A Graphical Tool Support to Process and Simulate ECATNets Models based on Meta-Modelling and Graph Grammars

Elhillali KERKOUCHE[1] and Allaoua CHAOUI[2]

[1]Department of Computer Science,
University of Oum El Bouaghi, Algeria
elhillalik@yahoo.fr

[2]MISC Laboratory,
Department of Computer Science,
University Mentouri Constantine, Algeria
a_chaoui2001@yahoo.com

Abstract. ECATNets are an algebraic Petri net category 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. Furthermore, the rewriting logic language Maude gives to ECATNEts dynamic aspects which are not measurable without simulation. The building of a modelling tool for the design and analysis from scratch (for ECATNets for example) is generally prohibitive task. Meta-Modelling approach is useful to deal with this problem, as it allows (possibly is done graphically) the modelling of the formalisms themselves. Since meta-model and model are graphs, further manipulations –simulation, transformation and code generation for an existing solver– of the models can be described graphically and formally as graph grammar. 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 ECATNets for analysis and simulation purposes. In our approach, the UML Class diagram formalism is used to define a meta-model of ECATNets. The meta-modelling tool ATOM^3 is used 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. Our approach is illustrated through an example.

Keywords: *ECATNets, Rewriting Logic, Maude, Meta-modelling, Graph Grammars, Graph Transformations, AToM^3, Automatic Code Generation.*

(Received September 02, 2009 / Accepted January 06, 2010)

1. Introduction

ECATNets are an algebraic Petri net category based on a safe combination of algebraic abstract types and high level Petri Nets [5]. In addition to modelling, ECATNets allow the verification and simulation of concurrent systems [4]. The most distinctive feature of ECATNets is that their semantic are defined in terms of rewriting logic [21], allowing us to build models by formal reasoning. The rewriting logic Maude [8] is considered as one of very powerful languages in the specification and verification of concurrent systems [21]. Rewriting logic gives to ECATNets a simple, more intuitive and practical version to analyse, without loosing formal semantic (mathematical rigor, formal reasoning). Furthermore, high level abstraction of this logic makes ECATNets, in spite of their complexity, to be dealt as simple as possible. The power of Maude in terms of specification, programming, simulation and verification in addition to the ECATNets' integration in Maude, implies that there is no need to translate ECATNets in several languages and thus any risks about their semantic loss [7]. On the other hand, the use of the rewriting logic language Maude is constrained by the textual way to create and deal with ECATNets system.

Execution under Maude system is done by using command prompt style. In this case, we loose the graphical aspect of ECATNets formalism which is important for the clarity, simplicity and readability of a system description.

The cost of building a modelling tool from the scratch is prohibitive. Meta-Modelling approach is useful to deal with this problem, as it allows (possibly graphical) the modelling of the formalisms themselves [11]. 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 [24]. 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 ECATNets meta-model and we use the meta-modelling tool AToM^3 to generate automatically a visual modelling tool to process models in ECATNets formalism. We also define a graph grammar to translate the models created in the generated tool to a Maude specification. Then the rewriting logic language Maude is used to perform the simulation of the resulted Maude specification.

The rest of this paper is organized as follows: Section 2 outlines the major related work. We give a brief introduction on ECATNets formalism and their integration in rewriting logic in section 3. In section 4, we recall some concepts about Graph Grammars. In section 5, we give an overview of the AToM^3 tool. In section 6, we define a meta-model for ECATNets and we generate a visual tool for this formalism. In section 7, we propose a graph grammar to generate Maude specification of models created with our tool. In section 8, we illustrate our tool with Router problem. First, we have created the ECATNets model for this problem. Then we have generated Maude specification of the model and invoked the rewriting logic language Maude to perform the simulation. Finally, section 9 concludes the paper.

2. Related Work

In addition to AToM^3, there are several visual tools to describe formalisms using metamodeling like Generic Modeling Environment (GME) [15], MetaEdit+ [18] and other tools from the Eclipse Generative Modeling Tools (GMT) project such as Eclipse Modeling Framework (EMF) [12], Graphical Editing Framework (GEF) [14] and Graphical Modeling Framework (GMF) [16]. In most of these tools, model transformations have to be described textually and user friendly support for visual analysis and testing is generally missing. In AToM^3, the user expresses such transformations by means of graph grammar models. Graph grammars are a natural, declarative, and general way to express transformations.

There are also similar tools which manipulate models by means of graph grammars, such as PROGRES [22], GReAT [17], FUJABA [13], TIGER [25] and AGG [1]. However, none of these has its own meta-modeling layer. Some of them are complemented with support for meta-modelling (for example, The GReAT model transformation engine is combined with GME).

The combined use of meta-modelling and graph grammars taken in AToM^3 allow users not only to benefit from the advantages of both (meta-modelling and graph grammars) but also to model with multi-paradigm modeling [9]. The AToM^3 tool has been proven to be very powerful, allowing the meta-modeling and the transformations of known formalisms. In [10] the authors presented a transformation between Statecharts (without hierarchy) and Petri Nets. In [19], the authors have presented a formal framework (a tool) based on the combined use of Meta-Modeling and Graph Grammars for the specification and the analysis of complex software systems using G-Nets formalism. The framework allows a developer to draw a G-Nets model and transform it into its equivalent PrT-nets model automatically. In order to perform the analysis using PROD analyzer, the framework allows a developer to translate automatically each resulted PrT-Nets model into PROD's net description language. In [20] the authors have proposed an approach for transforming UML Statechart and collaboration diagrams to colored Petri nets models. More precisely, they have proposed an automated approach and a tool environment that formally transforms dynamic behaviours of systems expressed using UML models into their equivalent colored Petri Nets (CPN) models for analysis purpose. This transformation aimed to bridge the gap between informal notation (UML diagrams) and more formal notation (coloured Petri nets models). It produces highly-structured, graphical, and rigorously-analyzable models that facilitates early detection of errors like deadlock, livelock, … .To make the analysis more easy, they have used the obtained CPN models to generate automatically their equivalent description in the input language of INA Petri net tool.

3. ECATNets

ECATNets [5] are a kind of net/data model combining the strengths of Petri Nets with those of abstract data types. The most distinctive feature of ECATNets is that their semantic is defined in terms of rewriting logics [21]. Motivating ECATNets (Extended Concurrent Algebraic Terms Nets) leads to motivating Petri Nets, abstract data types, as well as their combination into a unified framework [7].

From a syntactic point of view, places are marked with multi-sets of algebraic terms. Input arcs of each transition t, i.e. (p,t), are labeled by two inscriptions IC(p,t) (Input Condition) and DT(p,t) (Destroyed Tokens), output arcs of each transition t, i.e. (t,p'), are labelled by CT(t,p') (Created Tokens), and finally each transition t is labelled by TC(t) (Transition Condition). IC(p,t) specifies the enabling condition of the transition t, DT(p,t) specifies the tokens (a multi-set) which have to be removed from p when t is fired, CT(t,p') specifies the tokens (a multi-set) which have to be added to p' when t is fired. Finally, TC(t) represents a Boolean term which specifies an additional enabling condition for the transition t. the current ECATNets state is given by the union of terms having the following form (p,M(p)).

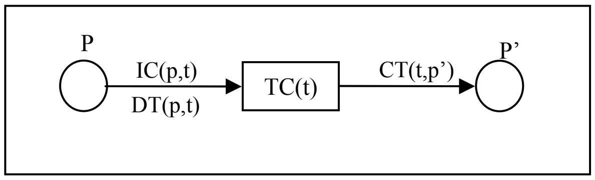


Figure 1. A generic ECATNets

The ECATNets behaviour may be informally commented in the following way. A transition t is enabled when various conditions are simultaneously true. The first condition is that every IC(p,t) for each input place p is enabled. The second condition is that TC(t) is true. Finally the addition of CT(t,p') to each output place p' must not result in p' exceeding its capacity when this capacity is finite. When t is fired, DT(p,t) is removed from the input place p and simultaneously CT(t,p') is added to the output place p'. Transition firing and its conditions are formally expressed by rewriting rules [7].

4. Graph Grammars: an introduction

Graph grammar [24] 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 Figure 2.

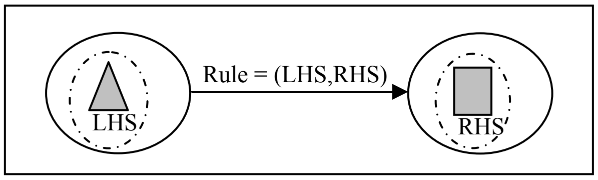


Figure 2. Rule-based Modification of Graphs

Graph grammars are composed of production rules; each having graphs in their left and right hand sides (LHS and RHS). 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 to the host graph until no more rules are applicable.

5. AToM^3: An Overview

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.

In the next sections, we will discuss how we use AToM^3 to meta-model ECATNets formalism, how to generate the ECATNets visual modelling environment, and how to convert models in ECATNets formalism to their equivalent description in Maude for the simulation purpose.

6. Meta-Modelling of ECATNets

To build models of ECATNets formalism in AToM^3, we have to define a meta-model for ECATNets. The meta-formalism used in our work is the UML Class Diagrams and the constraints are expressed in Python code [23].

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

Given our meta-model, we have used AToM^3 tool to generate a visual modelling environment for ECATNets models. Figure 4 shows the generated ECATNets tool 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 Figure 3 in ECATNetsPlace class).

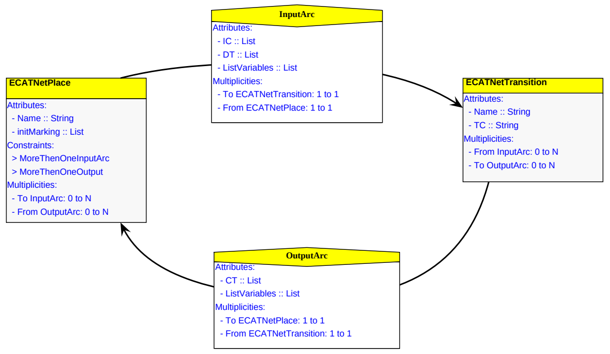
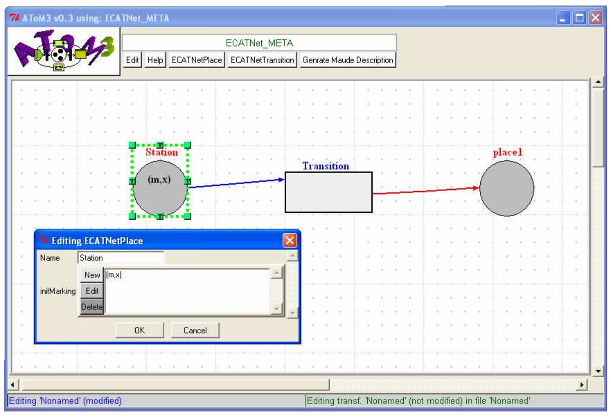


Figure 3. ECATNets Meta-Model



Name Station
initMarking (m,x)

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

Figure 4. Generated tool to process ECATNets models

7. Generation of Maude Specification

In order to simulate ECATNets models, it is necessary to translate these models into their equivalent representations in Maude syntax. In this section we show how to use the modelling environment generated in the previous section to generate Maude specification. We do this by defining a Graph Grammar to traverse the ECATNets model and generate the corresponding code in Maude. 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 *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 ECATNets models. 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 Maude 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): is applied to locate and initialise temporary attributes in places for processing the next transition .

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

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 Maude Description*" in Figure 4.

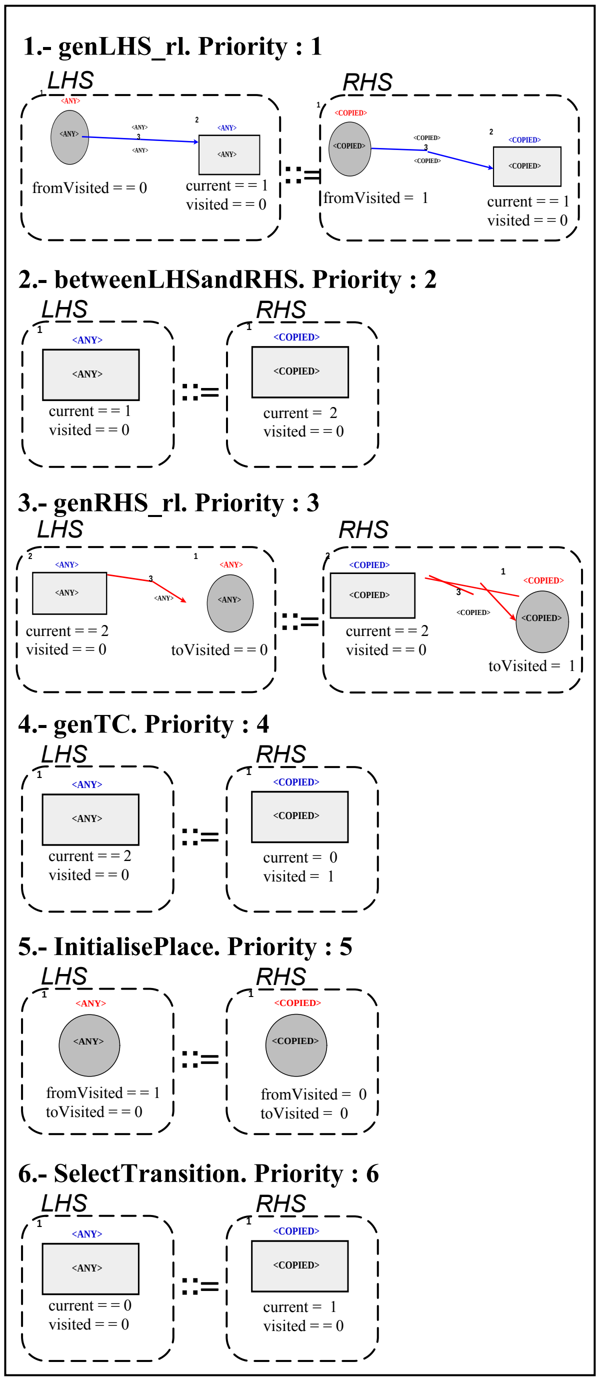


Figure5. Graphs Graph Grammar to generate Maude specification from an ECATNets model

8. Steps of ECATNets Simulator: router problem example

In this section, we describe the most principal steps of ECATNets Simulator trough an example presented in [6] about communication network that relies messages senders to receivers. We present first the ECATNets model describing this example via the generated tool. Thereafter, the translating of this model into its equivalent Maude specification using proposed graph grammar will be shown. Finally, the simulation of this example under Maude system will be given.

8.1. Example presentation

This example is about a network of communication that joins three messages senders to three receivers. Every sender (respectively receiver) is joined to a port of network. Every group of senders (or/and of receivers) sends (receives) messages in parallel.

The F*igure 6* presents the ECATNets model of the router problem created in our tool. Places, transitions and arcs inscriptions are as follows:

Places: route*, R1, R2, R3, S1,S2 ,S3, Queue1, Queue2, Queue3, Adr1,Adr2, Adrn.*

Transitions: *From-S1, From-S2, From-S3, To-R1, To-R2, To-R3, Check-Adr1, Check-Adr2, Check-Adr3.*

Arcs Inscriptions: we use the definition in term of algebraic specification of the queue: *q* is a variable of type queue. *front(q)* is a function that returns the message *m* that is in the head of the queue *q*. *addq(m, q)* is a function that adds the message at the end of *q*. *remove(q)* is a function that returns the remainder of the queue *q* after deleting the first message (in head).

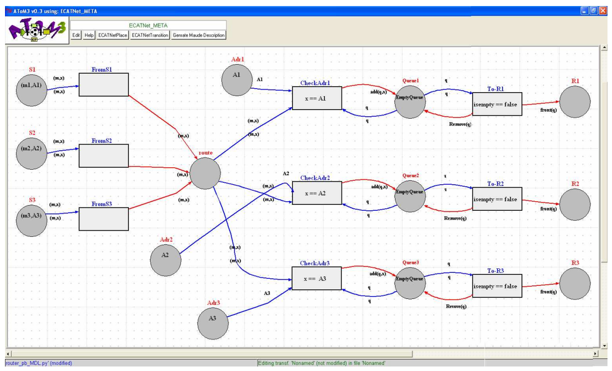


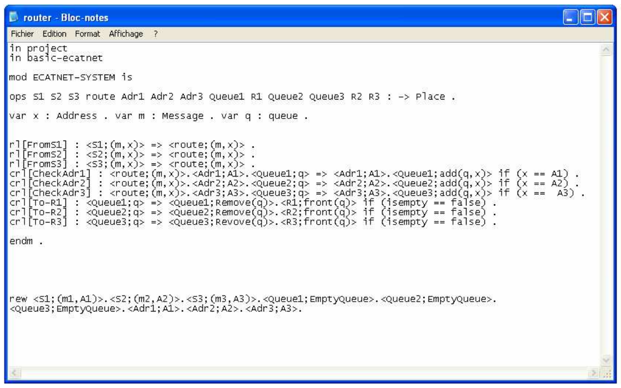
Figure6. ECATNets modeling a router problem created in our tool

It must be noted that the initial state (initial marking) of the ECATNets model is indicated by its places marking. For each place in model, we have place name on the upper and its contents marking inside. The marking of place *S1* for example is *(m1, A1).*

8.2. Translating ECATNets Model to Maude Description

This step has graphical representation of an ECATNets model as input. It consists of translating this graphical representation into its equivalent Maude description using the graph grammar defined in previous section. To realise this translation, the user have to click on the "*Generate Maude Description*" button in the interface of the generated tool.

In fact, Maude specification contains on one hand the structure of the ECATNet and on the other hand, the initial state of this ECATNets. The output of this step is the file (*router.maude)* which contains two elements: an equivalent code in Maude of ECATNet structure and an initial state in Maude syntax as shown in Figure 7.



```
in project
in basic-ecatnet

mod ECATNET-SYSTEM is

ops S1 S2 S3 route Adr1 Adr2 Adr3 Queue1 R1 Queue2 Queue3 R2 R3 : -> Place .

var x : Address . var m : Message . var q : queue .


rl[FromS1] : <S1;(m,x)> => <route;(m,x)> .
rl[FromS2] : <S2;(m,x)> => <route;(m,x)> .
rl[FromS3] : <S3;(m,x)> => <route;(m,x)> .
crl[CheckAdr1] : <route;(m,x)>.<Adr1;A1>.<Queue1;q> => <Adr1;A1>.<Queue1;add(q,x)> if (x == A1) .
crl[CheckAdr2] : <route;(m,x)>.<Adr2;A2>.<Queue2;q> => <Adr2;A2>.<Queue2;add(q,x)> if (x == A2) .
crl[CheckAdr3] : <route;(m,x)>.<Adr3;A3>.<Queue3;q> => <Adr3;A3>.<Queue3;add(q,x)> if (x ==  A3) .
crl[To-R1] : <Queue1;q> => <Queue1;Remove(q)>.<R1;front(q)> if (isempty == false) .
crl[To-R2] : <Queue2;q> => <Queue2;Remove(q)>.<R2;front(q)> if (isempty == false) .
crl[To-R3] : <Queue3;q> => <Queue3;Revove(q)>.<R3;front(q)> if (isempty == false) .

endm .



rew <S1;(m1,A1)>.<S2;(m2,A2)>.<S3;(m3,A3)>.<Queue1;EmptyQueue>.<Queue2;EmptyQueue>.
<Queue3;EmptyQueue>.<Adr1;A1>.<Adr2;A2>.<Adr3;A3>.
```

Figure 7. Generated Maude specification of router model

As illustrated in Figure7, the initial state of the ECATNets model in Maude syntax is a sequence of pairs separated with points. Where the first element of pair is a place and the second one is its marking. For example, The pair *<S1;(m1,A1)>* indicate that place *S1* has *(m1,A1)* as marking.

8.3. Simulation

The output of the previous steps (*router.maude* file) is the input of this one. In order to perform the simulation of the resulted Maude specification, we have invoked the rewriting logic language Maude. Simulation consists of transforming the initial state to another by doing one or many rewriting actions. Therefore, in addition to generated file, the user may give to the Simulator the number of rewriting steps if (he/she) wants to check intermediary states. If this number is not given, the Simulator continues the simulation operation until reaching a final state. We notice that infinite case is possible. The Result marking (final state) of the simulation is given in the same manner as initial one.

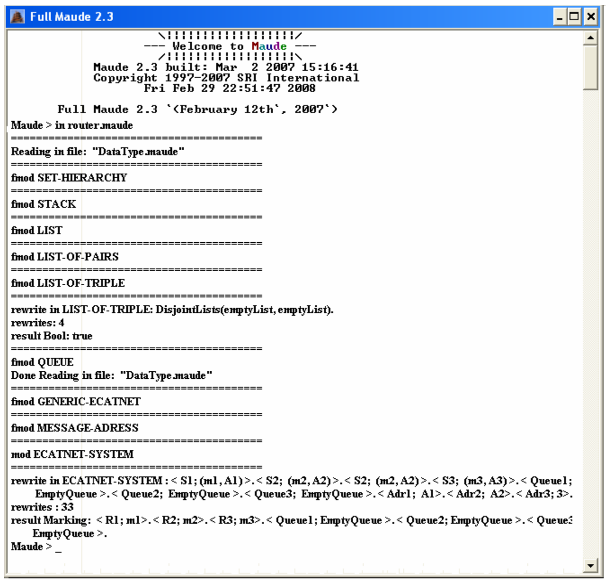
In our example (see Figure 8), we have asked the application to perform the simulation on the following initial state without indicating the number of rewriting steps:

S1;(m1,A1)>. <S2;(m2,A2)>. <S3;(m3,A3)>. <Queue1;EmptyQueue>. <Queue2;EmptyQueue>.<Queue3;EmptyQueue>. <Adr1;A1>. <Adr2;A2>. <Adr3;A3>.

The result marking of the simulation is:

<R1; m1>. <R2; m2>. <R3; m3>.<Queue1; EmptyQueue>. <Queue2; EmptyQueue>. <Queue3; EmptyQueue>.

This final marking indicates that all submitted messages (m1, m2 and m3) from senders (S1, S2 and S3 respectively) in network of communication are achieved in their destinations (R1, R2 and R3 respectively) according to their addresses (A1, A2 and A3 respectively).



```
                   \||||||||||||||||||/
                 --- Welcome to Maude ---
                   /||||||||||||||||||\
            Maude 2.3 built: Mar  2 2007 15:16:41
            Copyright 1997-2007 SRI International
                   Fri Feb 29 22:51:47 2008

        Full Maude 2.3 `(February 12th`, 2007`)
Maude > in router.maude
========================================
Reading in file:  "DataType.maude"
========================================
fmod SET-HIERARCHY
========================================
fmod STACK
========================================
fmod LIST
========================================
fmod LIST-OF-PAIRS
========================================
fmod LIST-OF-TRIPLE
========================================
rewrite in LIST-OF-TRIPLE: DisjointLists(emptyList, emptyList).
rewrites: 4
result Bool: true
========================================
fmod QUEUE
Done Reading in file:  "DataType.maude"
========================================
fmod GENERIC-ECATNET
========================================
fmod MESSAGE-ADRESS
========================================
mod ECATNET-SYSTEM
========================================
rewrite in ECATNET-SYSTEM : < S1; (m1, A1) >.< S2;  (m2, A2) >.< S2;  (m2, A2) >.< S3;  (m3, A3) >.< Queue1;
    EmptyQueue >.< Queue2;  EmptyQueue >.< Queue3;  EmptyQueue >.< Adr1;  A1>.< Adr2;  A2>.< Adr3; 3>.
rewrites : 33
result Marking:  < R1; m1>.< R2; m2>.< R3; m3>.< Queue1; EmptyQueue >.< Queue2; EmptyQueue >.< Queue
    EmptyQueue >.
Maude > _
```

Figure8. Execution of ECATNet example under Maude system.

9. 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 graphical way to ECATNets model structure.

References

[1] AGG, http://tfs.cs.tu-berlin.de/agg/

[2] AToM^3 , http://atom3.cs.mcgill.ca/

[3] Bardohl, R., Ehrig, H., De Lara, J. and Taentzer, G. Integrating Meta Modelling with Graph Transformation for Efficient Visual Language Definition and Model Manipulation, Lecture Notes in Computer Science, Springer. v. 2984, p. 214-228, 2006.

[4] Bettaz, M., Chaoui, A., and Barkaoui, K. On Finding Structural Deadlocks in ECATNets Using a Logic of Concurrency, Journal of Computing and Information. v.2 , p. 495-506, 1996.

[5] Bettaz, M. and Maouche, M. How to specify Non Determinism and True Concurrency with Algebraic Term Nets, Lecture Notes in Computer Science, Springer Verlag, Berlin, v. 655, p. 11-30, 1992.

[6] Bettaz, M., Maouche, M., Soualmi, M. and Boukebeche, M. Protocol Specification Using ECATNets, Networking and Distributed Computing. p. 7-35, 1993.

[7] Boudiaf, N., Chaoui, A. and Bakha, H. A rewriting logic based tool for ECATNets' analysis: Edition and Simulation steps description, European Journal of Scientific Research, v. 6, No 2, 2005.

[8] Clavel, M. Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J. and Quesada, J. Maude: Specification and Programming in Rewriting Logic, Internal report, SRI International. 1999.

[9] De Lara, J. and Vangheluwe, H. AToM^3: A Tool for Multi-Formalism Modelling and Meta-Modelling", Lecture Notes in Computer Science. Springer-Verlag, v. 2306, p.174-188, 2002.

[10] De Lara, J. and Vangheluwe, H. Computer aided multi-paradigm modelling to process petri-nets and statecharts, International Conference on Graph Transformations (ICGT), Lecture Notes in Computer Science, Springer-Verlag, Barcelona, Spain. v. 2505,p. 239-253, 2002.

[11] De Lara, J. and Vangheluwe, H. Meta-Modelling and Graph Grammars for Multi-Paradigm Modelling in AToM^3, Manuel Alfonseca, Software and Systems Modelling, Springer-Verlag. Special Section on Graph Transformations and Visual Modeling Techniques. v. 3, p. 194-209, 2004.

[12] EMF, Home page http://www.eclipse.org/emf/

[13] FUJABA, Home page http://www.fujaba.de/

[14] GEF, http://www.eclipse.org/gef/

[15] GME, http://www.isis.vanderbilt.edu/gme/

[16] GMF, http://www.eclipse.org/gmf/

[17] GReAT, http://www.escherinstitute.org/Plone/tools/

[18] Kelly, S., Lyytinen, K. and Rossi, M. MetaEdit+: A fully con_gurable Multi-User and Multi-Tool CASE and CAME Environment, In Advanced Information System Engineering, LNCS. v.1080. Berlin, 1996.

[19] Kerkouche, E. and Chaoui, A. A Formal Framework and a Tool for the Specification and Analysis of G-Nets Models Based on Graph Transformation, International Conference on Distributed Computing and Networking -CDCN'09-, Springer-Verlag Berlin Heidelberg. LNCS v. 5408, p. 206–211, 2009.

[20] Kerkouche, E., Chaoui, A., Bourennane, E. and Labbani, O. Modelling and verification of Dynamic behaviour in UML models, a graph transformation based approach, proceedings of SEDE'2009, Las Vegas, Nevada, USA, 2009.

[21] Meseguer, J. Rewriting Logic as a Semantic Framework of Concurrency: a Progress Report, Lecture Notes in Computer Science, Springer-Verlag. V.119, p. 331-372, 1996.

[22] PROGRES, http://www-i3.informatik.rwth-aachen.de/research/projects/progres/

[23] Python, htpp://www.python.org

[24] Rozenberg, G. Handbook of Graph Grammars and Computing by Graph Transformation, World Scientific. v.1, 1999.

[25] TIGER, http://tfs.cs.tu-berlin.de/tigerprj/