International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:2, No:8, 2008

A Graphical Environment for Petri Nets INA Tool Based on Meta-Modelling and Graph Grammars

Raida El Mansouri, Elhillali Kerkouche, and Allaoua Chaoui

***Abstract*—**The Petri net tool INA is a well known tool by the Petri net community. However, it lacks a graphical environment to cerate and analyse INA models. Building a modelling tool for the design and analysis from scratch (for INA tool for example) is generally a prohibitive task. Meta-Modelling approach is useful to deal with such problems since it allows the modelling of the formalisms themselves. In this paper, we propose an approach based on the combined use of Meta-modelling and Graph Grammars to automatically generate a visual modelling tool for INA for analysis purposes. In our approach, the UML Class diagram formalism is used to define a meta-model of INA models. The meta-modelling tool ATOM3 is used to generate a visual modelling tool according to the proposed INA meta-model. We have also proposed a graph grammar to automatically generate INA description of the graphically specified Petri net models. This allows the user to avoid the errors when this description is done manually. Then the INA tool is used to perform the simulation and the analysis of the resulted INA description. Our environment is illustrated through an example.

***Keywords*—**INA, Meta-modelling, Graph Grammars, AToM3, Automatic Code Generation.

I. Introduction

THE Petri nets INA (Integrated Net Analyser) tool was developed by Prof. Dr. Peter H. Starke [12]. It is an interactive menu-driven program that allows a user to edit (in a textual form), reduce, execute and analyze Petri nets models. One of the advantages of Petri nets is their graphical representations of the models. However, INA tool is not graphical and the user has to transform the graphical representation to a textual description manually. So there is a risk of errors during the transformation process. The cost of building a visual tool for INA from the scratch is prohibitive. Meta-Modelling approach is useful to deal with such problems, as it allows (possibly graphical) the modelling of the formalisms themselves [8]. A model of formalism should contain enough information to permit the automatic generation of a tool to check and build models subject to the described formalism's syntax. If this specification is done graphically, the time to develop a modelling tool can be drastically reduced to a few hours.

Since meta-model and model are stored as graphs, further manipulations of the models can be described graphically and formally as graph grammars [13]. Some of these manipulations are model simulation or animation, model optimisation, for example, to reduce its complexity, model transformation into another model (equivalent in behaviour), expressed in a different formalism, and the generation of textual model representations for use by existing simulators or tools. In this paper we will focus on the last kind of model transformation. These ideas presented above are implemented in ATOM^3: A Tool for Multi-formalism and Meta-Modelling [2].

In this paper, we propose an INA Petri net meta-model and we use the meta-modelling tool AToM^3 to generate automatically a visual modelling tool to process models in INA formalism. We also define a graph grammar to translate the models created in the generated tool to a textual description in INA language (INA specification). Then the tool INA is used to perform the analysis of the resulted INA specification.

This paper is organized as follows: section II outlines some related work. In section III, we give an example of a graphical representation of a Petri nets model, its specification in INA, and justify the need for an automatic translator from the graphical representation to the INA specification. In section IV, we recall some concepts about Graph Grammars and AToM^3 tool. In section V, we define a meta-model for INA Petri net models and generate a visual tool for this formalism. In section VI, we propose a graph grammar to generate INA specification of models created with our tool. In section VII, we illustrate our tool through an example. Finally, section VIII concludes the paper.

II. Related Work

The AToM^3 has been proven to be a very powerful tool allowing the meta-modeling and the transformations between formalisms. In [5] the authors proposed a transformation of non deterministic finite state automata to their equivalent deterministic finite state automata. In [6] the authors presented a transformation between Statecharts (without hierarchy) and Petri Nets. In [4)] a transformation between Statecharts and DEVS is given. In [7] the authors used meta-modeling and

Raida El Mansouri is with the Department of Computer Science, Faculty of Engineering, University Mentouri Constantine, Algeria (e-mail: raidaelmansouri@ yahoo.fr)..

Elhillali Kerkouche is with the Department of Computer Science, University of Oum Elbouaghi, Algeria (e-mail: elhillalik@ yahoo.fr).

Allaoua Chaoui is with the Department of Computer Science, Faculty of Engineering, University Mentouri Constantine, Algeria (e-mail: a_chaoui2001@ yahoo.com).

graph grammars to process GPSS models. The processing of UML Class Diagrams, Activity Diagrams, and many others using graph transformation can be found in [9] and [2]. In UML Activity Diagram for example, the authors were defined a graph grammar to transform UML Activity Diagram models into theirs equivalent Petri Nets models. Whereas in GPSS, the authors were defined a graph grammar to generate textual code for the HGPSS simulator from GPSS models. In this paper we propose a framework (a tool) based on the combined use of Meta-Modeling and Graph Grammars to generate a graphical environment for INA tool allowing the user to create the graphical representation of a Petri net model and then generate automatically its equivalent INA specification.

III. INA Specification

Consider the following graphical representation of a Petri net model for three programmers and two terminals [12].

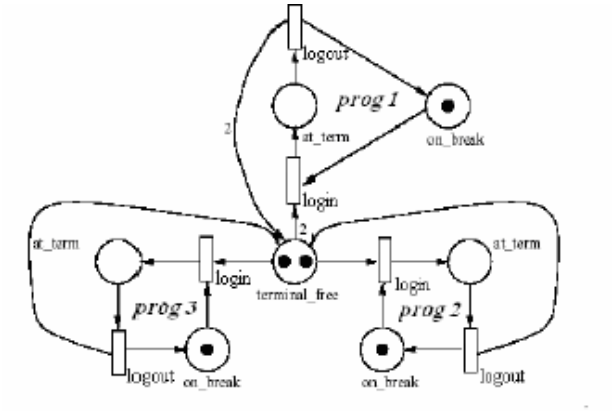


Fig. 1 Three programmers sharing two terminals

To perform analysis using INA tool, the graphical representation must be mapped to its equivalent INA specification. The textual description for INA tool (INA specification) of the above graphical representation is given in the file 3_prog_2_term.pnt as follows:

```
P  M    PRE,POST        NETZ 1:3_prog_2_term
 0 2       4: 2 5 6, 1: 2 2 3
 1 0       1, 4
 2 0       2, 5
 3 0       3, 6
 4 1       4, 1
 5 1       5, 2
 6 1       6, 3
@
place nr.                    name capacity  time
      0: terminal_free              00      0
      1: prog1_at_term              00      0
      2: prog2_at_term              00      0
      3: prog3_at_term              00      0
      4: prog1_on_break             00      0
      5: prog2_on_break             00      0
      6: prog3_on_break             00      0
@
trans nr.                    name priority time
      1: login_prog1                0      0
      2: login_prog2                0      0
      3: login_prog3                0      0
      4: logout_prog1               0      0
      5: logout_prog2               0      0
      6: logout_prog3               0      0
@
```

Fig. 2 The INA specification of the graphical representation of the net given in Fig. 1

When this INA specification is performed manually there is a risk of errors. So an automatic mapping from a graphical representation to INA specification will be welcome. So, we recall that our aim is to provide a user with a graphical tool for INA allowing it to create a graphical Petri net model (for example the model of Fig. 1) and then generate automatically the textual description for INA tool (for example the description of Fig. 2). The INA tool can be called for analysis purposes.

IV. Graph Grammars and ATOM3

We recall in the following subsections some basic notions about graph grammars and AToM^3.

A. Graph Grammars

Graph grammar [1] [13] is a generalization of Chomsky grammar for graphs. It is a formalism in which the transformation of graph structures can be modelled and studied. The main idea of graph transformation is the rule-based modification of graphs as shown in Fig. 2.

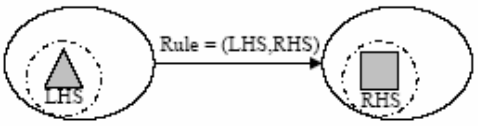


Fig. 3 Rule-based Transformation of Graphs

Graph grammars are composed of production rules; each having graphs in their left hand side (LHS) and right hand side (RHS). Rules are compared with an input graph called host graph. If a matching is found between the LHS of a rule and a subgraph in the host graph, then the rule can be applied and the matching subgraph of the host graph is replaced by the RHS of the rule. A rewriting system iteratively applies

matching rules in the grammar on the host graph until no more rules are applicable.

B. AToM^3

AToM^3 is a visual tool for multi-formalism modelling and meta-modelling. The two main tasks of AToM^3 are meta-modelling and model transformation. For *meta-modelling*, AToM^3 supports visual modelling using Entity Relationship (ER) formalism or UML Class Diagram formalism, which means that in AToM^3, we can use either ER model or UML Class Diagram model to meta-model the new formalisms of interest. To be able to fully specify modelling formalisms, the meta-formalism may be extended with the ability to express constraints (which cannot be expressed within ER or UML Class Diagram alone). Constraints provide a view on how a construct can be connected to another to be meaningful, and thus specify static semantics of the formalism. Whereas the meta-modelling formalism frequently uses a graphical notation, constraints are concisely expressed in textual form. For this purpose, some systems, including AToM^3 use the Object Constraint Language OCL used in the UML. As AToM^3 is implemented in the scripting language Python, arbitrary Python code may be also used. Once we build the meta-models for the interested models, AToM^3 can generate automatically a visual modelling environment, in which you can build and edit the new models.

For *model transformation*, AToM^3 supports graph rewriting, which uses graph Grammar rules to visually guide the procedure of the transformation (see section 4). The rules are specified by the user, and the rules are ordered according to certain criteria depending on the features of the model to be transformed. Expressing computations in the form of graph grammars has some advantages over an implicit representation (embedding the transformation computation in a program using a traditional programming language) [3]. The main advantages can be summarized as follows:

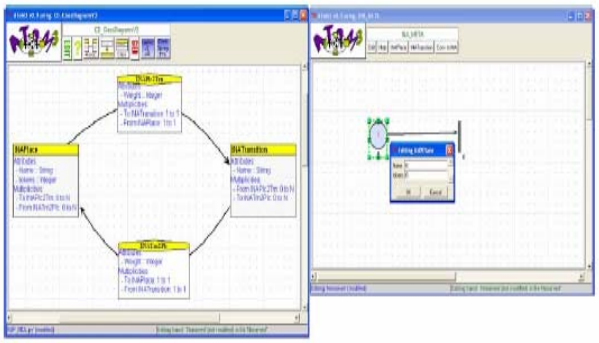
- It is an abstract, declarative, high level representation of the computation. This enables exchange, re-use, and symbolic analysis of the transformation model.
- The theoretical foundations of graph rewriting systems may assist in proving correctness and convergence properties of the transformation tool.

V. Meta-Modeling of INA Petri Nets

To build models of INA Petri nets formalism in AToM^3, we have to define a meta-model for the Petri nets formalism. The meta-formalism used in our work is the UML Class Diagram and the constraints are expressed in Python code [11].

Since INA Petri nets models consist of places, transitions, and arcs from places to transitions and from transitions to places, we have proposed to meta-model INA Petri nets model two Classes to describe Places and Transitions, and two associations for Input Arcs and Output Arcs as shown in Fig. 3. We have also specified the visual representation of each class or association according to the notation presented in Fig. 1.

Given our meta-model, we have used AToM^3 tool to generate a graphical environment for INA Petri nets models. Figure 4 shows the generated INA graphical environment and a dialog box to edit a place. Each place has two attributes (*name* and *initial marking*) which are defined in the proposed Meta-model (see Fig. 3 in INAPlace class).



(a) (b)

Fig. 4 (a) INA PN Model (b) INA Tool

VI. Generation of INA Specification

In order to analyse INA models, it is necessary to translate these models into their equivalent representations in INA specification. In this section we show how to use the modelling environment obtained in the previous section to generate INA specification. We do this by defining a Graph Grammar to traverse the INA model and generate the corresponding code in INA. The advantage of using a graph grammar to generate the textual code is the graphical and high-level fashion. The graph grammar has an *initial Action* which opens the file where the code will be generated and decorates all the *Transition* and *Place* elements in the model with temporary attributes to be used in the conditions specified in 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 generated yet. In Place elements, we use also two attributes: *fromVisited* and *toVisited*. The *fromVisited* attribute is used to indicate whether this place is processed as input place whereas the *toVisited* attribute is used to indicate if this place is processed as output place.

In our graph grammar, we have proposed *seven rules* which will be applied in ascending order by the rewriting system until no more rules are applicable. We are concerned here by automatic code generation, so none of these rules will change the INA models (i.e, in each of the six rules, the LHS is the same as the RHS). These rules are shown in *figure 5* and described as follows:

Rule1: genLHS_rl(priority 1): is applied to locate a place (not previously processed) which is related to current

transition with an input arc, and generate the corresponding INA specification.

Rule2: betweenLHSandRHS(priority 2): is applied to generate Maude code which separates LHS and RHS of the equivalent rewriting rule.

Rule3: genRHS_rl(priority 3): is applied to locate a place (not previously processed) which is related to current transition with a output arc, and generate the corresponding Maude specification.

Rule4: genTC(priority 4): is applied to generate the appropriate Maude syntax depending on the TC of the transition, and mark the transition as visited.

Rule5: InitialisePlace(priority 5):This rule is applied to locate and initialise temporary attributes in places for processing the next transition.

Rule6: SelectTransition(priority 6): is applied to select a transition that has not been previously processed to generate its equivalent in INA specification.

Rule7: SelectTransition(priority 7):

The graph grammar has also a final action which erases the temporary attributes from the entities and closes the output file. Finally, we have assigned the execution of this graph grammar to a button labelled as "*Generate* INA specification" in Fig. 4.

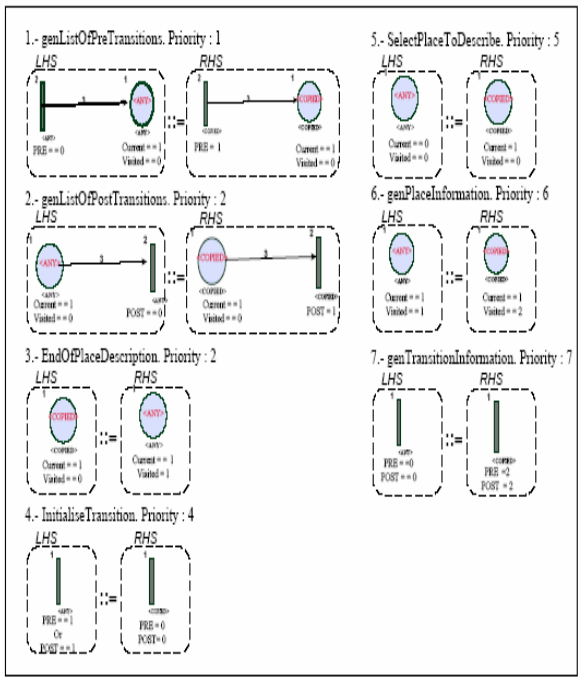


Fig. 5 Graph Grammar to generate INA specification from the graphical representation

VII. Case Study

In this section, we reconsider the same example presented in section 3 to illustrate our approach. We present first the graphical model describing this example via the generated tool. Thereafter, the translating of this graphical model into its equivalent INA specification using proposed graph grammar will be shown.

A. Example Presentation

This example is about three programmers sharing two terminals. Each programmer is either working at a terminal or pausing. Programmer 1 needs two terminals at the same time in order to work. The Fig. 6 presents the graphical model of this problem created in our tool.

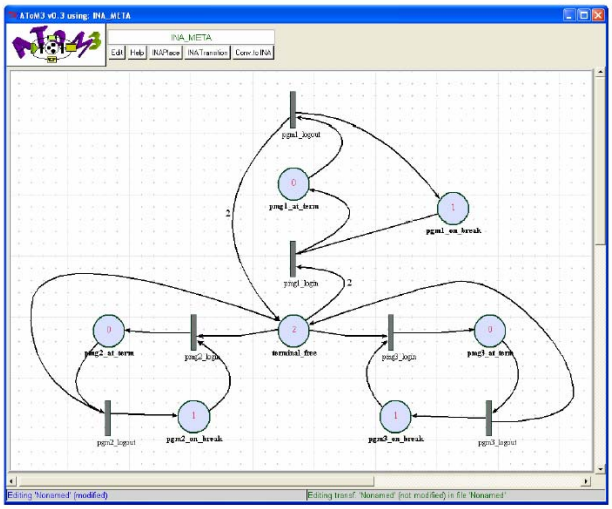


Fig. 6 3_prog_2_term problem created in our tool

B. Translating INA Petri Nets Model to INA Specification

In order to translate the graphical representation of the 3_prog_2_term problem into INA specification using the graph grammar defined in previous section, one have to click on the "Conv.to.INA" button in the interface of the generated tool. The result of this translation is the file (3_prog_2-term.pnt) which contains the description of the net and the initial marking in INA syntax as shown in Fig. 7.

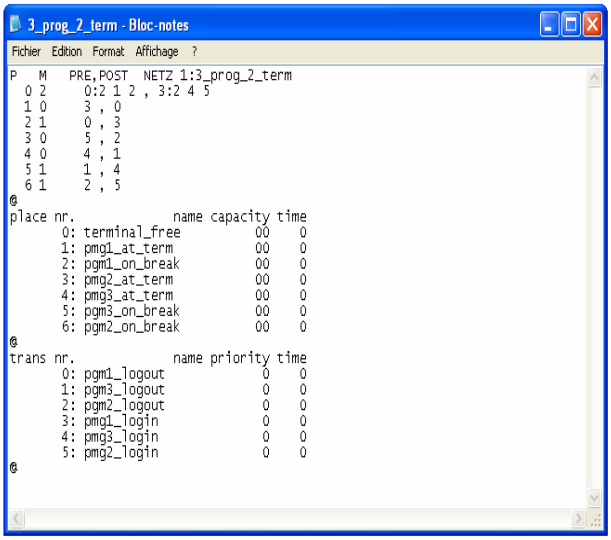


Fig. 7 Generated INA specification of 3_prog_2_term

VIII. Conclusion

In this paper, we have proposed an approach based on combining Meta-modelling and Graph Grammars to automatically generate a visual modelling tool for ECATNets for simulation and analysis purposes. ECATNets are a category of algebraic Petri Nets based on a safe combination of algebraic abstract types and high level Petri Nets. ECATNets' semantic are defined in terms of rewriting logic allowing us to built models by formal reasoning. The cost of building a visual modelling tool (for ECATNets for example) from scratch is prohibitive. We have demonstrated in this work that Meta-Modelling approach is useful to deal with this problem since it allows the modelling of the formalisms themselves. By means of Graph Grammars, models manipulations are expressed on a formal basis and in a graphical way. In our approach, the UML Class diagram formalism is used as meta-formalism to propose a meta-model of ECATNets. The meta-modelling tool ATOM^3 is used it to generate a visual modelling tool according to the proposed ECATNets meta-model. We have also proposed a graph grammar to generate Maude description of the graphically specified ECATNets models. Then the rewriting logic language Maude is used to perform the simulation of the resulted Maude specification.

In a future work, we are planning to hide the steps of the Simulation. The objective of this hiding is to unburden the user from having to manually invoke Maude language and to manipulate the textual version of the result of simulation. For this purpose, the result of simulation (final state) will be returned in graphically way in ECATNets model structure.

References

[1] AGG Home page: http://tfs.cs.tu-berlin.de/agg/
[2] Home page: http://atom3.cs.mcgill.ca/
[3] Bardohl, R., H. Ehrig, J. De Lara and G. Taentzer (2004). "Integrating Meta Modelling with Graph Transformation for Efficient Visual Language Definition and Model Manipulation". Lecture Notes in Computer Science 2984, pp.: 214-228.
[4] Borland, S., Vangheluwe, H (2003): Transforming Statecharts to DEVS. A. Bruzzone and Mhamed Itmi, editors, Summer Computer Simulation Conference, Student Workshop, pp. S154-- S159, Society for Computer Simulation International (SCS), Montréal, Canada (2003).
[5] De Lara, J., Vangheluwe, H (2002): AToM3: A Tool for Multi-Formalism Modelling and Meta-Modelling. Lecture Notes in Computer Science 2306, pp.174--188. Presented also at Fundamental Approaches to Software Engineering - FASE'02 , in European Joint Conferences on Theory And Practice of Software - ETAPS'02, Grenoble, France(2002).
[6] De Lara, J., Vangheluwe, H.(2002): Computer aided multi-paradigm modelling to process petri-nets and statecharts. In International Conference on Graph Transformations (ICGT), Lecture Notes in Computer Science, vol. 2505, pp. 239--253, Springer-Verlag, Barcelona, Spain(2002).
[7] De Lara, J., Vangheluwe, H. (2002): Using meta-modelling and graph grammars to process GPSS models. Hermann Meuth, editor, 16th European Simulation Multi-conference (ESM), pp. 100--107, Society for Computer Simulation International (SCS), Darmstadt, Germany (2002).
[8] De Lara, J and H. Vangheluwe (2004). "Meta-Modelling and Graph Grammars for Multi-Paradigm Modelling in AToM3". Manuel Alfonseca. Software and Systems Modelling, Vol 3(3), pp.: 194-209. Springer-Verlag. Special Section on Graph Transformations and Visual Modeling Techniques.
[9] De Lara, J., Vangheluwe, H. (2005): Model-Based Development: Meta-Modelling, Transformation and Verification, The Idea Group Inc, pp. 17 (2005)
[10] DOME (1999). Home page: http://www.htc.honeywell.com/dome/
[11] Python home page: htpp://www.python.org
[12] Roch S. and Starke P.H. (2002). Integrated Net Analyze, User manual, 2002.
[13] "Handbook of Graph Grammars and Computing by Graph Transformation". Vol.1 World Scientific.

Raida Elmansouri is Assistant Professor in the department of Computer science, Faculty of Engineering, University Mentouri Constantine, Algeria. Her research field is formal methods and Information Systems.

Elhillali Kerkouche is Assistant Professor in the department of Computer science, University of Oum Ebouaghi, Algeria. His research field is formal methods and Distributed Systems.

Allaoua Chaoui is with the department of computer science, Faculty of Engineering, University Mentouri Constantine, Algeria. He received his Master degree in Computer science in 1992 (in cooperation with the University of Glasgow, Scotland) and his PhD degree in 1998 from the University of Constantine (in cooperation with the CEDRIC Laboratory of CNAM in Paris, France). He has served as associate professor in Philadelphia University in Jordan for five years and University Mentoury Constantine for many years. During his career he has designed and taught courses in Software Engineering and Formal Methods. Dr Allaoua Chaoui has published many articles in International Journals and Conferences. He supervises many Master and PhD students. His research interests include Mobile Computing, formal specification and verification of distributed systems, and graph transformation systems.