International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:3, No:3, 2009

Towards an Automatic Translation of Colored Petri Nets to Maude Language

Noura Boudiaf and Abdelhamid Djebbar

***Abstract*—**Colored Petri Nets (CPN) are very known kind of high level Petri nets. With sound and complete semantics, rewriting logic is one of very powerful logics in description and verification of non-deterministic concurrent systems. Recently, CPN semantics are defined in terms of rewriting logic, allowing us to built models by formal reasoning. In this paper, we propose an automatic translation of CPN to the rewriting logic language Maude. This tool allows graphical editing and simulating CPN. The tool allows the user drawing a CPN graphically and automatic translating the graphical representation of the drawn CPN to Maude specification. Then, Maude language is used to perform the simulation of the resulted Maude specification. It is the first rewriting logic based environment for this category of Petri Nets.

***Keywords*—**Colored Petri Nets, Rewriting Logic, Maude, Graphical Edition, Automatic Translation, Simulation.

I. Introduction

COLORED Petri Nets (CPN) are very known kind of high level Petri nets. Rewriting logic has sound and complete semantics [6] and it is considered as one of very powerful logics in description and verification of non-deterministic concurrent systems. Also, the rewriting logic language Maude [7] is considered as one of very powerful languages in specification, programming and verification of non-deterministic concurrent systems. CPN semantics are defined in terms of rewriting logic [8], allowing us to built models by formal reasoning. Rewriting logic gives to CPN a simple, more intuitive and practical textual version to analyze, without loosing formal semantic (mathematical rigor, formal reasoning).

However, Maude system offers textual way to the user to create and deal with CPN system. Execution under Maude system is done by using command prompt style. In this case, we loose the graphical aspect of CPN formalism which is important for the clarity, simplicity and readability of a system description.

In this paper, we propose a simple tool for automatic translation of CPN to Maude language. This tool includes also graphical edition and simulation of CPN by using Maude. The proposed tool is a simple tool to well exploit the advantages of CPN formalism such as graphical aspect, simplicity and readability that are important for preliminary description in system development. This tool acts as follows : it allows to the user to edit graphically a CPN and then converts the graphical representation to its equivalent description in Maude. Thereafter, the tool calls the Maude system for the execution of the CPN and reconverts the resulting marking described in Maude to a graphical representation. This step allows us to perfectly exploit all advantages of Maude described above. We notice that the integration of CPN in rewriting logic facilitated the development of such environment. With the help of a CPN example, we will compare the simulations of the example under Maude system and our tool. One of the objectives of this work is to fully exploit the power of this logic and Maude language. Maude is simple, very expressive and efficient [4].

Of course, in the context of Petri nets simulation and analysis, many tools have been developed. Most of these tools for CPN are implemented using the known imperative languages like Java, C++. We mention here some much known tools that are proposed for CPNs. CPN/Tools [2] is a major redesign of the Design/CPN [9] tool for editing, simulation and state space analysis of Colored Petri Nets. CPN Tools [10] is a tool for editing, simulating and analyzing Colored Petri Nets. The functionality of the simulation engine and state space facilities are similar to the corresponding components in Design/CPN.

Let's note that these proposed environments are mature and include many analysis tools. We can not compare our tool with these environments in terms of services provided. But, our tool is special because it is the first one which offers an automatic translation of CPN to Maude. This integration allows to the CPNs to get a sound and complete semantic especially the true concurrency description. Our work concerning graphic and edition and simulation of Petri nets based Maude is not limited to just CPNs. We already proposed the development of graphic simulator for ordinary Petri nets and for another category of algebraic Petri nets ECATNets by using Maude [3]. This work constitutes a way of important investigation. We want to use, in fact, only one platform (Maude) to allow the communication between systems developed beforehand by using different kinds of Petri nets. This can be carried out by an automatic generation of Maude descriptions starting from ordinary Petri nets, ECATNets, CPN and other Petri nets models which are integrated in Maude.

The remainder of this paper is organized as follows: in section 2, we present a short outline on the rewriting logic and

Noura Boudiaf is with the University of Oum El Bouaghi, Algeria (e-mail: boudiafn@gmail.com).
Abdelhamid Djebbar is with the University of la méditerranée, Marseille, Marseille, France (e-mail: djb_abdelhamid@yahoo.fr)

Maude. We give a brief introduction on CPN formalism and their integration in rewriting logic in section 3 through a simple example. Most important functionalities of our simple application are illustrated in section 4. We show also in this section how we run our application with the help of the example. Software and hardware used in developing our application are mentioned in section 5. Finally, section 6 concludes the paper.

II. Rewriting Logic REVIEW

In rewriting logic, each concurrent system is represented by a rewrite theory $\Re$ = (Σ, E, L, R). Its static structure is described by the signature (Σ, E), whereas its dynamic structure is described by the set of labelled rewrite rules R, which are applied modulo the equation E. An important consequence of the rewriting logic definition is that a rewrite theory $\Re$ = (Σ, E, L, R) can be viewed as an executable specification of the concurrent system that it formalizes. In this section we recall the basic definitions of the rewriting logic.

A labelled rewrite theory $\Re$ is a 4-tuple $\Re$ = (Σ, E, L, R) where (Σ, E) is a signature; Σ is the sorts set and operators and E is a set of Σ-equations. The signature (Σ, E) is an equational theory which describes the particular algebraic structure of the states of a system (multiset, binary tree, etc.) which are distributed according to this same structure. $R \subseteq L \times (T_{\Sigma,E}(X))^2$ is the set of pairs whose first component is a label and the second is a pair of E-equivalence classes of terms, with X = $\{x_1,\ldots,x_n,\ldots\}$a countable infinite set of variables. The elements of R are called conditional rewrite rules. They describe the elementary and local transitions in a concurrent system. Each rewrite rule corresponds to an action being able to occur, simultaneously, with other actions. The rewriting will operate on equivalence classes of terms, modulo the set of equations E. For a rewrite rule (r, ([t],[t']), $([u_1],[v_1]),\ldots,([u_k],[v_k])$) we use the notation, r: $[t]\rightarrow[t']$ if $[u_1]\rightarrow[v_1] \wedge\ldots\wedge [u_k]\rightarrow[v_k]$, where [t] represents the equivalence class of the term t. A rule r expresses that the equivalence class containing the term t is changed to the equivalence class containing the term t' if the conditional part of the rule, $[u_1]\rightarrow[v_1] \wedge\ldots\wedge [u_k]\rightarrow[v_k]$, is verified.

Given a labeled rewrite theory $\Re$, we say that $\Re$ entails a sequent r: $[t]\rightarrow[t']$, or that r: $[t]\rightarrow[t']$ is a (concurrent) $\Re$-rewrite and write $\Re \vdash r : [t] \rightarrow [t']$ iff $[t]\rightarrow [t']$ is derivable from the rules in $\Re$ by a finite application of the deduction rules (reflexivity, transitivity, congruence, and replacement) of rewriting logic.

A rewrite theory is a static description of a concurrent system. Its semantics is defined by a mathematical model which describes its behavior. The model for a given labelled rewriting theory $\Re$ = (Σ, E, L, R) is a category $\tau_{\Re}(X)$ whose objects (states) are equivalence classes of terms $[t]\in T_{\Sigma,E}(X)$ and whose morphisms (transitions) are equivalence classes of proof-terms representing proofs in rewriting deduction .

- **Reflexivity.** For every $[t] \in T_{\Sigma,E}(X)$: $\dfrac{}{[t]\rightarrow[t]}$
- **Congruence.** For every $f \in \Sigma_n$, $n \in N$: $\dfrac{[t_1]\rightarrow[t_n'] \;\;\ldots\;\; [t_n]\rightarrow[t_n']}{[f(t_1,\ldots,t_n)]\rightarrow[f(t_1',\ldots,t_n')]}$
- **Replacement.** For every rewriting rule $r:[t(x_1,\ldots,x_n)]\rightarrow[t'(x_1,\ldots,x_n)]$ in R, $\dfrac{[w_1]\rightarrow[w'_n] \;\;\ldots\;\; [w_n]\rightarrow[w'_n]}{[t(\bar{w}/\bar{x})]\rightarrow[t'(\bar{w}'/\bar{x})]}$, such that $t(\bar{w}/\bar{x})$ indicates the simultaneous substitution of w_i for x_i in t.
- **Transitivity.** $\dfrac{[t_1]\rightarrow[t_2] \quad [t_2]\rightarrow[t_3]}{[t_1]\rightarrow[t_3]}$

A. *Maude Language*

Maude is a specification and programming language based on rewriting logic. Maude is simple, expressive and efficient. It is rather simple to program with Maude, considering that it belongs to the declarative programming languages. It is possible to describe using Maude different types of applications, from prototyping ones to high concurrent applications. Maude is a competitive language in terms of execution and simulation with imperative programming languages [4]. Three types of modules are defined in Maude. The functional modules, the system modules and the object-oriented modules which can, in fact, be reduced to system modules. Because we do not need object-oriented modules here, so we do explain only system and functional modules.

Functional Modules. The functional modules define data types and related operations, which are based on equations theory. By using equations like simplification rules, each expression called term could be evaluated to its reduced form called canonical representation. All the equal terms by means of equations form an equivalence class. The canonical form represents all the terms of the same equivalence class. The set of all the equivalence classes of the ground (i.e, variable-free) terms constitutes a denotational model for a functional module (initial algebra). Equations in a functional module are oriented. They are used from left to right and the final result of the simplification of an initial term is unique independently of the order in which these equations are applied. In addition to equations, this type of modules supports membership's axioms. These axioms impose constraints so that a term is of a particular type if a certain condition is satisfied. This condition is a conjunction of equations and unconditional tests of memberships.

System Modules. The system modules define the dynamic behavior of a system. This type of module augments the functional modules by the introduction of rewriting rules. This type of module offers a maximum degree of concurrency. A system module describes a "rewriting theory" which includes kinds, operations and three types of statements: equations, memberships and rewriting rules. These three types of statements can be conditional. A rewriting rule specifies a "local concurrent transition" which can proceed in a system.

The execution of such transition, specified by the rule, can take place when the left part of a rule matches to a portion of the global state of the system and the condition of the rule is valid.

III. Colored Petri Nets

Colored Petri nets (CPN) are very known and we can find easily their definition in the literature. Among these definitions we have the following one : a CPN is a tuple CN = [P, T, F, ψ, A, m_0] such that :

1. [P, T, F] is a net, i.e. P and T are finite and disjoint sets called places and transitions, respectively, and F is a relation $F \subseteq (P\times T)\cup(T\times P)$, the elements of which are called arcs;
2. ψ assigns a countable set of colors to every element of $P\cup T$;
3. A assigns to every $[x, y] \in F$ a mapping which in turn assigns a finite semi-positive multiset over ψ(p) to every color in ψ(t) where p is the place of x and y and t is the transition of x and y;

m_0 is a marking which assigns a finite semi-positive multi-set over ψ(p) to every $p \in P$. m_0 is called the initial marking.

A. Representation of CPN in Maude

In this section, we will explain through a very simple example how to express a CPN in Maude language according to [8]. For domains of colors C1 = {A, B}and C2 = {C, D}, we have the simple CPN in the figure 1.

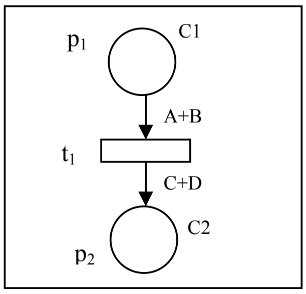


Fig. 1 Example of CPN

Let's take an initial marking such that $M_0(p_1)$ = 2A+2B and $M(P_2)$ is empty. In the sequel, we describe how to obtain the equivalent Maude code of this CPN example. First we create a sort MS to describe a general multi-set, the constant empty for an empty multi-set and the operation without name (__) (the underlines indicate the two parameters of this operation). This operation is commutative, associative and has as an element identity empty. This operation allows the construction of any multi-set including empty multi-set (empty).

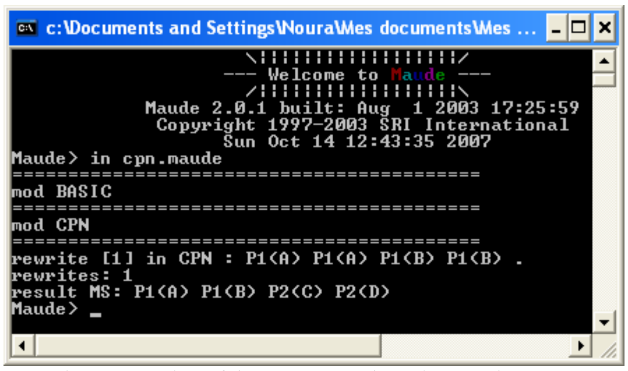
```
mod BASIC is
 sort MS .
 op empty : -> MS .
 op __ : MS MS -> MS [assoc comm. id: empty] .
endm
```

In fact, the marking of CPN is just a multi-set, so we define for C1 (resp. C2) a sort of the same name and we declare that C1 (resp. C2) is a sub-sort of MS. So C1 and C2 are multi-sets. Now, we can declare that A and B are elements of sort C1, C and D are elements of sort C2. Each place P of CPN is represented by an operation in Maude. The domain of P is its domain color C and its co-domain also. In this case, P contains only colors belonging to its domain, then we can write P(r) when r is a color which belongs to C. In Maude code describing the previous CPN example, we have the marking P1(A) P1(B) which denotes that the place P1 contains two colors A and B. A transition in CPN is represented by a rewriting rule in Maude, we have in our example the transition T1 which is represented by a rewriting rule of the same label. This rewriting rule describes how the marking (multi-set) P1(A) P1(B) is transformed to the marking P2(C) P2(D). The unlimited rewriting of an initial marking P1(A) P1(A) P1(B) P1(B) gives us the marking P1(A) P1(A) P2(C) P2(D) as showed in the figure 2.

```
mod CPN is
 pr BASIC .
 sorts C1 C2 .
 subsort C1 < MS . subsort C2 < MS .
 ops A B : -> C1 . ops C D : -> C2 .
 op P1 : C1 -> C1 . op P2 : C2 -> C2 .
 rl [T1] : P1(A) P1(B) => P2(C) P2(D) .
endm
rew P1(A) P1(A) P1(B) P1(B) .
```

B. Execution of CPN Example under Maude System

We find an example of initial state and the result of its unlimited rewriting in figure 2.



```
                    \||||||||||||||||||/
                  --- Welcome to Maude ---
                    /||||||||||||||||||\
            Maude 2.0.1 built: Aug  1 2003 17:25:59
             Copyright 1997-2003 SRI International
                  Sun Oct 14 12:43:35 2007
Maude> in cpn.maude
==========================================
mod BASIC
==========================================
mod CPN
==========================================
rewrite [1] in CPN : P1(A) P1(A) P1(B) P1(B) .
rewrites: 1
result MS: P1(A) P1(B) P2(C) P2(D)
Maude> _
```

Fig. 2 Execution of the CPN example under Maude system

IV. Steps of CPN Simulator Maude-Based

As described in figure 3, most principal steps in this tool are given. Trough the previous example, we explain in following sections the main options of the application.

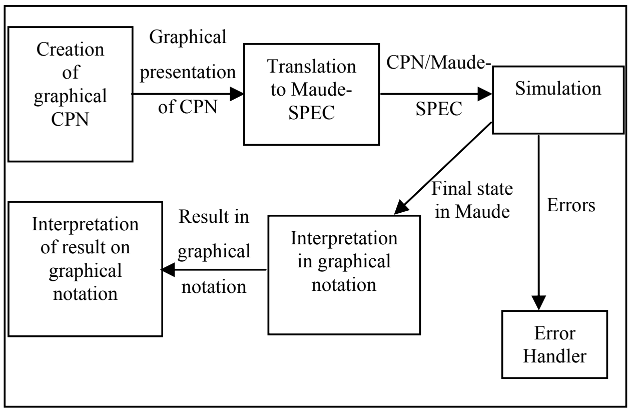

Creation of graphical CPN
Graphical presentation of CPN
Translation to Maude-SPEC
CPN/Maude-SPEC
Simulation
Final state in Maude
Errors
Interpretation in graphical notation
Result in graphical notation
Interpretation of result on graphical notation
Error Handler

Fig. 3 Methodical view on CPN Simulator based Maude

A. *Graphical User Interface*

In this step, user can create his own CPN system. In the figure 4, we find principal window, on what the previous example of CPN is created. In this interface, we show that our application contains from left to right the following options: ToolBar, New, Open, Save, Print, Code Generation and Exit. So, New to create new file containing new CPN, Open to open existent file, Save to save current file, Print for printing current file and Exit to close the tool. In addition to these options, we find two other options ToolBar and Code Generation. ToolBar includes necessary options to create CPN, in the tool bar 'Draw', we have :

- Circle to create new place, rectangle to create new transition, arcs to create new arcs linking places and transitions;
- **D** for **D**elete : to delete a place, a transition or an arc;
- **S** for **S**ub-domain : to create or modify a
- sub-domain for a selected place;
- **M** for **M**arking : to create or modify a marking in a selected place;
- **C** for Arc **C**ondition : to create or modify condition arc.

After creating graphically a CPN, user can click on the icon of Code Generation to generate the corresponding Maude code.

In this figure, we find in the right of the place P1 the set A, B which is the color domain of this place and the multi-set 2A+2B which represents the current marking of P1 (initial marking in this case).

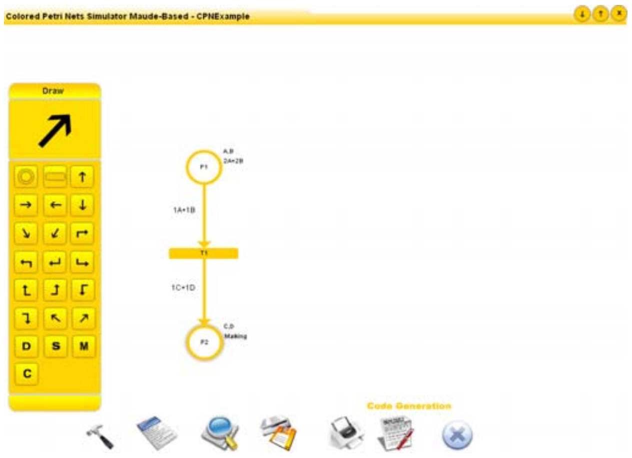
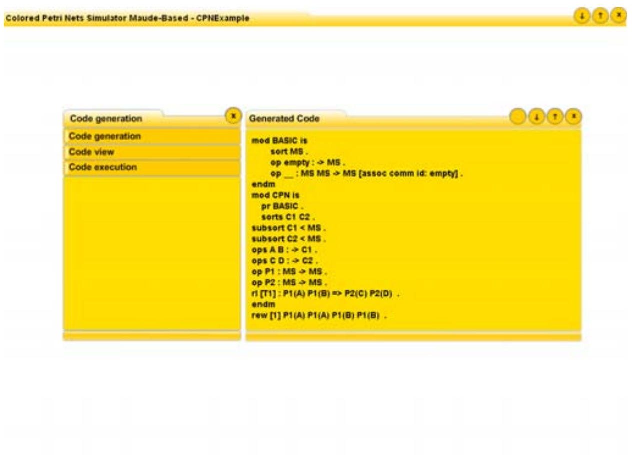


Fig. 4 Application Main Menu with CPN Example

B. *Automatic Translating CPN Graphical Representation to Maude Description*

This step has the graphical representation of CPN model as input. It consists of translating this representation into an equivalent Maude description. In fact, this representation contains on one hand the structure of the CPN and on the other hand, the initial state of this CPN. The output of this step is two elements: an equivalent code in Maude of CPN structure and an initial state in Maude syntax. After clicking on 'code generation' option, another new menu occurs (figure 5). By clicking on the option code generation, we get the generated code equivalent in Maude to CPN according to the translation proposed in [8]. We have to click on 'watch code' option to get the Maude code displayed on the right of the screen as depicted in the figure 5.



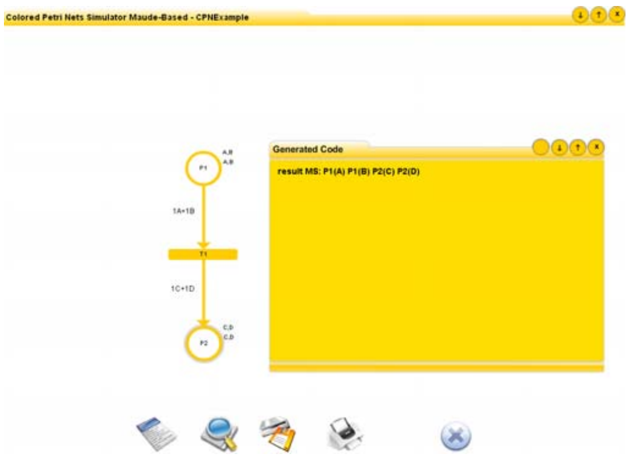
```
mod BASIC is
      sort MS .
      op empty : -> MS .
      op __ : MS MS -> MS [assoc comm id: empty] .
endm
mod CPN is
   pr BASIC .
   sorts C1 C2 .
subsort C1 < MS .
subsort C2 < MS .
ops A B : -> C1 .
ops C D : -> C2 .
op P1 : MS -> MS .
op P2 : MS -> MS .
rl [T1] : P1(A) P1(B) => P2(C) P2(D)  .
endm
rew [1] P1(A) P1(A) P1(B) P1(B)  .
```

Fig. 5 Automatic generated Maude code for CPN example

C. *Simulation*

The output of the previous step is the input of this one. To do simulation, the simulator needs from the user the initial state of the CPN. Simulation consists of transforming this initial state to another by doing one or many rewriting actions.

Therefore, in addition to initial state, 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 rewriting operation until reaching final state. In figure 5, we asked the application to perform the simulation on the previous initial marking without indicating the number of rewriting steps. The Result (final state) of the simulation is given in the same manner as initial one. We notice that infinite case is possible. Let's go back to the example, after displaying the Maude code equivalent to CPN, we can now execute the code by clicking 'code execution' option. In our case, the unlimited rewriting of the example gives the result in the figure 6. Let's note that we exposed the result marking in graphical manner (on the CPN) and in textual way in Maude language.



```
result MS: P1(A) P1(B) P2(C) P2(D)
```

Fig. 6 Application Main Menu with CPN Example

D. Interpretation of the Result in a Graphical Notation

In the case of no errors have been detected, the user obtains (his/her) the result in a box like the case of initial marking. This result is one returned by Maude after being reinterpreted in our description (figure 6).

E. Errors Handler

Our tool deals with errors at different levels. For instance, if the user draws an isolated arc, so when the user asks the tool to generate the code, the application displays that there are mistakes before generating Maude code. In case of some errors have been detected in the level of Maude code, we give to the user by the errors generated by Maude.

V. Technical Aspect of the CPN Simulator

This tool is implemented under MS-Windows XP with the following tools: the programming language Flash MX of Macromedia is used for implementing the graphical editor, Delphi 7.0 for the translation of graphical representation of a CPN to its equivalent Maude description and the version 2.0.1 of Maude system is used for the simulation of the generated description of the CPN. For graphic modelling reason, we held the Flash MX language of Macromedia with regard to the other languages for its graphics power, its capacity to manipulate objects and its compatibility with the other languages. However, it is insufficient for an easy development of this kind of application because it is poor in matter of system interactions as the creation of text files. So, we have needed the use of a hidden module built in Delphi 7 of Borland. As regards to the choice of Delphi for the interactions system for two reasons, the first one it is because it is rich in terms of the interactions system and the second it is for that compatibility with the Flash MX.

VI. Conclusion

In this paper, we proposed a simple rewriting logic based tool for CPN edition and simulation. This application includes tasks like: edition, automatic translation of CPN to Maude and simulation. The editor allows the user to draw a CPN graphically; the automatic translator allows translating the graphical representation of the drawn CPN to a Maude specification. Then the rewriting logic language Maude is used to perform the simulation of the resulted Maude specification. We think that the power of Maude in terms of specification, programming, simulation and verification in plus of the CPN integration in Maude, implies that there is no need to translate CPN to several languages for such requirements. Therefore, we avoid any mistranslation of CPN in several languages and thus any risks about their semantic loss.

The CPN's tool will be extended by adding other tools. The tools planned to be integrated in this application will be developed under Maude system. This tool may be enriched by all already proposed tools for Maude including coherence checker for system modules [5] or and inductive theorem prover [1] to check functional properties modules' properties. The objective of the integration of such tools is making easy the utilization of these tools.

Our work concerning graphic and edition and simulation of Petri nets based Maude is not limited to just CPNs. We already proposed the development of graphic simulator for ordinary Petri nets and for another category of algebraic Petri nets ECATNets by using Maude [3]. This work constitutes a way of important investigation. We want to use, in fact, only one platform (Maude) to allow the communication between systems developed beforehand by using different kinds of Petri nets. This can be carried out by an automatic generation of Maude descriptions starting from ordinary Petri nets, ECATNets, CPN and other Petri nets models which are integrated in Maude.

Let's note that in the same paper [8], authors propose also an integration of algebraic Petri nets (AP¨N) in rewriting logic. In this way, we plan also performing a graphic interface of this kind of Petri Nets with an automatic translation of APN to Maude language for simulation.

References

[1] D. Basin, M. Clavel, and J. Meseguer, “Rewriting logic as a metalogical framework”. In S. Kapoor and S. Prasad, editors, FST TCS 2000, pages 55–80. Springer LNCS, 2000.

[2] M. Beaudouin-Lafon and al., “CPN/Tools: A Tool for Editing and Simulating Coloured Petri Nets - ETAPS Tool Demonstration Related to TACAS”. In: LNCS 2031: Tools and Algorithms for the Construction and Analysis of Systems, pages 574-pp. Springer Verlag, 2001.

[3] N. Boudiaf, “Développement des Outils Basés Maude pour les ECATNets. Domaine d’Application : Analyse des Programmes Ada,”. Phd Thesis, University of Constantine, 2007.

[4] M. Clavel and aL,” Maude Manual (Version 2.2)”, Internal report, SRI International, December 2007.

[5] Francisco Durán, "Coherence Checker and Completion Tools for Maude Specifications”. Manuscript. University of Málaga. July 2000.

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

[7] J. Meseguer, “Rewriting logic and Maude: a Wide-Spectrum Semantic Framework for Object-based Distributed Systems”, In S. Smith and C.L. Talcott, editors, Formal Methods for Open Object-based Distributed Systems, (FMOODS’2000), p. 89-117, 2000.

[8] Mark-Oliver Stehr, José Meseguer, Peter Csaba, “Rewriting Logic as a Unifying Framework for Petri Nets”, Lecture Notes in Computer Science, volume 2128, 2001.

[9] Meta Software Corporation., “Design/CPN Tutorial for X–Windows : Version 2.0”, Cambridge, England, 1993. http://www.daimi.au.dk/designCPN/man/

[10] L. Wells, “CPN–Tools : Computer Tool for Coloured Petri Nets” , Version 2, Fri 06 Dec 2002 Department of Computer Science University of Aarhus, Denmark, 2002. http://www.daimi.au.dk/CPNTools/