From UML Sequence Diagrams to ECATNets: a Graph Transformation based Approach for modelling and analysis

Allaoua Chaoui

Department of Computer Science, University Mentouri Constantine, Algeria

a_chaoui2001@yahoo.com

Raida ElMansouri

Department of Computer Science, University Mentouri Constantine, Algeria

raidaelmansouri@yahoo.fr

Wafa Saadi

Department of Computer Science, University of Biskra, Algeria

saadiwafa07@yahoo.fr

Elhillali Kerkouche

Department of Computer Science, University of Oum Elbouaghi, Algeria

elhillalik@yahoo.fr

ABSTRACT

It is now recognized that UML is considered nowadays as the standardized language for object oriented modeling and analysis. However, UML cannot be used for automatic analysis and simulation. So, UML needs a well-defined semantic base for its notation. Petri nets are a formal and graphical language appropriate for systems modelling and analysis. ECATNets are a category of Petri nets based on a safe combination of algebraic abstract types and high level Petri Nets. ECATNets' semantic is 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. In this paper we propose an approach to generate ECATNets models from UML sequence diagrams. Then the resulting models are mapped to Maude language for analysis purposes. The approach is illustrated by two examples.

Key Words: UML, Graph transformation, graph grammars, colored Petri nets

1. Introduction

The Unified Modeling Language (UML) [24] is widely accepted by the Software Engineering community as a standard software modelling language. It consists of many diagrams. The most important ones are *use cases* diagrams, *class* diagrams, *sequence* diagrams, *collaboration* diagrams, *state chart* diagrams, etc… Some diagrams are used to model the structure of a system (use cases diagrams, class diagrams, etc…) while others are used to model the behaviour of a system (state charts diagrams, collaboration diagrams, etc…). UML Sequence diagrams model the interaction between a

set of objects through the messages (or events) that may be dispatched among them.
Petri nets [20] were introduced firs by *Carl Adam Petri* in the early 1960s as a mathematical tool for modeling distributed systems supporting the notions of concurrency, non determinism, communication and synchronization. There are many varieties of Petri nets from simple net [20] to more complex nets (high level Petri Nets) such as colored nets [13], ECATNets [3], Predicate/Transition nets [12], object Petri nets [16], G-Nets [10], etc ...

ECATNets are an algebraic Petri net category based on a safe combination of algebraic abstract types and high level Petri Nets [3]. In addition to modelling, ECATNets allow the verification and simulation of concurrent systems [4, 18]. The most distinctive feature of ECATNets is that their semantic are defined in terms of rewriting logic [19], allowing us to built models by formal reasoning. The rewriting logic Maude [19] is considered as one of very powerful languages in the specification and verification of concurrent systems. Rewriting logic gives to ECATNets a simple, more intuitive and practical textual 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 plus of 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 [4].

In this paper, we propose an approach to translate UML sequence diagrams models to their equivalent ECATNets models. The resulting models can be subjected to various Petri net analysis techniques. This helps in the validation of UML behavioral specifications. Our approach is based on graph transformation since UML sequence diagrams and ECATNets models are graphs.
The rest of the paper is organized as follows: In section 2, we present some related work. In section 3, we recall some basic notion about ECATNets and their integration in rewriting logic. In section 4, we recall some concepts about Graph Grammars and give an overview of the AToM^3 tool [1]. In section 5, we describe our approach that transforms UML sequence diagrams models to their equivalent ECATNets models. In section 6, we illustrate our generated tool through two examples. Finally concluding remarks are drawn from the work and perspectives for further research are presented in section 7.

2. Related Work

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 [2] a transformation between Statecharts and DEVS is given. In [7] the authors used meta-modeling and graph grammars to process GPSS models. The processing of UML Class Diagrams, Activity Diagrams, and many others using graph transformation can be found in [1,2,9]. 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 [14], the authors have presented an approach that generates automatically a Maude specification from ECATNets models. First they have proposed an ECATNets meta-model in the UML Class Diagram formalism with the meta-modelling tool AToM^3, and use it to generate automatically a visual modelling tool to process models in ECATNets formalism. They also defined 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. In [11], the authors have provided the INA Petri net tool with a graphical environment. First, they have proposed a meta-model for Petri net models and used it in the meta-modelling tool AToM^3 to generate automatically a visual modelling tool to process models in INA formalism. Then they defined a graph grammar to translate the models created in the generated tool to a textual description in INA language (INA specification). Then the INA is used to perform the analysis of the resulted INA specification. In [15], 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. Their 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, their framework allows a developer to translate automatically each resulted PrT-Nets model into PROD's net description language. To this end, they 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 a visual modeling tool for each formalism according to its

proposed Meta-Model. They have also proposed two graph grammars. The first one performs the transformation of the graphically specified G-Nets models to semantically equivalent PrT-Nets models. The second one translates the resulted PrT-Nets models into PROD's net description language.

In this paper, we propose an approach that translates UML sequence diagrams models to their equivalent ECATNets models. The resulting models can be subjected to various Petri net analysis techniques. This helps in the validation of UML behavioral specifications. Our approach is based on graph transformation since UML sequence diagrams and ECATNets models are graphs.

3.ECATNets,Graph transformation, and ATOM^3

In this section we recall some main concepts about ECATNets, graph transformation, and ATOM3 tool.

3.1 ECATNets

ECATNets [3] 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 [19]. Motivating ECATNets (Extended Concurrent Algebraic Terms Nets) leads to motivating Petri Nets, abstract data types, as well as their combination into a unified framework [4]. Petri net are used for their foundation in concurrency and dynamics, while abstract data types are used for their data abstraction power and solid theoretical foundation. Their association into a unified framework is motivated by the need to explicitly specify process behaviour and complex data structure in real systems [3]. For more details see [3].

3.2 Graph Grammars and ATOM^3

This section recalls some fundamental notions about graph transformation and ATOM^3 tool.

3.2.1 Graph Grammars

The research area of Graph Grammars is a discipline of computer science which dates back to the early of seventies. Methods, techniques, and results from the area of graph transformations have already been applied in many fields of computer science such as formal language theory, concurrent and distributed systems modelling, software engineering, visual modelling, etc. The wide applicability is due to the fact that graphs are a very natural way of explaining complex situations on an intuitive level. Hence, they are used in computer science almost everywhere. On the other hand, Graph grammars provide dynamic aspect to these descriptions since it can describe the evolution of graphical structures. 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 1*.

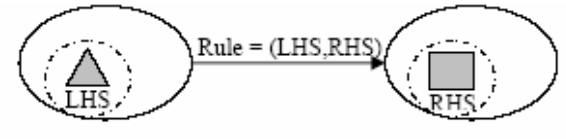

Figure 1. Rule-based Modification of Graphs

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. Furthermore, rules may also have a condition that must be satisfied in order for the rule to be applied, as well as actions to be performed when the rule is executed. A rewriting system iteratively applies matching rules in the grammar to the host graph until no more rules are applicable.

3.2.2 AToM3 :An Overview [1]

AToM^3 is a visual tool for multi-formalism modelling and meta-modelling. As it has been implemented in Python [Python], it is able to run (without any change) on all platforms for which an interpreter for Python is available: Linux, Windows and MacOS. The two main tasks of AToM^3 are meta-modelling and model transformation.

In the next sections, we will discuss how we use AToM^3 to meta-model sequence diagrms and ECATNets formalism and how to generate the ECATNets models from sequence diagrams.

4. The Approach

4.1 UML Sequence diagram Meta-Model

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

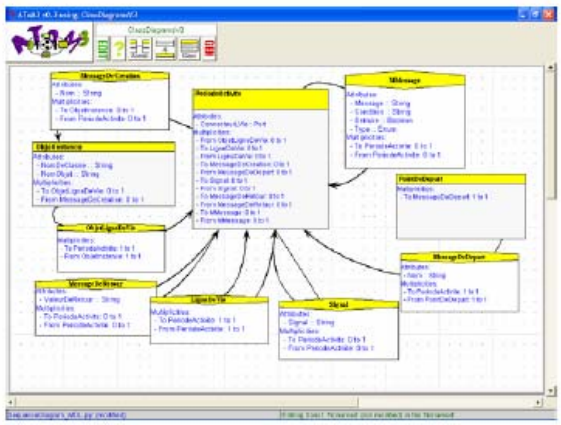

Figure 2. Meta-model of sequence diagrams

Since a sequence diagram is composed of classes and messages, we have proposed to meta-model sequence diagrams 3 main classes (see figure 2):

- The class *PointDeDepart*
This class represents the start of a sequence diagram. It is represented visually through a *Gray square* (see figure 6) and connected with the class PeriodeActivite by an association MessageDeDepart. The association has an attribute called **Nom** and connects a single instance of the class PointDeDepart with a single instance of the class PeriodeActivite. It is represented graphically by a blue arrow.

- The class *ObjetInstance*
it represents the interacting objects in the sequence diagram. Each object has its name and the name of his class. The class ObjetInstance is linked by the association *ObjetLigneDeVie* with the class *PeriodeActivite*. it connects with an object not more than one instance of the class PeriodeActivite. An object is viewed through a blue rectangle that bears his name and the name of his class (Figure 6).

- The class *PeriodeActivite*
This is the class that determines the period of the activity of an object. It has an attribute *ConnecteurLVie* of type *Port* which acts as a connector between the various segments of the period of activity of an object. Two periods of activity coming from different objects may be linked either:

- By a message (the association *MMessage*) wich has a name (attribute Message) and a type. If the type is a condition then it should be set in the attribute condition. It may also be a message of destruction if the Destroy attribute has the value true. In our meta-model, two periods of activity may be linked to at most one message.
- By a signal (the sssociation *Signal*) that has an attribute signal. It can have at most one instance between two period of activities.
- By an association *MessageDeRetour* that has an attribute *ValeurDeRetour*. A return message is a response to a given message.
- And finally by an association *LigneDeVie*.

An instance of the class *PeriodeActivite* is represented graphically by a *blue rectangle*. It can be linked with an instance of the class *ObjetInstance* through a creation message (*MessageDeCreation* Association) which has a name and whether it exists is unique.
The meta-model shown in **Figure 2** will enable us to generate a modeling tool for different models of sequence diagrams by only clicking on **GEN** button. **Figure 3** presents the tool generated with an example of a sequence diagram.

4.2 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 also the UML Class Diagrams and the constraints are also expressed in Python code.
Since ECATNets models 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 4*. We have also specified the visual representation of each component according to ECATNets notation. To fully define our meta-model, we have added two constraints "MoreThanOneInputArc" and "MoreThanOneOutputArc" in places class. The former is used to don't allow more than one input arc to a given transition, whereas the latter is used to don't allow more than one output arc from a given transition.

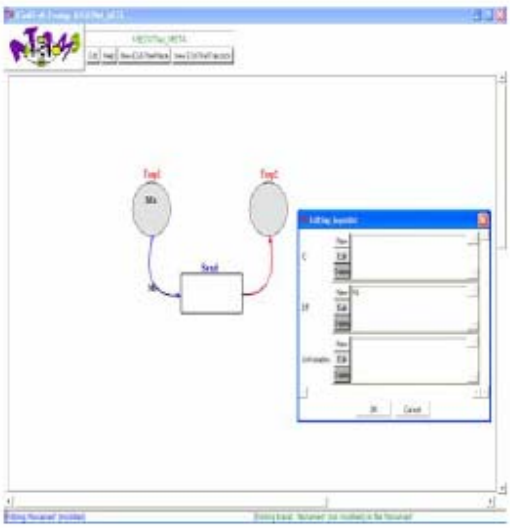

Figure3. Generated tool to process ECATNets

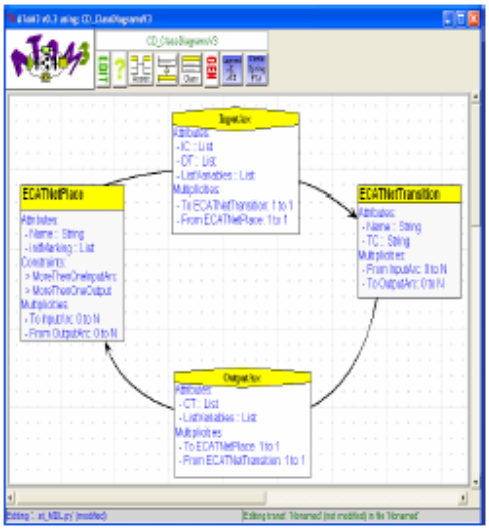

Figure 4. ECATNets Meta-Model

Given our meta-model, we have used AToM^3 tool to generate a visual modelling environment for ECATNets models. The generated ECATNets modelling tool is shown in f*igure 3*.

The following sub section describes the grammar proposed to perform the transformation between sequence diagrams and ECATNets models.

4.3 The graph grammar

To perform the transformation between sequence diagrams and ECATNets models, we have proposed a grammar called *SequenceDiagVEcatNet* composed of rules divided into five categories.

Category 1. Rules that transforms all messages: This category includes three rules that aim to transform the messages in the source model.

Rule 1: Transforming a message

Name: RtransformerMS **Priority:** 1

Role: This rule allow us to transform a message from the sequence diagram to two places and a transition in ECATNets formalism. The first place represents the message during its emission, the second one represents it during its reception and the transition symbolizes the sending of the message. The node number 4 on the right hand side (RHS) of the rule gives the marking of the message. Sending the message is simulated by the destruction of the input place of the transition (DT of the input arc) and its reception by its creation (CT arc output) in the output place. If the message is conditioned, the Transition Condition (TC) receives the condition of the message. The node number four (emission place) is named "**PSend.[Message]**", the node number five (reception place) is called **"PReceive.[Message]"** and number six (transition : transmettre) is named **"Send.[Message]"** (Figure 5).

Rule 2: Transformation of an initial message

Name: RMsInitial **Priority:** 2

Role: It allows us to transform the original message in a sequence diagram to a place and a transition in the equivalent ECATNets formalism, the place is named "Departure" and its marking is initialized by the starting message. The transition symbolizes the reception of the sent query in the message. It is named **"Receive.[MessageDeDepart]."** (Figure 5.b).

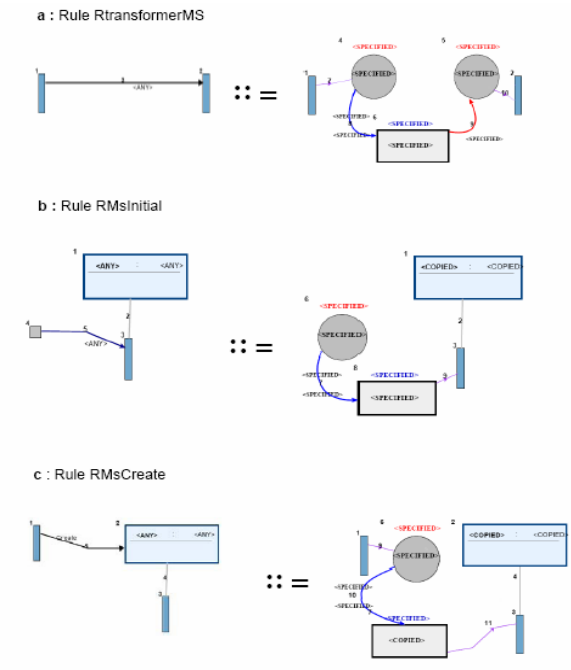

Figure 5. Rules that transform messages

Rule 3: Transform a message creation
Name: RMsCreate **Priority:** 2
Role: This rule allows us to transform a message that creates an object. A message that creates an object can be received by an object to start its activity (beginning of its life line). In our case, a message of creation is seen as a place that models its sending and a transition representing its reception by the created object. The place is called **"PSend.[MessageDeCréation]"** and the transition **"Receive.[MessageDeCréation]"** (see Figure 5.c).

Category 2: Rules that rename all transitions and create the reception transition
Category 3: Rule that renames all places
Category 4: Rule that process a return message
Category 5: Rules of suppression of periods of activity
For lack of space, we have not presented all the rules.

6. Examples

Let us apply our approach on the two following examples.

6.1 Example 1

Let us consider an example of sequence diagram from a system describing an internet commerce system as shown by figure 6. We have applied our automatic approach on this example and we have obtained the equivalent ECATNets shown in Figure 7.

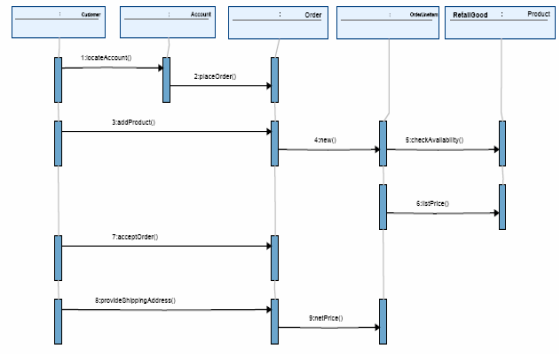

Figure 6. Sequence diagram of the example 1

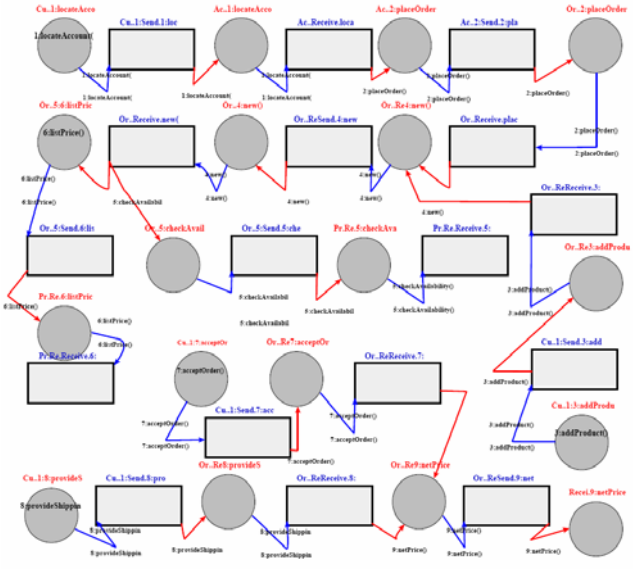

Figure 7. The obtained ECATNets of the example 1

6.2 Example 2

We have also applied our automatic approach on the example described by the sequence diagram shown in figure 8 and obtained the equivalent ECATNets shown in Figure 9.

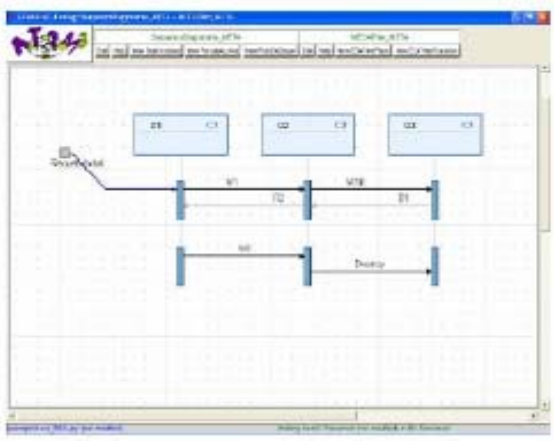

Figure 8. Sequence diagram of the example 2

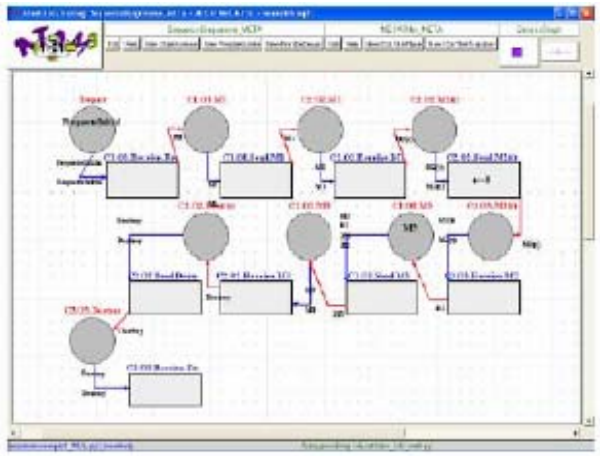

Figure 9. The obtained ECATNets of the example 2

We have then applied our previous tool [14] on the resulted ECATNets of figure 9 and obtained the Maude specification in of figure 10.
Now the process of verification using Maude system can be used.

7. Conclusion and Future Work

In this paper, we proposed an approach to translate UML sequence diagrams models to their equivalent ECATNets models. The resulting models can be subjected to various Petri net analysis techniques. This helps in the validation of UML behavioral specifications. Our approach is based on graph transformation since UML sequence diagrams and ECATNets models are graphs. We have illustrated our approach using two examples. In a future work we plan to translate other UML diagrams to ECATNets and perform some verification.

```
in project
in basic-ecatnet

mod ECATNET-SYSTEM is

ops C1.O1.M1 C2.O2.M1 C2.O2.M2(t) C3.O3.M2(t) C1.O1.M3 C2.O2.M3 C2.O2.Destroy C3.O3.Destroy Depart : -> Place .

crl[C1.O1.Send.M1] : <C1.O1.M1;M1> => <C2.O2.M1;M1> if (M1) .
crl[C2.O2.Send.M2(t)] : <C2.O2.M2(t);M2(t)> => <C3.O3.M2(t);M2(t)> if (t>=0) .
rl[C1.O1.Send.M3] : <C1.O1.M3;M3>.<C1.O1.M3;R2> => <C2.O2.M3;M3> .
rl[C2.O2.Send.Destroy] : <C2.O2.Destroy;Destroy> => <C3.O3.Destroy;Destroy> .
rl[C1.O1.Receive.RequetteInitial] : <Depart;RequetteInitial> => <C1.O1.M1;M1> .
rl[C2.O2.Receive.M1] : <C2.O2.M1;M1> => <C2.O2.M2(t);M2(t)> .
rl[C3.O3.Receive.M2(t)] : <C3.O3.M2(t);M2(t)> => <C1.O1.M3;R2> .
rl[C2.O2.Receive.M3] : <C2.O2.M3;M3> => <C2.O2.Destroy;Destroy> .
rl[C3.O3.Receive.Destroy] : <C3.O3.Destroy;Destroy> => .

endm .

rew <C1.O1.M3;M3>.<Depart;RequetteInitial>.
```

Figure 10 : Maude specification of the example

8. References

[1] Home page: http://atom3.cs.mcgill.ca/

[2] 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.

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

[2] 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] Y.Deng et.al. "Integrating Software Engineering Methods and Petri nets for the specification and prototyping of complex information systems". *Application and Theory of Petri nets 1993, 14th International Conference proceedings*, Chicago, pp 203-223, June 1993.

[11] R. Elmansouri, E. Kerkouche, and A. Chaoui, A Graphical Environment for Petri Nets INA Tool Based on Meta-Modelling and Graph Grammars, Proceedings offing and Technology, ISSN 2070-3740, Vol 34 October 2008,.

[12] Genrich, H. J., Lautenbach, K.System Modelling with High-Level Petri Nets. Theoretical Computer Science, vol. 13 (1981)

[13] K. Jensen, *Coloured Petri Nets*, Vol 1: Basic Concepts, Springer-Verlag 1992.

[14] E. Kerkouche, and A. Chaoui, On the use of Meta-Modelling and Graph Grammars to process and simulate ECATNets model, proceeding of MS'2008, Port Said, April 8-10, 2008, EGYPT.

[15] E. Kerkouche, and A. Chaoui, A Formal Framework and a Tool for the specification and analysis of G-Nets models based on graph transformation, To appear in V. Garg, R. Wattenhofer, and K. Kothapalli (Eds.): ICDCN 2009, LNCS 5408, pp. 206–211, 2009, Springer-Verlag Berlin Heidelberg 2009

[16] C.A.Lakos. *Object Petri nets – Definition and relationship to colored nets*. Technical Report TR 94-3, Computer Science Department, University of Tasmania.

[17] M. Clavel and AL. "Maude : Specification and Programming in Rewriting Logic". Internal report, SRI International, 1999.

[18] M. Maouche, M. Bettaz, G. Berthelot and L. Petrucci. "Du vrai parallélisme dans les réseaux algébriques et de son application dans les systèmes de production". MOSIM'97, 1997.

[19] J. Meseguer, "Conditional Rewriting Logic as a unified model of concurrency". Theoretical Computer Science, 1992.

[20]. T. Murata, "Petri Nets: Properties, Analysis and Applications", *Proceedings of the IEEE*, Vol.77, No.4 pp.541-580, April 1989.

[21] Python home page: htpp://www.python.org

[22] Roch S. and Starke P.H. (2002). Integrated Net Analyze, User manual, 2002.

[23] Grzegorz Rozenberg: Handbook of Graph Grammars and Computing by GraphTransformation, World Scientific, 1999.

[24] G. Booch, J. Rumbaugh and I. Jacobson. *The Unified Modeling Language User Guide*, Addison-Wesley.