An Automatic Petri-net Generator for Modeling Multi-agent Systems

Meriem Taibi, Malika Ioualalen, Riad Abdmeziem
LSI - USTHB
Algiers, Algeria
emails: {taibi,ioualalen,abdmeziem}@lsi-usthb.dz

***Abstract*— A multi-agent system can be studied as a concurrent, asynchronous, stochastic and distributed computer system. These characteristics of multi-agent systems make them also a discrete-event dynamic system; it is, therefore, important to analyze the behavior of such system to ensure that it terminates correctly and satisfies other important properties. Several analytical methodologies have been used to study multi-agent system, particularly Petri nets. Petri nets have a well-defined mathematical structure that can be leveraged to provide formal analysis on discrete-event systems. In this work, we propose an automatic transformation to model multi-agent systems using Colored Petri nets.**

Keywords-Multi-agent system; Colored Petri net; Modeling; Description language.

I. INTRODUCTION

Multi-agent systems have been widely studied in the past few decades, where several frameworks have been defined in order to apply the multi-agent system concept to different applications in control and optimization of complex systems [1][2]. An agent is a computer system or computer program that presents several complex characteristics. A Multi-Agent System (MAS) [3] consists of a set of agents, interacting to achieve a common goal. Generally, MAS are known to work properly in a dynamic large-scale complex environment (open environment), thanks to several properties like: autonomy, adaptability, robustness and flexibility. The complexity and capabilities of a multi-agent system are greater than those presented in distributed software systems. In both cases, the study of system properties is becoming more important due to the fact that we are faced more and more to deal with large complex dynamic systems.

Tests and simulations have contributed for a long time to validate such systems. However, these techniques allow to investigate just a part of the global behavior. By that, they differ from the formal verification techniques, which ensure that a property is verified by all possible system executions.

Therefore, the important challenge in this field is the development of analytical methods to assess key properties of such systems. Such methods could be used to impart a preliminary analysis of the multi-agent system, providing design and operation feedback before the development of expensive systems. Many models based on $Belief - Desire - Intention$ (BDI) architecture were proposed in [4] and [5].

Other works include various attempts to deliver a formal model from AUML (Agent Unified Modelling Language) diagrams [6][7]. The advantage of these methods is that most developers are familiar with the (A)UML and an automatic transformation of their diagrams into formal models and model-check them, would greatly simplify the software quality control. The difficulty is that AUML diagrams allow much more freedom for the designer than formal models and the automatic translation is not trivial.

Petri nets have a well-defined mathematical structure that can be leveraged to provide formal analysis on discrete-event systems. In addition, Petri nets have been successfully used in several areas for the modeling and analysis of distributed systems [8].

Several studies have been proposed to model MAS with Petri nets (PN). In [9], a model was proposed for a promotional game of viral marketing on the Internet. Specifically, authors used stochastic Petri nets for modeling a multi-agent wish list. As well, Gazdare [10] used Colored Petri nets (CPN) as a formal method to model a transport system with containers, then, simulate and solve the storage problem. In EL Fallah-Seghrouchni [11], Boukredera [12] and Khosravifar [13], authors also proposed to use the CPN formalism to model interaction protocols.

In this work, we propose an automatic transformation for modeling multi-agent systems. This automation is based on two steps: first, the system is described using a language called MASDL, then, a set of transformation rules are applied to obtain the CPN models.

This document is organized as follows. Motivations and the problem statement are presented next. Section III gives an introduction to MAS. Section IV presents the main aspects of the language we define to specify MAS. In Section V, we present our transformation algorithm allowing to model MAS using CPN. Section VI presents our application, and finally, Section VII discusses the obtained results and presents future work.

We assume that the reader is familiar with Colored Petri net [14].

II. MOTIVATIONS

The Petri nets can be considered as graphic and mathematical tools of modeling and analyzing the discreet system, particularly the competitive, parallel and non-determinist ones. In the field of MAS, the previous works of the Petri nets concentrated on their uses and not on the creation of the new tools and platforms. The goal of our work is to develop a platform which generates automatically models for multi-agent system using CPN. The system in question must be described in an intermediate language. We find in literature two classes of specification languages [15][16]. The first allows the definition of agent and its behavior (e.g., *AgentSpeak* [17])

 ISBN: 978-1-61208-304-9

and the other describes the system environment (e.g., ELMS [18]). Therefore, the definition of a new language including both aspects is necessary. We propose, then, a new language based on XML. The use of XML has many advantages:

- **Universality**: The adoption of a simple and powerful syntax which allows the representation of the most generic models with hierarchical elements, attributes and textual content.
- **Interoperability**: Thanks to their universal syntax, XML documents are easily transportable and readable between systems.
- **Independence between models and data**: We can write an XML document without resorting ever to a schema. If we need to validate the document, we can build a schema afterward.

III. MULTI-AGENT SYSTEMS (MAS)

According to Weiss [19], agents are computational systems situated in some environment, and are capable of autonomous action in this environment in order to meet their design objectives. Agents perceive and interact with each other via the environment, and they act upon it, so that it reaches a certain state where their goals are achieved. Consequently, the MAS environment consists of a set of states $S = \{s_1, s_2, ...\}$, where an agent can undertake a set of actions $A = \{a_1, a_2, ...\}$ and perceive a set of percepts $P = \{p_1, p_2, ...\}$. Therefore, environment modelling is an important issue in the development of multi-agent systems. Although, some multi-agent systems may be situated in an existing environment, in agent-based simulations, the environment is necessarily a computational process too, so modelling multi-agent environments is always an important issue. For this objective, we present in the next section Multi-Agent System Description Language, a language used for the specification of multi-agent environments.

IV. MULTI-AGENT SYSTEM DESCRIPTION LANGUAGE

In this section, we introduce the main aspects of the language we defined for the specification of the multi-agent system and its environment. The language is called Multi-Agent System Description language (MASDL). MASDL is inspired from the Environment Description Language for Multi Agent Simulation (ELMS) language [18], which is an XML-based language that provides the ability to describe multi-agents.
The syntax and the various components of our language are given below. The validation of the syntax is done using World Wide Web Consortium (W3C) scheme which is a grammar defined in XML formalism.

- **MAS general structure:** MAS specification contains the name of the system, a list of agents, a set of objects (system environment), a list of states (agents states and objects states) and finally a list of actions may be performed by agents. The code sample Listing 1 gives the general structure of the system.

Listing 1: **MAS description structure**

```
<MAS NAME = "">
        <AGENTS_LIST>
                <AGENT NAME = "">
                </AGENT>
        </AGENTS_LIST>
        <OBJECTS_LIST>
                <OBJECT NAME = "">
                <OBJECT></OBJECTS_LIST>
        <STATES_LIST>
          <AGENT_STATE_LIST></AGENT_STATE_LIST>
          <OBJECT_STATE_LIST></OBJECT_STATE_LIST>
        </STATES_LIST>
        <ACTIONS_LIST>
                <ACTION NAME = ""></ACTION>
        </ACTIONS_LIST>
</MAS>
```

- **Agent description:** This description contains the name of the agent, a list of its attributes (agent properties), the current state and list of actions. The following example in Listing 2, defines an agent named *agent1* which has an attribute *prop1* of type *type1* with a value *val1*. The *agent1* has *state_agent1* like initial state and can perform *action1* and *action2*.

Listing 2: **Agent description example**

```
<AGENT NAME = "agent1">
    <ATTRIBUTES>
        <ATT NAME= "prop1"
             TYPE="type1"
             VALUE = "val"/>
    </ATTRIBUTES>
    <CURRENTSTATE>
        <ITEM NAME = "state_agent1"/>
    </CURRENTSTATE>
    <ACTIONS>
        <ITEM NAME = "action1"/>
         <ITEM NAME = "action2"/>
    </ACTIONS>
</AGENT>
```

- **Resources description** This concept allows the specification of the different objects in the MAS environment (all the entities in the environment that are not agent). An object class includes its name, the current state of the object, its identifier and the available quantity (a negative amount is used in case where the amount is unlimited).

Listing 3: **Example of object description**

```
<OBJECTS_LIST>
    <OBJECT NAME = "Objet1">
        <ATTRIBUTES>
            <ATT NAME= "quantity"
                 TYPE= "int"
                 VALUE="100"/>
        </ATTRIBUTES>
        <CURRENTSTATE>
            <ITEM NAME = "state_res"/>
        </CURRENTSTATE>
    </OBJECT>
</OBJECTS_LIST>
```

The code sample Listing 3 above, defines a resource called *object1*. This object has an integer attribute representing the number of units available in the system and current state *state_res*.

 ISBN: 978-1-61208-304-9

- **State description** The state is defined in the tag *<AGENT_STATE_LIST>* when it relates to agents and in the tag *<OBJECT_STATE_LIST>* when it concerns resources. A state is described by its name and an informal description given by the designer, it corresponds to the semantics of the state. A state represents a situation in which an agent or a resource can be, during the running of the system. In Listing 4, the code exemplifies definition of two states (one for agent state and the second for resource one).

Listing 4: **Description of states**

```
<STATES_LIST>
        <AGENT_STATE_LIST>
                <STATE NAME = "state_agent1">
                <DESCRIPTION></DESCRIPTION>
                </STATE>
        </AGENT_STATE_LIST>
        <OBJECT_STATE_LIST>
                <STATE NAME = "state_res">
                <DESCRIPTION></DESCRIPTION>
                </STATE>
        </OBJECT_STATE_LIST>
</STATES_LIST>
```

- **Action description** The description of the action includes its name, its content and an informal description. *Content* specifies the agents that are involved in the execution of the action and also the potential resources (objects) needed. The agents specification includes their name, input and output states. An action can be executed by an agent, if it is in the defined input state. These states represent the preconditions of the action.
Resources can also be instantiated or removed by action. The specification of resources including their type, input and output states and the number of units to subtract (*sub_quantity_entry*) or to add (*add_quantity_exit*).

Listing 5: **Action description**

```
<ACTIONS_LIST>
        <ACTION NAME = "action1">
         <CONTENT>
                <ACTIONS_AGENT>
                        <ACTION_ITEM NAME= "Ag"
                        ENTRYSTATE ="state1_Ag"
                        EXITSTATE="state2_Ag"/>
                </ACTIONS_AGENT >
                <ACTIONS_OBJECT>
                        <ACTION_ITEM NAME= "Res"
                        ENTRYSTATE="state1_Res"
                        EXITSTATE="state2_Res"
                        SUB_QUANTITY_ENTRY= "2"
                        ADD_QUANTITY_EXIT="5"/>
                </ACTIONS_OBJECT>
        </CONTENT>
        <DESCRIPTION>
                <!-- Informel Description -->
        </DESCRIPTION>
  </ACTION>
</ACTIONS_LIST>
```

In the example above Listing 5, an action named *action1* is defined and has as a precondition: the agent *Ag* must be in the state *state1_Ag* and the object *Res* in the state *state1_Res*. As a result of the execution of this action, the agent *Ag* will be in the output state *state2_Ag* and the resource *Res* in *state2_Res* state with the production of three units of this resource.

V. Automatic Multi-Agent Modeling Using CPN

The objective of this section is to give an algorithm allowing to transform a description of multi-agent system to Colored Petri net models. The CPN models obtained are written in a XML based language with a specific syntax which we call *Petri Net Description Language* (PNDL).

A. Transformation algorithm

The transformation algorithm 1 allows to generate automatically CPN models of the described system. The important steps of the algorithm are given in Fig. 1. Based on MASDL language, the system is defined by a set of states $S = \{s_1, s_2, ...\}$ and agent is able to perform a set of actions $A = \{a_1, a_2, ...\}$. The execution of an action causes changes in the environment.

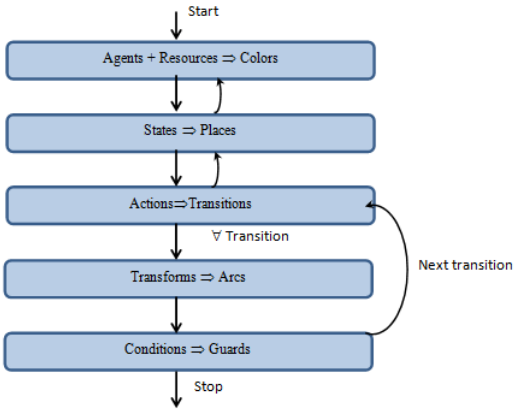

Fig. 1: Schema of the modeling process.

- **Algorithm assumptions:** We assume that our algorithm has as input and output data the following sets, described in the Fig. 2, which are calculated from the MASDL system specification.

Name	Description
AG	Set of agent
RE	Set of resources
SA	Set of agents states
SR	Set of resources states
AC	Set of actions
P	Set of places
T	Set of transitions
Arc	Set of Arcs
C_1	Color with the structure $< id, state >$
C_2	Color with the structure $< id, state, quantity >$

Fig. 2: Algorithm's input and output data

The algorithm also uses a set of predefined functions, the definition of which is as follows:

 ISBN: 978-1-61208-304-9

- $Act : AG \rightarrow AC$: *Act(a)* allows to give all the actions which can be made by the agent a,
- $Act_R : RE \rightarrow AC$: $Act_R(r)$ calculates all actions that affect the resource r,
- $Entry\_Agent : AG \otimes AC \rightarrow SA$: Returns the entry state of one agent to undertake an action
- $Entry\_object : RE \otimes AC \rightarrow SR$: Returns the entry state of one resource to undertake an action states,
- $Exit\_Agent : AG \otimes AC \rightarrow SA$:Returns the exit state of one agent after action execution,
- $Exit\_object : RE \otimes AC \rightarrow SR$: Returns the exit state of one resource after action execution,
- $Sub\_Object : RE \otimes AC \rightarrow N$: Gives the number of units to subtract from one resource after action execution,
- $Add\_Object : RE \otimes AC \rightarrow N$: Gives the number of units to add to one resource after action execution,
- $Create\_Place()$: Allow to create places,
- $Create\_Transition()$: Allows to create transitions,
- $Create\_Arc()$: Creates arcs connecting places to transitions or vice versa.

Algorithm 1 Petri net Generator

$P \leftarrow \oslash$
$T \leftarrow \oslash$
for each $sa \in SA$ **do**
  $Create\_Place(p_{sa})$
  $P \leftarrow P \cup p_{sa}$
end for
for each $sr \in SR$ **do**
  $Create\_Place(p_{sr})$
  $P \leftarrow P \cup p_{sr}$
end for
for each $c \in Act(a)$ **do**
  $Create\_Transition(t_a)$
  $T \leftarrow T \cup t_a$
end for
for each $a \in AG$ **do**
  for each $c \in Act(a)$ **do**
    $sa \leftarrow Entry\_Agent(a,c)$
    $sa' \leftarrow Exit\_Agent(a,c)$
    $Create\_arc(p_{sa},t_a)$ with color function $1/ < a,sa >$
    $Create\_arc(t_a,p_{sa'})$ with color function $1/ < a,sa' >$
  end for
end for
for each $r \in RE$ **do**
  for each $c \in Act_R(r)$ **do**
    $sr \leftarrow Entry\_Agent(r,c)$
    $sr' \leftarrow Exit\_Agent(r,c)$
    $Create\_arc(p_{sr},t_c)$ with color function $Sub\_Object(r,c)/ < r,sr,quantity >$
    $Create\_arc(t_a,p_{sa'})$ with color function $Add\_Object(r,c)/ < r,sr',quantity >$
  end for
end for

- **Initial marking:** the initial state is calculated as:
  1) If an agent a is initially in the state sa, we put one token of color $< a,sa > \in C_1$, in the place p_{sa};
  2) If an resource r is initially in the state sr, we put one token of color $< r,sr,quantity > \in C_2$, in the place p_{sr};

B. Output format

We propose an XML-based language for the description of the models generated by the algorithm. We entitle our language Petri Net Description Language, which is based on the tags $< PLACES >$, $< TRANSITIONS >$ and $< ARCS >$ to describe the model and on $< COLORS >$ and $< TOKENS >$ to give its marking. The general structure of the language is presented in the following Listing 6:

Listing 6: Description of CPN Model

```
<RDPC NAME = "RdP_Example">
        <PLACES>
                <PLACENAME = "p1"/>
        </PLACES>
        <TRANSITIONS>
                <TRANSITIONNAME = "t1"/>
        </TRANSITIONS>
                <ARCS>
                <PRE_ARCS>
                        <ARC FROM "p1" TO "t1">
                        <WEIGHT COLOR = "c1" PRE = "1"/>
                        </ARC>
                </PRE_ARCS>
                <POST_ARCS>
                        <ARC FROM "t1" TO "p1">
                        <WEIGHT COLOR = "c1" POST = "1"/>
                        </ARC>
                </POST_ARCS>
        </ARCS>
        <COLORS>
                <COLOR NAME = "c1">
                        <ITEM NAME = "id" VALUE = "01"/>
                </COLOR>
        </COLORS>
        <TOKENS>
                <TOKEN COLOR = "c1" PLACE = "p1"/>
        </TOKENS>
</RDPC>
```

VI. RUNNING MASDL ENVIRONMENT

The use of XML provides various advantages, wide range of XML tools are currently available and it can be useful for the future development. The validation of the description is done using W3C scheme. For the implementation of our tool, we chose Java, which allows us to use Java Architecture for XML Binding (JAXB) and Application Programming Interface (API) to create XML application data. The global architecture of our application is shown in Fig. 3.

Our tool allows users to introduce environment specifications from a graphical interface, as shown in Fig. 4. With this interface, users do not need to deal with the language syntax but just fill the different fields.

Filled fields will be checked and compiled to generate the corresponding XML file, as shown in Fig. 5, on which the transformation algorithm will be applied. To generate a

 ISBN: 978-1-61208-304-9

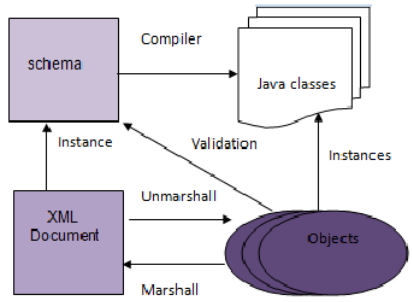

Fig. 3: The global architecture

graphical representation of the Petri net model, as shown in Fig. 6, we use the GraphViz tool [20].

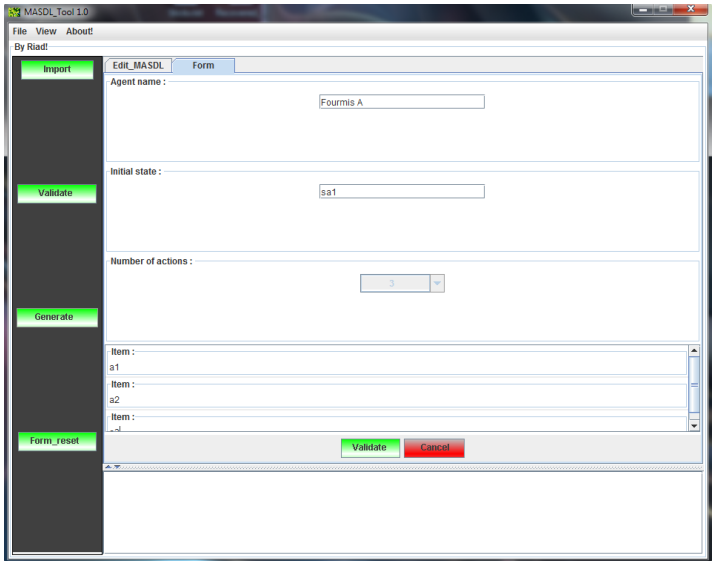

Fig. 4: Tool interface

VII. Conclusion and Future Works

In this paper, we introduced the MASDL language, used for the specification of the agents and their environment. The language is based on XML and is independent of the agent runtime platform and implementation language. We defined also transformation rules to obtain formal models from the system specification to analyze and verify the described multi-agent system. We plan in our further work to connect our tool to another verification tool, as CPN Tool or GreatSPN for the general properties verification (deadlock, boundedness, etc.). We will focus mainly on extending our model by introducing the temporal dimension in order to perform a quantitative analysis and compute MAS system performances (average waiting time, average resources available, etc.).

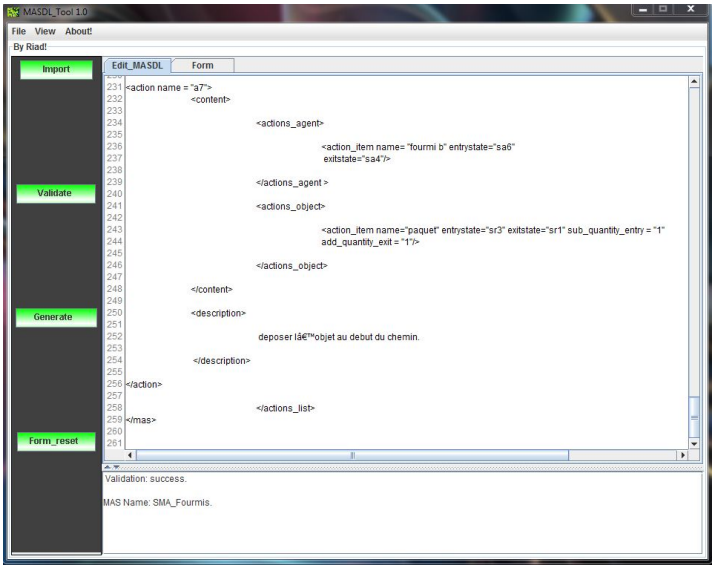

Fig. 5: MASDL file exemple

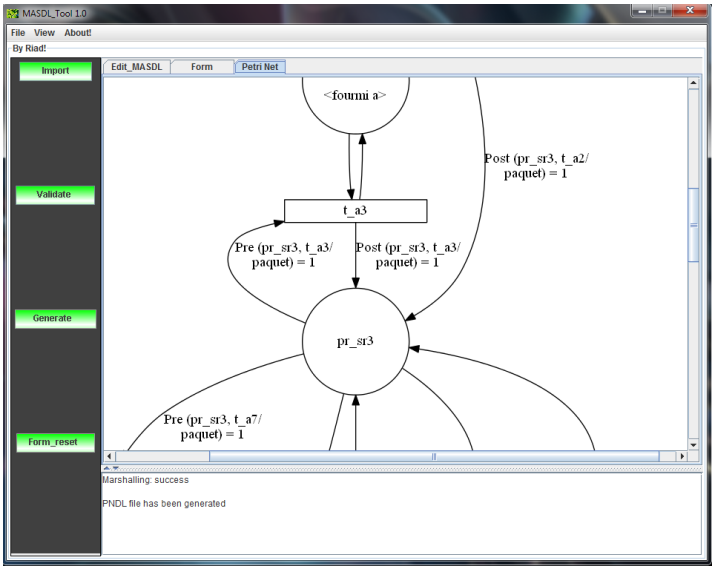

Fig. 6: Petri net generation

References

[1] M. Greaves, V. Stavridou-Coleman, and R. Laddaga, “Guest editors’ introduction: Dependable agent systems,” *IEEE Intelligent Systems, NJ, USA*, vol. 19, September 2004, pp. 20-23.

[2] S. S. Heragu, R. J. Graves, B.-I. Kim, and A. St Onge, “Intelligent agent based framework for manufacturing systems control,” *Trans. Sys. Man Cyber. Part A, NJ, USA*, vol. 32, no. 5, September 2002, pp. 560-573.

[3] J. Ferber, *Les systèmes multi-agents vers une intelligence collective*. Inter-Editions, 1995.

[4] H. Mouratidis, M. Kolp, P. Giorgini, and S. Faulkner, “An architectural description language for secure multi-agent systems,” *Web Intelli. and*

 ISBN: 978-1-61208-304-9

Agent Sys., Amsterdam, The Netherlands, vol. 8, no. 1, January 2010, pp. 99-122.

[5] M. Dziubiński, "Complexity of multiagent bdi logics with restricted modal context," in *The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 3*, ser. AAMAS '11. International Foundation for Autonomous Agents and Multiagent Systems, Taipei, Taiwan, 2011, pp. 1171–1172.

[6] S. Maalal and M. Addou, "A new approach of designing multi-agent systems," *CoRR*, vol. abs/1204.1581, 2012.

[7] L. J. B. Ayed and F. Siala, "Event-b based verification of interaction properties in multi-agent systems," *JSW*, vol. 4, no. 4, 2009, pp. 357-364.

[8] J. R. Celaya, A. A. Desrochers, and R. J. Graves, "Modeling and analysis of multi-agent systems using petri nets," *JCP*, 2009, pp. 981-996.

[9] C. Balague, "Multi-agent system in marketink: Modelisation by petri net," Ph.D. dissertation, École des Hautes études Commerciales, 2005.

[10] M. K. Gazdare, "Heuristic optimization of distributed problem storage containers in port," Ph.D. dissertation, ECOLE CENTRALE DE LILLE, 2008.

[11] A. El Fallah-Seghrouchni, S. Haddad, and H. Mazouzi, "Protocol engineering for multi-agent interaction," in *International Workshop on Modeling Autonomous Agents in a Multi-Agent World (MAAMAW), Valencia, Spain*, 1999.

[12] D. Boukredera, S. Aknine, and R. Maamri, "Modeling temporal aspects of contract net protocol using timed colored petri nets," in *STAIRS*, December 2012, pp. 83–94.

[13] S. Khosravifar, "Modeling multi agent communication activities with petri nets," *International Journal of Information and Education Technology, Singapore*, vol. 3, no. 3, September 2013, pp. 310–3014.

[14] K. Jensen, *Coloured Petri nets: basic concepts, analysis methods and practical use, vol. 2*. London, UK, UK: Springer-Verlag, 1995.

[15] M. M. Dastani, C. M. Jonker, and J. Treur, "A requirement specification language for configuration dynamics of multi-agent systems," *International Journal of Intelligent Systems.*, vol. 19, 2004, pp. 277–300.

[16] M.-P. Huget, "Agent uml notation for multiagent system design," *IEEE Internet Computing, Piscataway, NJ, USA*, vol. 8, no. 4, 2004, pp. 63–71.

[17] A. S. Rao, "Agentspeak(l): Bdi agents speak out in a logical computable language," in *Proceedings of the 7th European workshop on Modelling autonomous agents in a multi-agent world : agents breaking away: agents breaking away*, ser. MAAMAW '96. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1996, pp. 42–55.

[18] F. Y. Okuyama, R. H. Bordini, and A. C. da Rocha Costa, "Elms: an environment description language for multi-agent simulation," in *Proceedings of the First international conference on Environments for Multi-Agent Systems*, ser. E4MAS'04. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 91–108.

[19] G. Weiss, Ed., *Multiagent systems: a modern approach to distributed artificial intelligence*. Cambridge, MA, USA: MIT Press, 1999.

[20] Graphviz - graph visualization software. http://www..graphviz.org. Retrieved: June, 2013.

 ISBN: 978-1-61208-304-9