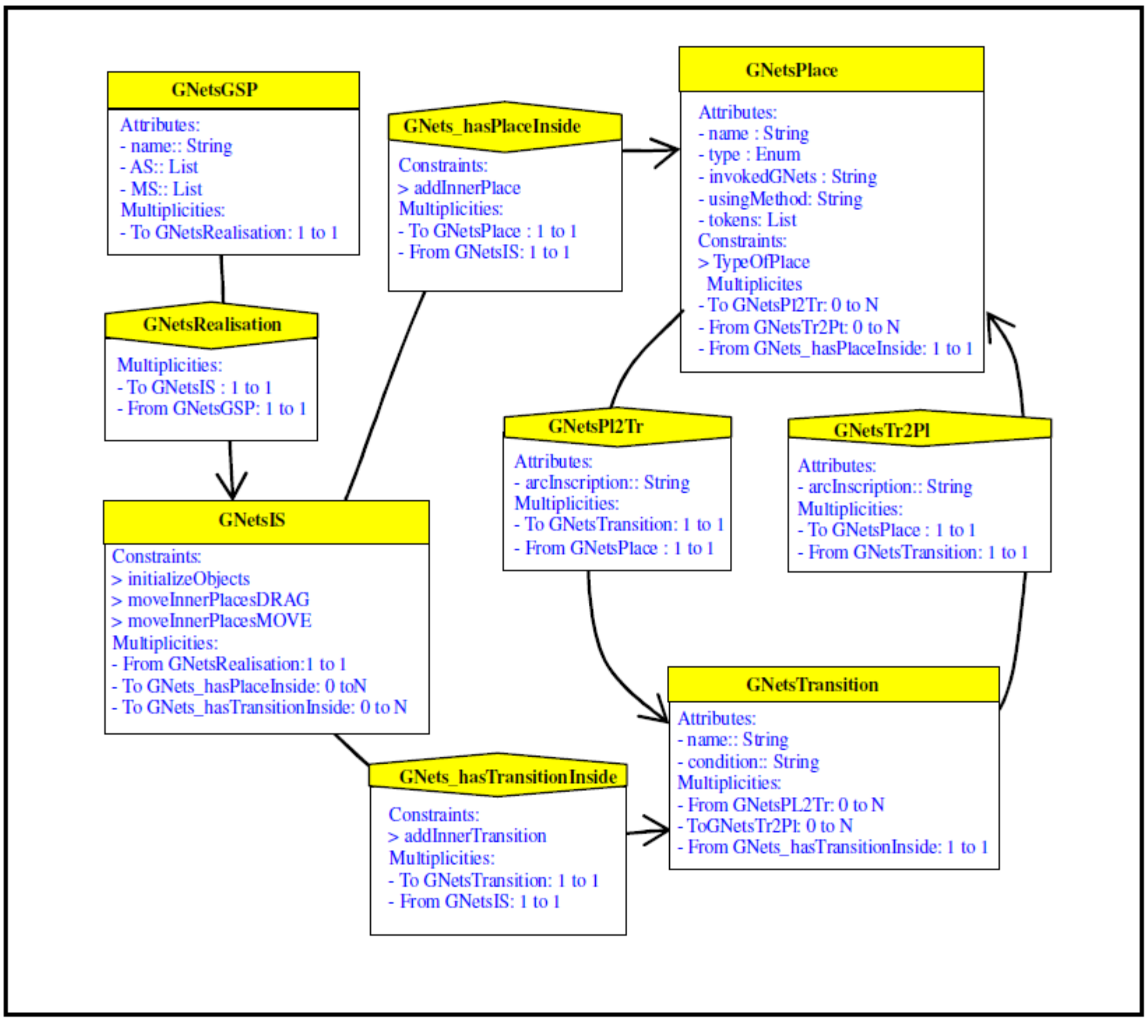
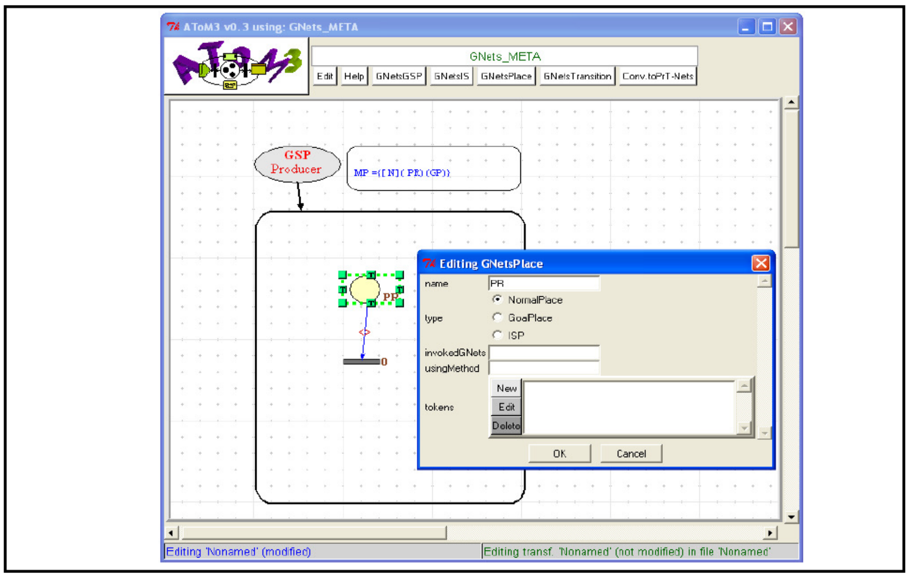


FIGURE 5. G-Nets meta-model



name PR
type [✓ NormalPlace] GoalPlace ISP
invokedGNets
usingMethod
tokens

Editing 'Nonamed' (modified)
Editing transf. 'Nonamed' (not modified) in file 'Nonamed'

FIGURE 6. Generated tool for G-Nets

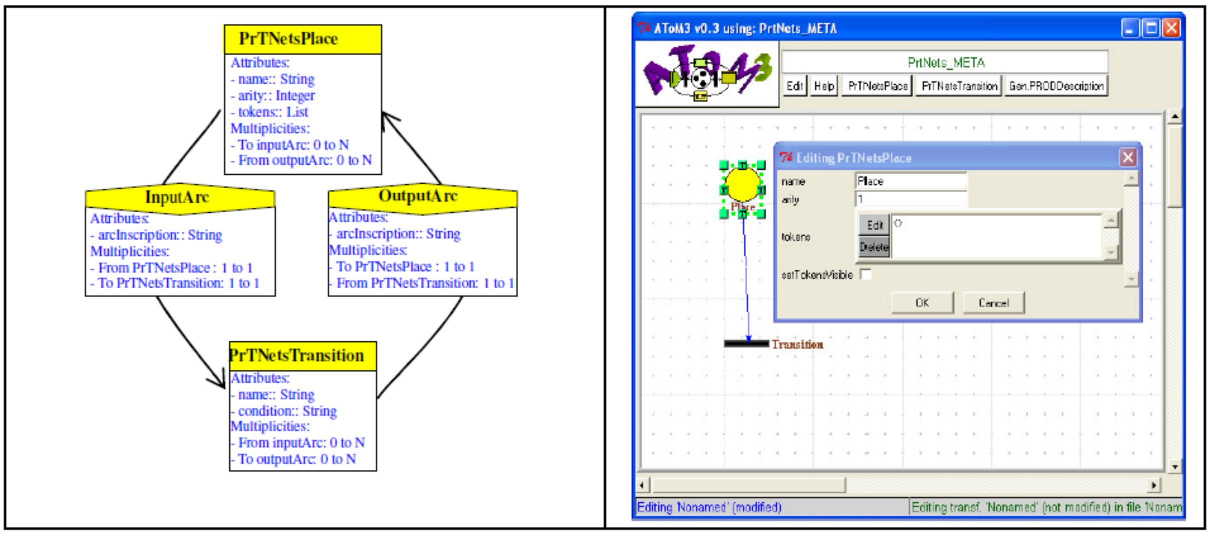
(Normal place, Goal place or Instantiated Switch Place (*ISP*)). The *invokedGNets* and *usingMethod attributes* are used only if the type of the place is *ISP*. The last class "*GNetsTransition*" represents transitions. It contains a *name* attribute which must be unique and a *TransitionCondition* attribute which specifies the enabling condition of the transition. *GSP* and *IS* are related by means of "*GNetsRealisation*" association which connects each *GSP* to its realization in *IS*. The two associations "*GNest_hasPlaceInside*" and "*GNest_hasTransitionInside*" are used to specify places and transitions (respectively) which are included in a given IS. Finally, "*GNetsPl2Tr*" and "*GNetsTr2Pl*" represent input arcs and output arcs between places and transitions.

We have also specified the visual representation of G-Nets entities according to the notation presented in Figure 1. To fully define our meta-model, we have added graphical constraints which are used to assure a correct appearance of G-Nets models. For example, "*TypeOfPlace*" constraint is used to give appropriate visual representation of places according to their types.

Given our meta-model, we have used AToM^3 tool to generate a visual modeling environment for G-Nets models. The generated G-Nets modeling tool and the dialog box to edit a place are shown in Figure 6. The user interface buttons allow the user to create the entities defined in the meta-model.

We have also defined a meta-model for PrT-Nets formalism and we have also used AToM^3 tool to generate a visual modeling tool. A PrT-Nets model consists of places, transitions, and arcs from places to transitions and from transitions to places. To meta-model PrT-Nets formalism, we have proposed two classes: "*PrTNetsPlace* " class to describe places and "*PrTNetsTransition*" to describe transitions. These classes are related with two associations named "*InputArc*" and "*OutputArc*" which represent input arcs and output arcs as shown in the left of Figure 7. The generated tool is shown in the right of Figure 7.

Since AToM^3 is a visual tool for multi-formalism modeling, we can show in a user interface of AToM^3 the two generated toolbars at the same time. Additionally, we have added a button in G-Nets toolbar that executes the graph grammar which transforms the G-Nets specification into a PrT-Nets model. Another button is also added in PrT-Nets toolbar that executes another graph grammar to generate PROD description of the



name: Place
arity: 1

FIGURE 7. PrT-Nets meta-model and the generated tool

resulted PrT-Nets models. In the next section, we will present these graph grammars and their use in AToM^3.

7. **Formal Framework for G-Nets.** The framework which we have obtained in the previous section by means of meta-modeling only allows the user to create, load and save models, as well as verify that they are syntactically correct. In this section, we improve these capabilities by means of graph grammar. More precisely, we have defined two graph grammars. The first one translates G-Nets specifications to semantically equivalent PrT-Nets models for further analysis. In order to exploit the PROD analyzer, the second graph grammar generates the equivalent PROD description of the resulted PrT-Nets models (see Figure 8).

As we mentioned earlier, graph transformation system iteratively applies a list of rules to the host graph. In the *LHS* of rules, the attributes of the nodes must be provided with attribute values which will be compared with the nodes attributes of the host graph during the matching process. These attributes can be set to $\langle ANY \rangle$ or have specific values. In order to specify the mapping between *LHS* and *RHS*, nodes in both *LHS* and

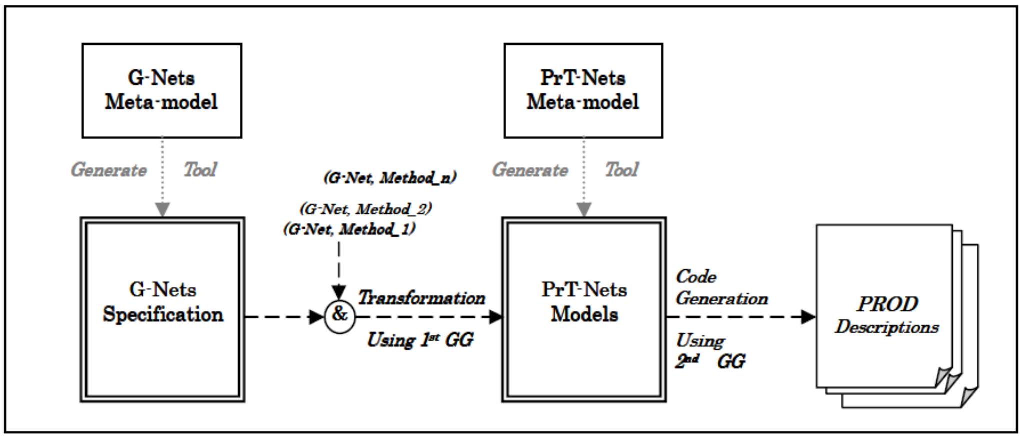


FIGURE 8. The proposed framework

RHS are identified by means of labels (numbers). If a node label appears in the *LHS* of a rule, but not in the *RHS*, then the node is deleted when the rule is applied. Conversely, if a node label appears in the *RHS* but not in the *LHS*, then the node is created when the rule is applied. Finally, if a node label appears both in the *LHS* and in the *RHS* of a rule, the node is not deleted. If a node is created or maintained by a rule, we must specify in the *RHS* the attributes' values after the rule application. In AToM^3, there are several possibilities. If the node label is already present in the *LHS*, the attribute value can be copied ($\langle COPIED \rangle$). We also have the option to give it a specific value or assign it a Python program to calculate the value ($\langle SPECIFIED \rangle$), possibly using the value of other attributes.

In the following subsections, we use AToM^3 to describe the two graph grammars for our framework.

7.1. **GG: transforming a G-Nets specification into a PrT-Nets model.** We have named this graph grammar G-Net2PrT-Nets. During the execution of GNet2PrT-Nets, the model is indeed a blend of G-Nets and PrT-Nets. However, when the graph grammar execution finishes, the resulting model is entirely a PrT-Nets model.

G-Net2PrT-Nets graph grammar has an initial action which creates global variables needed in the transformation process. *List_Visited_GNets_Names* is a list used to keep the names of transformed G-Nets. Each G-Net in *List_Visited_GNets_Names* is invoked using a method specified in the same range in *List_Visited_Mtd_Names* list. *Current_GNets_Name* and *Current_Mtd_Name* variables are used to specify the name of the G-Net and the name of the method under transformation. Finally, *List_Init_Place* is a list which contains the list of initial places for the method under transformation.

To start transformation, the initial action prompts the user to select both a G-Net and one of its methods from G-Nets specification. The selected G-Net name and method are kept in *Current_GNets_Name* and *Current_Mtd_Name* (respectively). To transform a selected pair (*Current_GNets_Name*, *Current_Mtd_Name*) into an equivalent model in PrT-Nets model using our *G-Net2PrT-Nets* grammar, we have proposed twenty-one rules which will be applied in ascending order. Note that each rule has a priority. The proposed rules are shown in Figure 9 and Figure 10, and described as follows.

Rules 1, 2, 3, 4, 5 and 6: *TrnsfArc_... (Priority resp. 1, 2, 3, 4, 5 and 6).* One of these rules is applied to replacing input arc or output arc of regular (normal or goal) places in G-Nets model with its equivalent in PrT-Nets model. To apply these rules, at least one of arc borders (place or transition) in G-Nets model must already be transformed. The non processed border of the arc will be transformed with these rules. For example, in the rule No. 1 (*TrnsfArc_VisitedPlace2VisitedTrans)*, both place and transition are previously processed. When this rule is applied, it removes the output arc of a G-Net place and creates an output arc in the PrT-Nets segment associated with G-Nets place to the PrT-Nets segment associated with G-Nets transition.

Rules 7, 8, 9 and 10: *TrnsfArc_... (Priority resp. 7, 8, 9 and 10).* These rules are similar to the previous rules, but they are applied to replacing input arc or output arc of *ISP* places in G-Nets model.

Rules 11, 12 and 13: *TrnsfArc_... (Priority resp. 11, 12 and 13).* Because multiple *ISP*s places corresponding to the same pair (*G-Net, Method*) may be used in the G-Nets specification, these rules are applied to ensuring that a pair (*G-Net, Method*) is only translated once. For example, the rule No. 11 (*TrnsfArc_VisitedTrans2CalledISP*) is applied when the invoked (*G-Net.Method*) in the ISP place was previously transformed. A pair (*G-Net. Method*) has already transformed if the G-Net name is in *List_Visited_GNets_Names* list and the method name is in *List_Visited_Mtd_Names* list. When applied, this rule locates the transformed *ISP* place which contains the same pair (*G-Net.Method*). Then, it removes the input arc of *ISP* place in G-Nets model and creates an input arc in the PrT-Nets segment associated with G-Nets transition to the PrT-Nets segment associated with the transformed ISP place.

Rule 14: *Gen_StartOfInvocation (Priority14)* is applied to relating the invocation interface created for the *ISP* place in the rule No. 18 to the PrT-Nets segment associated with the initial places of invoked (*G-Net.Method*).

Rule 15: *Trnsf_InitialPlace (Priority15)* is applied to locating the G-Net to be transformed, i.e. its name is equal to *Current_GNets_Name* value. Then, it associates with each initial place of the G-Net the equivalent PrT-Nets segment. We note that the names of all initial places are in *List_Init_Place* list.

Rule 16: *Gen_EndOfInvocation (Priority16)* is applied to relating the PrT-Nets segment associated with goal places of the invoked (*G-Net.Method*) to the invocation interface created for the *ISP* place in the rule No. 18.

Rule 17: *GetInitialPlaces (Priority17)* is applied to locating the G-Net whose name is equal to *Current_GNets_Name* value. Then, get the list of initial places of the method whose name is equal to *Current_Mtd_Name* value in *List_Init_Place* list.

Rule 18: *GenInterface_ISP2GNets (Priority18)* is applied to generating the invocation interface in PrT-Nets model which mimics the invocation mechanism in G-Nets (see Figure 3(c)). More precisely, this rule creates a PrT-Nets segment which interfaces the equivalent segment in PrT-Nets model of G-Net which contains the *ISP* place to the equivalent segment in PrT-Nets model of invoked G-Net. Furthermore, this rule terminates with action expressed in Python code. This action sets *Current_GNets_Name* variable to the invoked G-Net name and *Current_Mtd_Name* variable to the method name used in the *ISP* place. With these new values, the *G-Net2PrT-Nets* graph grammar performs the transformation of the invoked G-Net and relates their initial place and goal place when applying rules No. 14 and No. 16 respectively.

Rule 19: *DeleteGNetsPlace (Priority19)* is applied to removing all places in the G-Nets model.

Rule 20: *DeleteGNetsTransition (Priority20)* is applied to removing all transitions in the G-Nets model.

Rule 21: *DeleteGNetsGSP (Priority21)* is applied to removing all *GSP* and *IS* in the G-Nets specification.

G-Net2PrT-Nets graph grammar also has a final action which erases the global variables.

7.2. **GG: generating PROD description.**

We have named the second graph grammar *PrT-Nets2PROD*. *PrT-Nets2PROD* grammar has an initial action which opens the file where the PROD description will be generated. Also, it decorates all transition and place elements in the model with temporary attributes which will be used in the enabling conditions to apply the rules.

In transition elements, we use two attributes: *Current* and *Visited*. The *Current* attribute is used to identify the transition in the model whose code has to be generated, whereas the *Visited* attribute is used to indicate whether code for the transition has been already generated. In place elements, we use three attributes: *Visited, Visited_Input* and *Visited_Output*. The *Visited* attribute is used to identify the place whose code has been already generated. The *Visited_Input* attribute is used to indicate whether this place is processed as input place, whereas the *Visited_Output* attribute is used to indicate if this place is processed as output place in the generation of code for a given transition.

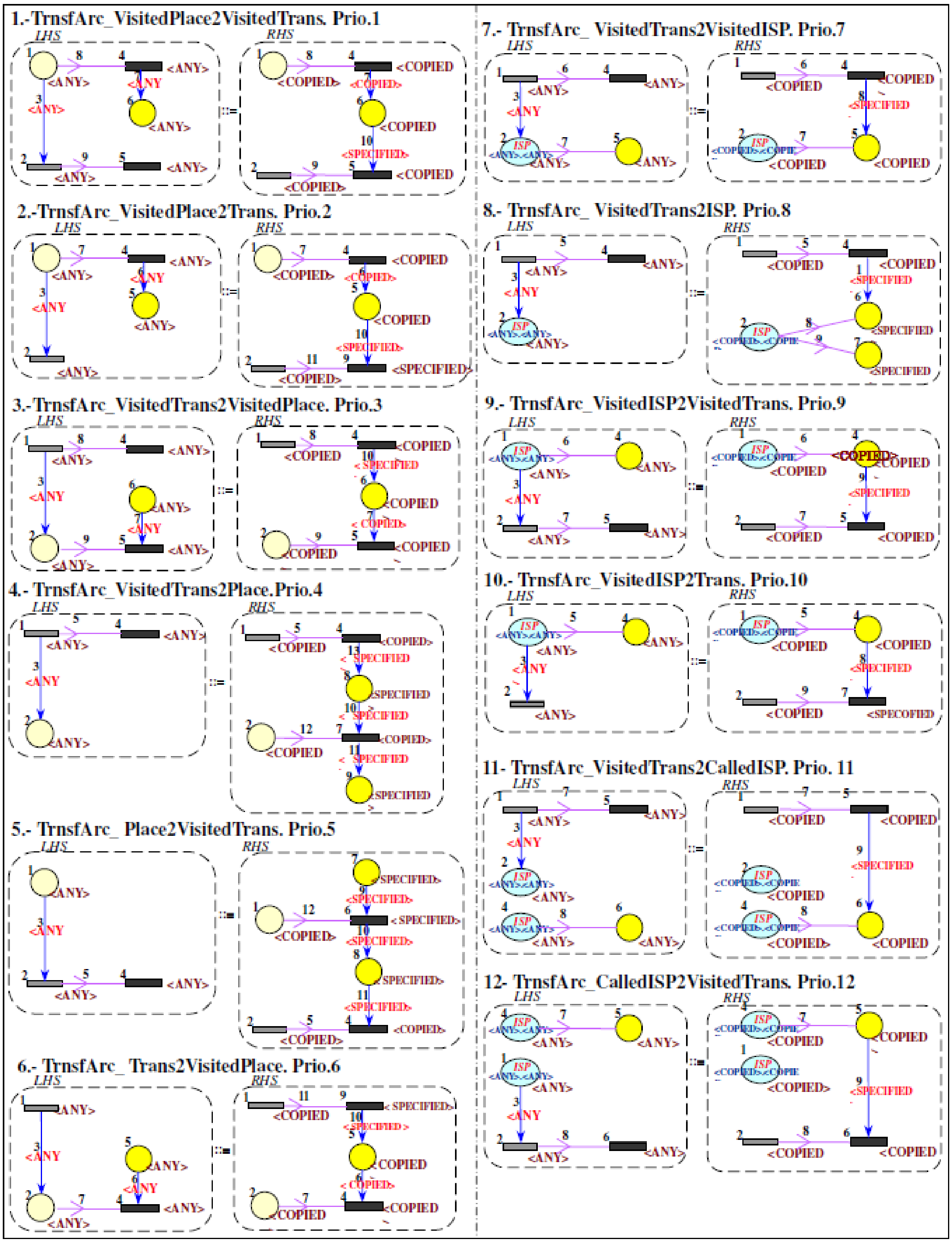

1.-TrnsfArc_VisitedPlace2VisitedTrans. Prio.1
2.-TrnsfArc_VisitedPlace2Trans. Prio.2
3.-TrnsfArc_VisitedTrans2VisitedPlace. Prio.3
4.- TrnsfArc_VisitedTrans2Place.Prio.4
5.- TrnsfArc_ Place2VisitedTrans. Prio.5
6.- TrnsfArc_ Trans2VisitedPlace. Prio.6
7.- TrnsfArc_ VisitedTrans2VisitedISP. Prio.7
8.- TrnsfArc_ VisitedTrans2ISP. Prio.8
9.- TrnsfArc_VisitedISP2VisitedTrans. Prio.9
10.- TrnsfArc_VisitedISP2Trans. Prio.10
11- TrnsfArc_VisitedTrans2CalledISP. Prio. 11
12- TrnsfArc_CalledISP2VisitedTrans. Prio.12
LHS
RHS
::=
<ANY>
<COPIED>
<SPECIFIED>
ISP

FIGURE 9. GG: Transform G-Nets specification into PrT-Nets, Rules 1-12

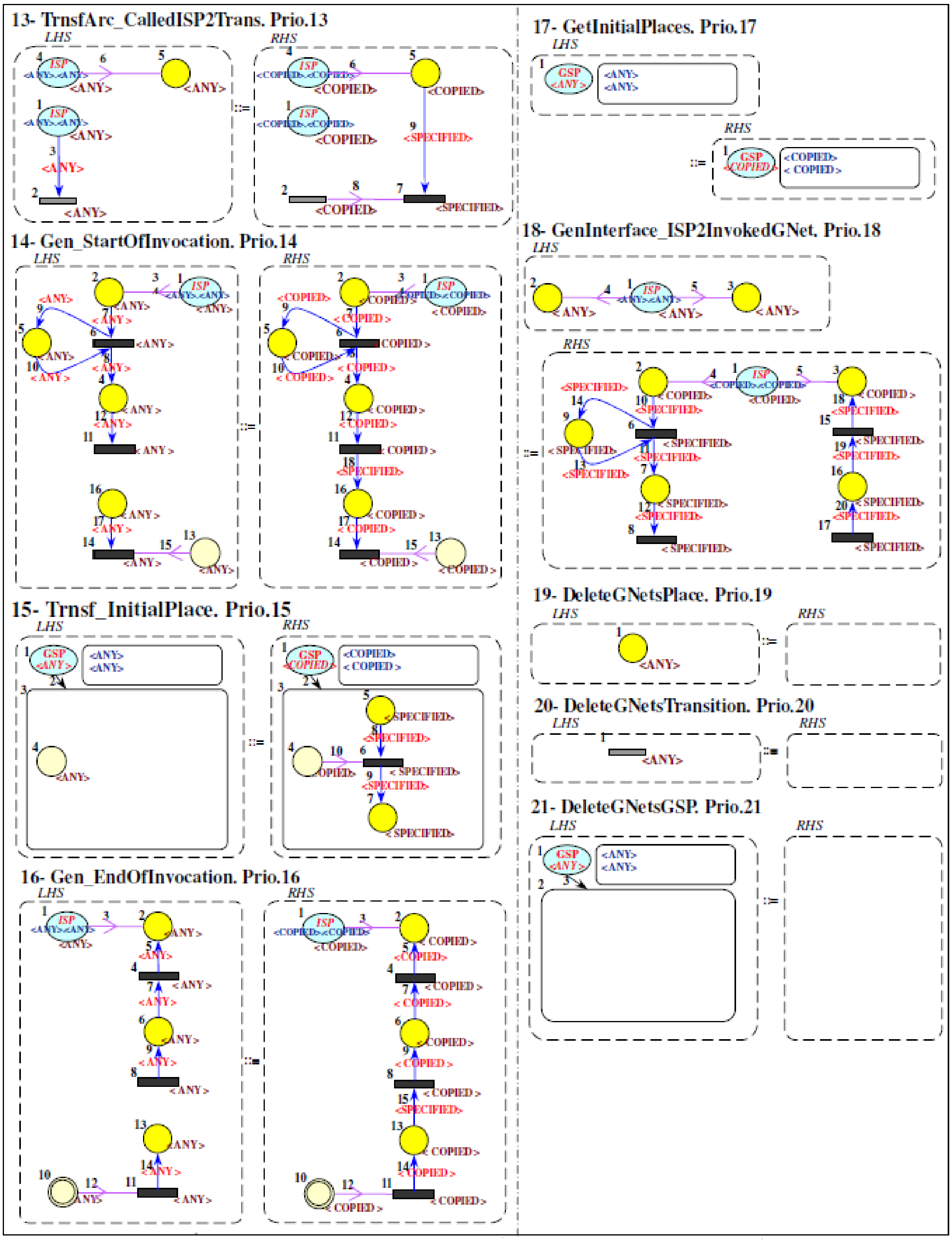

13- TrnsfArc_CalledISP2Trans. Prio.13
14- Gen_StartOfInvocation. Prio.14
15- Trnsf_InitialPlace. Prio.15
16- Gen_EndOfInvocation. Prio.16
17- GetInitialPlaces. Prio.17
18- GenInterface_ISP2InvokedGNet. Prio.18
19- DeleteGNetsPlace. Prio.19
20- DeleteGNetsTransition. Prio.20
21- DeleteGNetsGSP. Prio.21
LHS
RHS
::=
ISP
GSP
<ANY>
<COPIED>
<SPECIFIED>

FIGURE 10. GG: Transform G-Nets specification into PrT-Nets, Rules 13-21

In *PrT-Nets2PROD* graph grammar, we have proposed six rules which will be applied in ascending order by the rewriting system until no more rules are applicable. We are concerned here by code generation, so none of these rules will change the input PrT-Nets model. These rules are shown in Figure 11 and described as follows.

Rule 1: *genPlaceDescription (Priority 1)* is applied to locating a place not previously processed (Visited = = 0), and to generating the corresponding PROD description.

Rule 2: *genListOfInputPlace (Priority 2)* is applied to locating a place (not previously processed) which is related to current transition with an input arc, and to generating the corresponding PROD description.

Rule 3: *genListOfOutputPlace (Priority 3)* is applied to locating a place (not previously processed) which is related to current transition with output arc, and to generating the corresponding PROD description.

Rule 4: *EndOfTransDescription (Priority 4)* is applied to generating the appropriate PROD syntax depending on the condition of the transition, and to marking the transition as visited (Visited = 1).

Rule 5: *InitialisePlaceFlags (Priority 5)* is applied to locating and initialise flags attributes in places for processing the next transition.

Rule 6: *SelectTransToDescribe (Priority 6)* is applied to selecting a PrT-Nets transition that is not previously processed to generate its equivalent PROD syntax.

PrT-Nets2PROD graph grammar also has a final action which erases the temporary attributes from the entities and closes the output file.

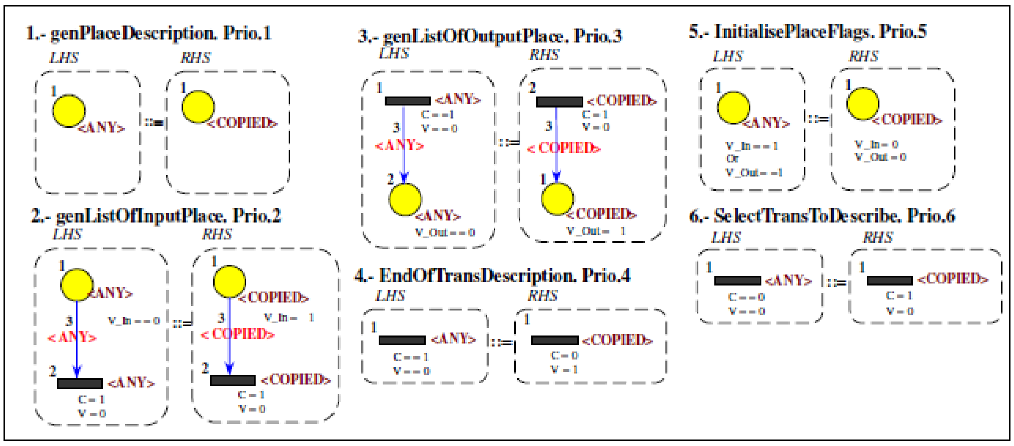

1.- genPlaceDescription. Prio.1
LHS
RHS
3.- genListOfOutputPlace. Prio.3
5.- InitialisePlaceFlags. Prio.5
2.- genListOfInputPlace. Prio.2
4.- EndOfTransDescription. Prio.4
6.- SelectTransToDescribe. Prio.6
<ANY>
<COPIED>

FIGURE 11. GG: Generate PROD description from a PrT-Nets model

8. Case Study: Producer/Consumer Problem.

The Producer/Consumer problem is a well known problem. It consists of the synchronization between one or more producers and one or more consumers. In this example, we assume that one producer is capable of producing n items and one consumer is able to consume one item at a time.

Figure 12 presents a G-Nets specification of Producer/Consumer problem created in our framework. For G-Net Producer, one method is defined and named *mp* (method produce). The function of the method *mp* is to produce n items to be consumed. When G-Net Producer is invoked, one token together with a field n (n expressing the number of items to be produced) is deposited in place *PR* (producer ready). The action associated with this place simply decrements the field n. If $n < 0$, transition *pr* (producer resumes) fires and the invocation of G-Net Producer returns when the *GP* (goal place) is reached. Otherwise, transition *rs* (request status consumer) fires and the token reaches *isp* (Consumer.ms), and

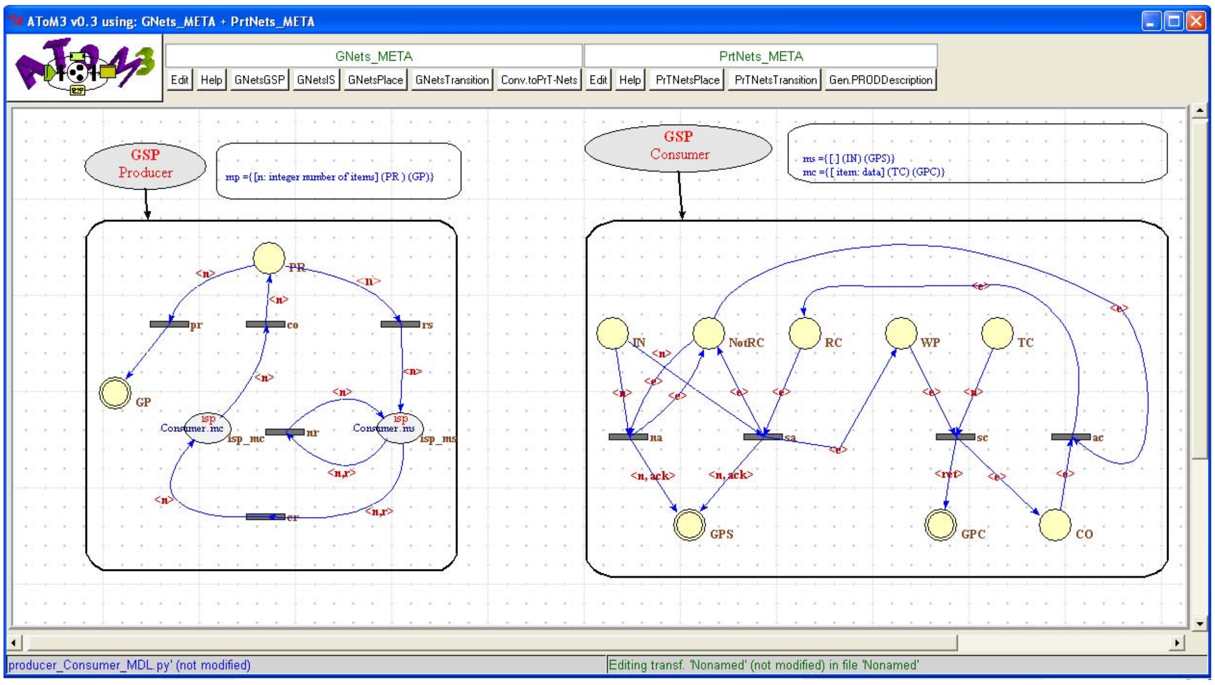


FIGURE 12. Producer/Consumer problem created in our framework

the G-Net Consumer is invoked using method *ms* (method status). If G-Net Consumer is not ready, transition *nr* (consumer not ready) fires, and the G-Net Consumer is invoked again. Otherwise, the consumer is ready and transition *cr* (consumer ready) fires. After the firing of *cr*, a token is deposited in *isp* (Consumer.mc) and G-Net Consumer is invoked with the method *mc* (method consume) together with the item to be consumed. When the token is returned from G-Net Consumer, transition *co* (consumed) fires and a token is put in place *PR* again. In the G-Net Consumer, eight places are defined: *IN* (Inquire net Consumer), *RC* (ready to consume), *NotRC* (not ready to consume), *TC* (trigger consumer), *CO* (consuming), *WP* (waiting producer), *GPS* (end status) and *GPC* (item accepted). These places are related to four transitions as shown in Figure 12. The defined transitions are: *na* (not available), *sa* (send acknowledge), *sc* (start consuming) and *ac* (already consumed).

In order to analyze the G-Nets specification of Producer/Consumer problem, we have to transform this specification into its equivalent PrT-Nets model. To realize this transformation in our framework, we have just to click on the "Conv.toPrT-Nets" button in the user interface that executes *G-Net2PrT-Nets* graph grammar defined in the previous section. The resulted PrT-Nets model of the automatic transformation is shown in Figure 13.

To perform the analysis of the resulted PrT-Nets model using PROD analyzer, we have to translate it into its equivalent PROD description. To generate PROD description in our framework, we have to click on the "Gen.PRODDescription" button in the user interface. This button executes *PrT-Nets2PROD* graph grammar defined in the previous section. The automatic generated file which contains the PROD description of Producer/Consumer problem is shown in Figure 14.

9. **Conclusion.**

In this paper, we have presented a formal framework based on the combined use of Meta-modeling and Graph Transformation Grammars for the specification

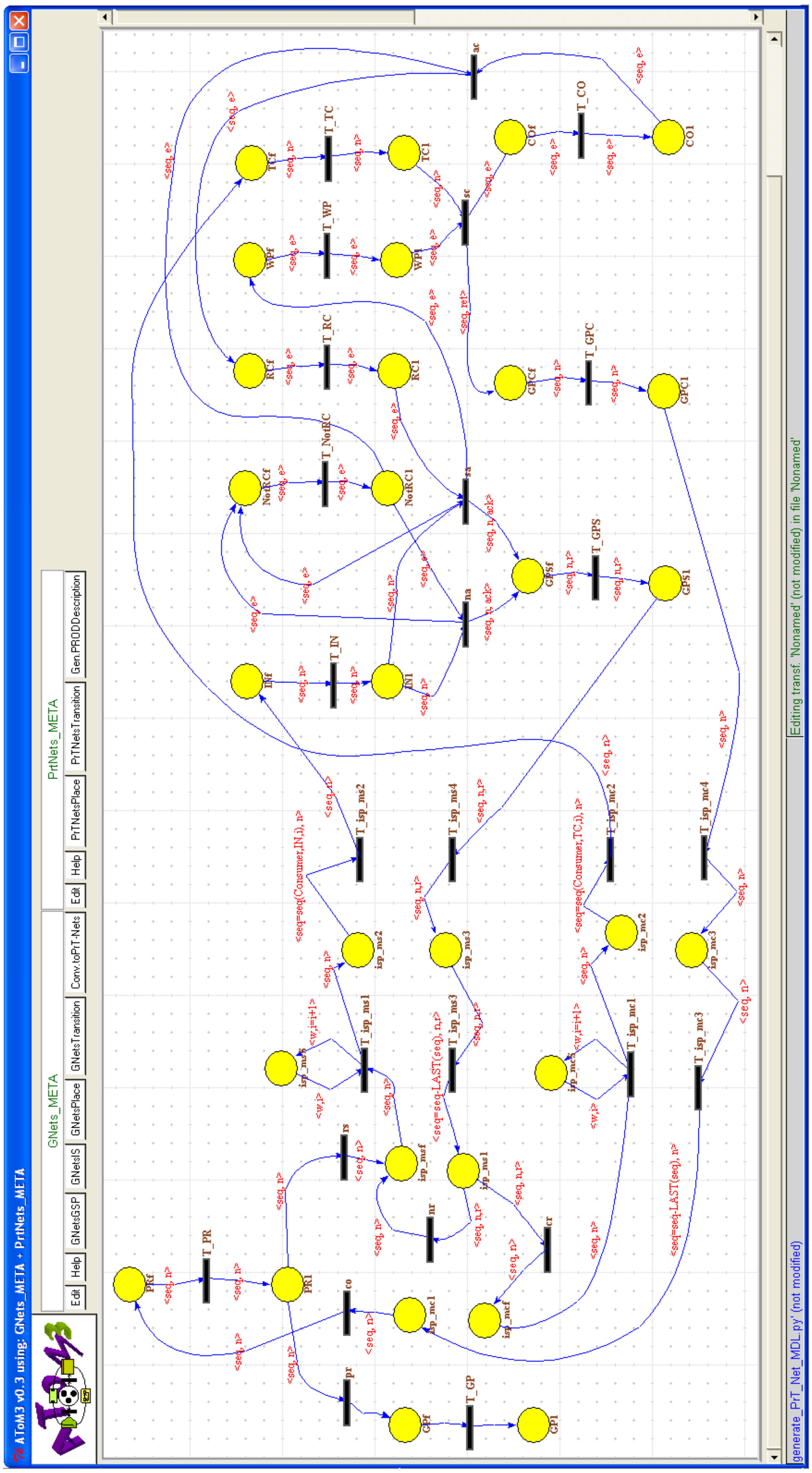
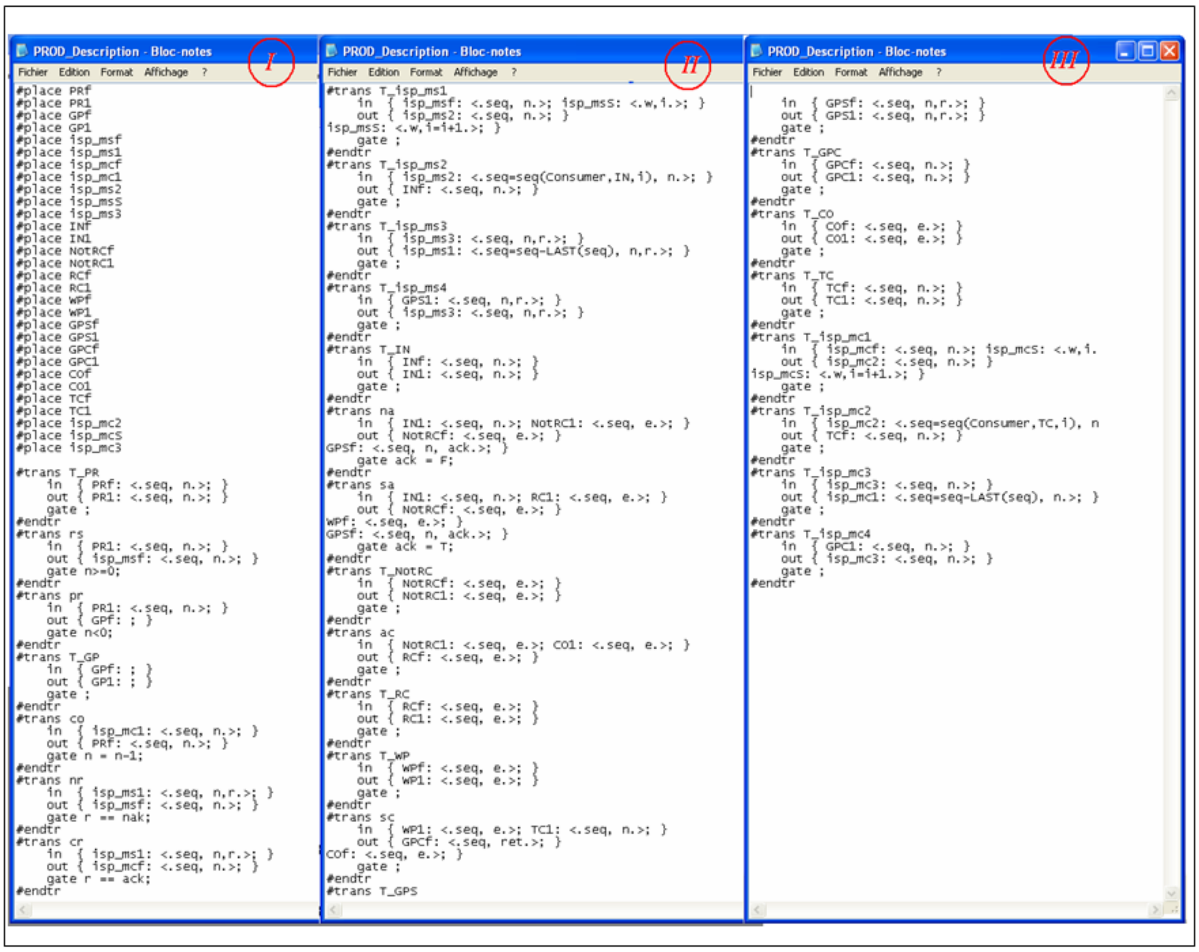


FIGURE 13. The resulted PrT-Nets


PROD_Description - Bloc-notes
Fichier Edition Format Affichage ?
I
II
III

```
#place PRf
#place PR1
#place GPf
#place GP1
#place isp_msf
#place isp_ms1
#place isp_mcf
#place isp_mc1
#place isp_ms2
#place isp_msS
#place isp_ms3
#place INf
#place IN1
#place NotRCf
#place NotRC1
#place RCf
#place RC1
#place WPf
#place WP1
#place GPSf
#place GPS1
#place GPCf
#place GPC1
#place COf
#place CO1
#place TCf
#place TC1
#place isp_mc2
#place isp_mcS
#place isp_mc3

#trans T_PR
     in  { PRf: <.seq, n.>; }
     out { PR1: <.seq, n.>; }
     gate ;
#endtr
#trans rs
     in  { PR1: <.seq, n.>; }
     out { isp_msf: <.seq, n.>; }
     gate n>=0;
#endtr
#trans pr
     in  { PR1: <.seq, n.>; }
     out { GPf: ; }
     gate n<0;
#endtr
#trans T_GP
     in  { GPf: ; }
     out { GP1: ; }
     gate ;
#endtr
#trans co
     in  { isp_mc1: <.seq, n.>; }
     out { PRf: <.seq, n.>; }
     gate n = n-1;
#endtr
#trans nr
     in  { isp_ms1: <.seq, n,r.>; }
     out { isp_msf: <.seq, n.>; }
     gate r == nak;
#endtr
#trans cr
     in  { isp_ms1: <.seq, n,r.>; }
     out { isp_mcf: <.seq, n.>; }
     gate r == ack;
#endtr
```

```
#trans T_isp_ms1
     in  { isp_msf: <.seq, n.>; isp_msS: <.w,i.>; }
     out { isp_ms2: <.seq, n.>; }
isp_msS: <.w,i=i+1.>; }
     gate ;
#endtr
#trans T_isp_ms2
     in  { isp_ms2: <.seq=seq(Consumer,IN,i), n.>; }
     out { INf: <.seq, n.>; }
     gate ;
#endtr
#trans T_isp_ms3
     in  { isp_ms3: <.seq, n,r.>; }
     out { isp_ms1: <.seq=seq-LAST(seq), n,r.>; }
     gate ;
#endtr
#trans T_isp_ms4
     in  { GPS1: <.seq, n,r.>; }
     out { isp_ms3: <.seq, n,r.>; }
     gate ;
#endtr
#trans T_IN
     in  { INf: <.seq, n.>; }
     out { IN1: <.seq, n.>; }
     gate ;
#endtr
#trans na
     in  { IN1: <.seq, n.>; NotRC1: <.seq, e.>; }
     out { NotRCf: <.seq, e.>; }
GPSf: <.seq, n, ack.>; }
     gate ack = F;
#endtr
#trans sa
     in  { IN1: <.seq, n.>; RC1: <.seq, e.>; }
     out { NotRCf: <.seq, e.>; }
WPf: <.seq, e.>; }
GPSf: <.seq, n, ack.>; }
     gate ack = T;
#endtr
#trans T_NotRC
     in  { NotRCf: <.seq, e.>; }
     out { NotRC1: <.seq, e.>; }
     gate ;
#endtr
#trans ac
     in  { NotRC1: <.seq, e.>; CO1: <.seq, e.>; }
     out { RCf: <.seq, e.>; }
     gate ;
#endtr
#trans T_RC
     in  { RCf: <.seq, e.>; }
     out { RC1: <.seq, e.>; }
     gate ;
#endtr
#trans T_WP
     in  { WPf: <.seq, e.>; }
     out { WP1: <.seq, e.>; }
     gate ;
#endtr
#trans sc
     in  { WP1: <.seq, e.>; TC1: <.seq, n.>; }
     out { GPCf: <.seq, ret.>; }
COf: <.seq, e.>; }
     gate ;
#endtr
#trans T_GPS
```

```
     in  { GPSf: <.seq, n,r.>; }
     out { GPS1: <.seq, n,r.>; }
     gate ;
#endtr
#trans T_GPC
     in  { GPCf: <.seq, n.>; }
     out { GPC1: <.seq, n.>; }
     gate ;
#endtr
#trans T_CO
     in  { COf: <.seq, e.>; }
     out { CO1: <.seq, e.>; }
     gate ;
#endtr
#trans T_TC
     in  { TCf: <.seq, n.>; }
     out { TC1: <.seq, n.>; }
     gate ;
#endtr
#trans T_isp_mc1
     in  { isp_mcf: <.seq, n.>; isp_mcS: <.w,i.
     out { isp_mc2: <.seq, n.>; }
isp_mcS: <.w,i=i+1.>; }
     gate ;
#endtr
#trans T_isp_mc2
     in  { isp_mc2: <.seq=seq(Consumer,TC,i), n
     out { TCf: <.seq, n.>; }
     gate ;
#endtr
#trans T_isp_mc3
     in  { isp_mc3: <.seq, n.>; }
     out { isp_mc1: <.seq=seq-LAST(seq), n.>; }
     gate ;
#endtr
#trans T_isp_mc4
     in  { GPC1: <.seq, n.>; }
     out { isp_mc3: <.seq, n.>; }
     gate ;
#endtr
```

FIGURE 14. Generated PROD description of Producer/Consumer problem

and verification of software information systems using G-Nets formalism. With meta-modeling, we have defined the syntactic aspect of both G-Nets and PrT-Nets formalisms. Then, we have used AToM^3 to generate their visual modeling environment. By means of graph grammar, we have extended the capabilities of our framework to transform G-Nets specification into their equivalent PrT-Nets Models and to generate the PROD description of the resulted PrT-Nets Models for analysis purpose. For the future work, we plan to include the verification phase using PROD on a more complicated example and to provide our tool by giving a feedback of the results (interpreting the results) of this verification in the initial G-Nets model. This work is an important step in a large project aiming at using graph transformation to formalize UML diagrams using G-Nets. We have chosen G-Nets to formalize UML because with G-Nets formalism we can specify several aspects of system at a high level of abstraction and in Object Oriented modular paradigm.

REFERENCES

[1] R. Fairley, *Software Engineering Concepts*, MacGraw-Hill, New York, NJ, USA.
[2] H. J. Genrich and K. Lautenbach, System modelling with high-level Petri nets, *Proc. of Theoretical Computer Science*, vol.13, no.1, pp.109-135, 1981.
[3] Y. Deng, S. K. Chang, J. De Figueired and A. Psrkusich, Integrating software engineering methods and Petri nets for the specification and prototyping of complex information systems, *Proc. of the 14th International Conference on Application and Theory of Petri Nets*, Chicago, pp.206-223, 1993.
[4] *PROD Tool*, http://www.tcs.hut.fi/Software/prod/.

[5] J. De Lara and H. Vangheluwe, Meta-modelling and graph grammars for multi-paradigm modelling in AToM^3, *Software and Systems Modelling*, vol.3, pp.194-209, 2004.

[6] G. Rozenberg, *Handbook of Graph Grammars and Computing by Graph Transformation*, World Scientific, 1999.

[7] *AToM^3: A Tool for Multi-Formalism and Meta-Modelling*, McGill University, Canada, http://atom3.cs.mcgill.ca/.

[8] M. Baldassari, G. Bruno, V. Russi and R. Zompi, PROTOB a hierarchical objectoriented CASE tool for distributed systems, *Lecture Notes in Computer Science*, vol.387, pp.424-445, 1989.

[9] E. Battiston and F. de Cindio, Class orientation and inheritance in modular algebraic nets, *Proc. of IEEE International Conference on Systems, Man and Cybernetics*, Le Touquet, France, pp.717-723, 1993.

[10] E. Battiston, F. de Cindio and G. Mauri, OBJSA nets: A class of high-level nets having objects as domains, *Advances on Petri Nets 1988*, vol.340, pp.20-43, 1988.

[11] Buchs and N. Guelfi, CO-OPN: A concurrent object oriented Petri net approach, *Proc. of the 12th International Conference on the Application and Theory of Petri Nets*, Aarhus, Denmark, 1991.

[12] J. Engelfriet, G. Leih and G Rozenberg, Net-based description of parallel objectbased systems, or POTs and POPs, in *Foundations of Object-Oriented Languages, Lecture Notes in Computer Science*, J. W. de Bakker, W. P. de Roever and G. Rozenberg (eds.), Springer-Verlag, 1990.

[13] C. Lakos, Pragmatic inheritance issues for object Petri nets, *Sem Maiores Indicações*, 1995.

[14] D. S. Guerrero, J. C. A de Figueiredo and A. Perkusich, An object-based modular CPN approach: Its application to the specification of a cooperative editing environment, *Concurrent Object-Oriented Programming and Petri Nets, Advances in Petri Nets, Lecture Notes in Computer Science*, pp.338-354, 2001.

[15] *GME: Generic Modelling Environment*, http://www.isis.vanderbilt.edu/gme/.

[16] S. Kelly, K. Lyytinen and M. Rossi, MetaEdit+: A fully configurable multi-user and multi-tool CASE and CAME environment, *Proc. of Advanced Information System Engineering*, Berlin, 1996.

[17] *EMF: Eclipse Modelling Framework*, http://www.eclipse.org/emf/.

[18] *GMF: Graphical Modelling Framework*, http://www.eclipse.org/gmf/.

[19] *PROGRES: A Graph Grammar Programming Environment*, http://www-i3.informatik.rwth-aachen.de/research/projects/progres/.

[20] *GReAT: Graph Rewrite And Transformation*, http://www.escherinstitute.org/Plone/tools/.

[21] *FUJABA: From UML to Java and Back Again*, http://www.fujaba.de/.

[22] *AGG: The Attributed Graph Grammar System*, http://tfs.cs.tu-berlin.de/agg/.

[23] J. De Lara and H. Vangheluwe, AToM^3: A tool for multi-formalism modelling and meta-modelling, *Proc. of Fundamental Approaches to Software Engineering*, Grenoble, France, vol.2306, pp.174-188, 2002.

[24] J. De Lara and H. Vangheluwe, Computer aided multi-paradigm modelling to process petri-nets and statecharts, *Proc. of International Conference on Graph Transformations*, Barcelona, vol.2505, pp.239-253, 2002.

[25] E. Kerkouche, A. Chaoui, E. Bourennane and O. Labbani, On the use of graph transformation in the modeling and verification of dynamic behavior in UML models, *Journal of Software*, vol.5, no.11, pp.1279-1291, 2010.

[26] E. Kerkouche and A. Chaoui, A formal framework and a tool for the specification and analysis of G-Nets models based on graph transformation, *Proc. of International Conference on Distributed Computing and Networking*, India, pp.206-211, vol.5408, 2009.

[27] A. Perkusich and J. De Figueiredo, G-Nets: A Petri net based approach for logical and timing analysis of complex software systems, *Journal of Systems and Software*, vol.39, pp.39-59, 1997.

[28] Y. Deng, *A Unified Framework for the Modelling, Prototyping and Design of Distributed Information Systems*, Ph.D. Thesis, University of Pittsburgh, 1992.

[29] *Python Programming Language*, http://www.python.org.