

# Automated Transformation of Multi-agent Protocols to Coloured Petri Nets

Ashwag Omar Maghraby  
 Computer Science Department  
 Umm Al-Qura university  
 Makkah, Saudi Arabia  
 e-mail: aomaghraby@uqu.edu.sa

**Abstract**— As multi-agent protocols are getting more and more complex, analyzing the behavior of such protocols is becoming increasingly important to ensure that they satisfy agents objectives and terminate correctly. This paper presents a tool for automated transformation from multi-agent protocols written in Lightweight Coordination Calculus language to high level Colored Petri nets models. This automation constructs a well-defined mathematical structure model that can be leveraged to formal analysis multi-agent protocol and used with the Standard Functional Programming language to automatically check whether the protocol is understandable and advantageous to the objectives of agents. The benefits of our approach consist in the new approach of analysing the MAS protocol and automatically validate key behavior properties of the MAS protocol to ensure that the protocol satisfies agents objectives.

**Keywords**- *Multi-agent protocol; Colored Petri net; Automated transformation; Protocol analysis.*

## I. INTRODUCTION

In a Multi-Agent System (MAS), two or more agents have to work together to find a final solution and satisfy their individual goals by exchanging messages following interaction protocols. An interaction protocol is a set of rules that direct the communication between several agents [1]. These protocols constrain the possible sequences of messages that may occur in agent interactions and describe how agents should react to messages received during interactions [2]. There are a finite number of messages in transmission and reception for each MAS protocol.

The need to understand, study and analyze MAS protocols properties is growing, as these protocols are becoming more complex due to the rich behavior introduced by concurrency, communication, and uncertainty. In fact, interactions between agents may be affected by different unexpected factors, for example, unexpected message, loss of messages or deviation in the message order.

Coloured Petri Nets (CPNs) [3] and CPN Tool [4] have been widely used to address these challenges. CPNs and CPN Tools provide a graphical representations and a mathematical formalism for the description, construction, execution, formal analyzing, and understanding of distributed and complex MAS protocols. CPNs ensure that a property is verified by all possible protocol executions [5].

In this paper, we propose an automatic transformation from multi-agent protocols written in Lightweight

Coordination Calculus language (LCC) to high level CPNs models. This automation constructs formal and executable models of MAS protocols. It is used with the Standard Functional Programming language (SML is a general-purpose, modular, functional programming language with compile-time type checking and type inference [6]) to automatically validate key behavior properties of the protocol and to ensure that the protocol satisfies agents objectives and terminates correctly. This approach is divided into three main steps: (1) automated transformation LCC protocol to CPNs model; (2) construction of state space; (3) automated comparing of the agent's objectives properties and the behavioral properties of the LCC protocol.

The rest of this paper is organized as follows. Section II gives an overview of the CPNs, and how to use it to model agent protocol. Section III gives an overview of MAS protocol language (LCC). Section IV describes our approaches. Section V describes the automated transformation from the LCC protocol into an equivalent CPNs model. Section VI highlights the construction of state space approaches. Section VII details the verification approach and Section VIII gives an example of our approach.

## II. CPNS AND USING CPNS TO MODEL MAS PROTOCOLS

CPNs used in a large variety of different areas such as MAS communication protocols. It provides a framework for the construction and analysis of these protocols. A CPN model of a protocol describes the states of the protocol and the transitions between these states [3]. A brief introduction to CPNs is presented in this section.

### A. CPNs

The CPN model consists of four elements [7][8]: data, place, transition, and arc which describe the net structure of the CPN model. An example of a CPN modelled in the CPN tool is depicted in Figure 1. This model has:

1) *Three colour sets (Topic, Message and Role):* A colour set can be a basic colour set (integer, string, real and Boolean) or a product of colour sets or a combination of other colour sets (a declared colour set from already declared colour sets). Colour sets are used to declare variables, other colour sets, functions, operations, constants and a place's inscription. A token is associated with a colour set and has data values (token colours) attached to it.

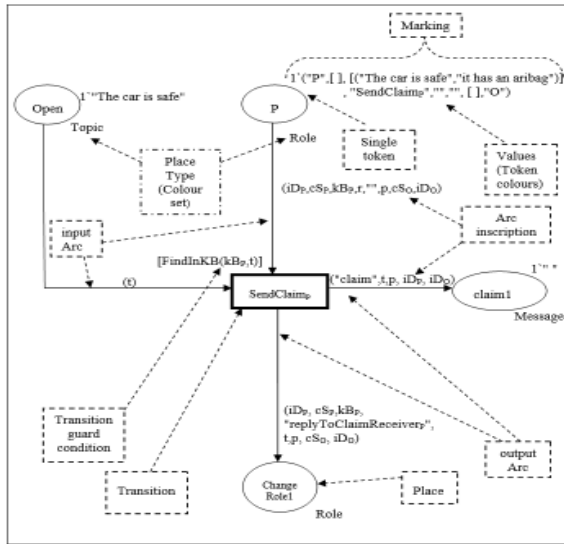


Figure 1. CPNs Model Elements Example

2) *Four places (Open, P, claim1 and ChangeRole1)*: A place is a location (drawn as ellipse). It is used to hold data items (tokens). Tokens must match the place type (colour set). A place is associated with a marking, which indicates the number of stored tokens and the value (token colours) of these tokens. The state of the CPN model, at a particular moment, is represented by the set of markings of all the places.

3) *One transition called SendClaimp*: A transition is an activity, which represents an event and is drawn as a rectangle. It is used to transform data between places. In practice, transition receives data from one or more places, checks its guard condition, executes its associated code segment, and sends the result to other places. A guard condition is a Boolean expression enclosed in square brackets that appears above the transition rectangle. A code segment is a computer program written in the CPN SML language (in the CPN Tool) or in the other kinds of notations, which has a well-defined syntax and semantic.

4) *Four arcs*: An arc is used to connect a place and a transition and to specify the data flow (the pre- and post-condition relation between transitions). An arc is associated with inscription, which is used to describe how the state of the modelled system changes. In the CPN Tool, an arc inscription is an expression that consists of CPN SML variables, constants and functions.

One of the key features of the CPN is its ability to construct large models in a hierarchical manner [8] by using subpages to build superpages. The subpages interact with each other and with the superpages through a set of transitions and a set of places. In practice, subpages used to model individual agent where superpages used to model communication protocol, which enables the message

exchange among the agents of the protocol and produces a change of the protocol state.

### B. Using CPNs to model MAS protocols

There are a number of works using CPNs to model MAS protocols. In some related work, Calderon [9] developed a tool to transform UML-based systems of two large-scale UML systems [10] to CPN models (Design/CPN XML file) [3][8]. But the CPN models generated by the tool are not ready for analysis. The user needs to perform some manual work to get an executable CPN model and to be able to verify the correctness of the generated CPN. Another difference between our verification tool and Calderon's tool is that in the Calderon's approach, the dynamic behavior of the system is analyzing by running the Design/CPN tool simulator, while in our approach, the dynamic behavior of the system is analyzing by using state space techniques and the CPN SML language.

Suriadi's et al. [5] used the CPN Tool to model one case study of the Privacy Enhancing Protocols (PEPs) called the Private Information Escrow Bound to Multiple Conditions Protocol (PIEMCP) manually. Then, this work used the state space techniques, CPN SML language and session-data files to model validation and verification of the PIEMCP. The similarity between our verification approach and Suriadi's et al. approach [5] is that both use the state space techniques, CPN SML language and files (the session-data file in Suriadi's et al. approach and the properties file in our work). However, the main difference between our verification approach and this approach is that Suriadi's et al. approach generates a CPN model from a PIEMCP system model manually, while our approach generates a hierarchical CPN model from an LCC protocol by using a set of transformational rules automatically.

## III. AGENT PROTOCOL DEVELOPMENT LANGUAGE

The Lightweight Coordination Calculus (LCC) [11] is a declarative, process calculus-based, executable specification language for choreography [12], which is based on logic programming and is used for specifying the message-passing behavior of MAS interaction protocols.

### A. LCC Syntax

The abstract syntax of an LCC clause [11] is shown in Table 1. In an LCC framework, each of the  $N \geq 2$  agents is defined with a unique identifier *Id* and plays a *Role*. Each agent, depending on its *Role*, is assigned an LCC protocol.

An LCC protocol can be recursively defined as a sequential composition (denoted as *then*) or choice (denoted as *or*) of LCC protocols. In an LCC protocol, agents can change roles, exchange (receive or send) messages and exit the dialogue under certain constraints *C* ( $\text{null} \leftarrow C$ ). Null represents an event (a do-nothing event) that does not involve role changing or message exchanging. A constraint is defined as a propositional formula specified over *terms* connected by *or* and *and* operators.

TABLE I. THE ABSTRACT SYNTAX OF LCC

|               | Meaning   |
|---------------|---|
| Framework :=  | {Clause,.....}  |
| Clause :=     | Agent ::= Dn  |
| Agent :=      | a(Role, Id)   |
| Dn :=         | Agent   Message   null $\leftarrow$ Constraint   Dn then Dn   Dn or Dn                            |
| Message :=    | M => Agent   M => Agent $\leftarrow$ Constraint   M <= Agent   Constraint $\leftarrow$ M <= Agent |
| Constraint := | Term   Constraint and Constraint   Constraint or Constraint                                       |
| Role :=       | Term  |
| M :=          | Term  |
| Term:=        | Constant (Argument,.....)   |
| Id            | Constant   Variable   |
| Constant      | Character sequence made up of letters or numbers beginning with a lower case letter               |
| Variable      | Character sequence made up of letters or numbers beginning with an upper case character           |
| Argument      | Term   Constant   Variable  |

Messages  $M$  are the only way to exchange information between agents. An agent can send a message  $M$  to another agent ( $M \Rightarrow Agent$ ), and receive a message from another agent ( $M \Leftarrow Agent$ ). There are two types of constraints over the messages exchanged: pre-condition and post-condition. Pre-conditions ( $M \Rightarrow Agent \leftarrow C$ ) specify the required conditions for an agent to send a message. Post-conditions ( $C \leftarrow M \Leftarrow Agent$ ) explain the states of the receiver after receiving a message.

B. LCC Examples

This is the simplest example of a persuasion protocol between two agents  $P$  and  $O$ .  $P$  and  $O$  have arguments for and against  $Topic$ . Agent  $P$  sends a *claim* message  $Topic$  and agent  $O$  receives this *claim* message  $Topic$ . A fragment of LCC protocol for the interchange in this argument is:

```

a(R1,P)::=
  claim(Topic) => a(R2, O)
  then
    a(R3,P).
a(R2,O)::=
  claim(Topic) <= a(R1, P)
  then
    a(R4,O).
    
```

This is read as: role  $R1$  of agent  $P$  sends a claim message to the role  $R2$  of agent  $O$  and role  $R2$  of agent  $O$  receives the claim message from role  $R1$  of agent  $P$ . Then  $P$  changes its role to  $R3$  and  $O$  changes its role to  $R4$ .

IV. AUTOMATED TRANSFORMATION APPROACH

Our automated transformation approach can answer the following question: Does the LCC MAS interaction protocol satisfy the agent’s objectives (behavior properties) and terminates correctly? Three steps are needed to answer this question:

- 1) Transforming the LCC protocol into an equivalent CPNs model. This step is processed in a fully automatic way;
- 2) Constructing the state space from the generated CPNs model.
- 3) Comparing the agent’s objectives properties and the behavioral properties of the LCC protocol using CPN SML

functions. A positive (negative) result indicates that a specific property is satisfied (unsatisfied).

The following sections discuss the details of each of these three steps.

V. STEP ONE: AUTOMATED TRANSFORMATION FROM LCC TO CPNS MODEL

Given the LCC interaction protocol as an input, the automated tool transforms the LCC protocol into an equivalent CPNs model using a set of transformational rules. In our approach, CPN model is described as CPNXML file. A CPNXML file [13] is an extended markup language (XML) document that describes the modelling elements of the CPN model.

We have developed a step-by-step technique that allows the user to automatically transform an LCC protocol into the CPNXML file by:

- 1) Declaring colour sets and functions.
- 2) Generating a CPN subpage for each LCC role. Each subpage represents a role behavior .
- 3) Connecting all the CPN subpages for each individual agent by generating one CPN superpage. CPN superpage describes the interaction between roles, where the messages that are passed between two roles determine the interaction between the subpages of the two roles.

In practice, to automate the transformation process from an LCC protocol into CPNXML file we use 12 LCC-CPNXML tables (9 tables to generating CPN subpage and 3 tables to generate CPN superpage), where transitions and places are connected according to a set of transformation rules. The use of LCC-CPNXML tables makes the transformation faster and the resulting CPN model can be executed with data and analyzed, not only by our tool, but also by other users (using CPN Tool) since CPN has a comprehensible graphical representation. The following subsections give more details of the transformation process from an LCC protocol into CPNXML file.

A. Declaration of Colour Sets and Functions

Many communication between agents will be dialogues, and will specify more than two roles. In this approach, we use three different primary types of colour sets: *TOPIC*, which is used to model the main dialogue topic; *Message*, which is used to model messages arguments and *Role*, which is used to model role arguments.

Each agent has a knowledge base  $KB$  (private knowledge) and a commitment store  $CS$  (common knowledge). During the agent interactions, the agents take turns to send messages. Each agent makes his choice between possible messages depending on its  $CS$  and  $KB$ . In practice, the  $CS$  is continuously updated after sending or receiving each message by either adding to or subtracting from its argument. For that reason, we defined thirteen different basic functions, which are used to find, get, add or subtract an argument from either a  $CS$  or  $KB$  list. These functions are written in the CPN SML language [7].

The CPNXML format of the three types of colour sets and thirteen functions are saved in the Global Declaration file called "CPNmainCode". The user does not need to know about these colour set types or functions unless he/she needs to define new types or functions. For more information about how to define new CPN SML colour set types or functions, please read [4][8].

**B. Generation of a CPN Subpage**

Nine tables are used to automate the transformation process from LCC roles into CPN subpages. Space limitations prevent us from presenting each and every one of those tables instead we will discuss LCC Message Sending Statement table, which gives more details of the transformation process from an LCC message sending statement into CPNs model. The LCC message sending code is transformed into a high level CPN model by creating (as shown in Table II):

1) One new transition where the transition ID = unique identifier, the transition name= "Send" + Message name, and guard condition = LCC message Boolean conditions (line 1 to 7 of Table II);

2) One new place where the place ID = unique identifier, the place name = message name, place colour set type = Message and place (port) type= Out (line 8 to 19 of Table II);

3) One arc (output arc), which is used to connect the new transition to the new place, where the arc ID = unique identifier, the arc type= TtoP (output arc), the transition ID reference = the new transition ID, the place ID reference = the new place ID, the arc inscription = (Message arguments) (line 20 to 28 of Table II).

TABLE II. LCC-CPNC+XML TRANSFORMATION TABLE (SEND A MESSAGE)

| LCC Code<br>(Send a Message)      | Message(Topic) => a(RoleName(Arguments),AgentID)<br>← Conditions   |
|-----------------------------------|--|
| CPNs Model                        | CPNXML Structure   |
| <p><i>Send message symbol</i></p> | <ol style="list-style-type: none"> <li>&lt;trans id="ID1423689023"&gt;</li> <li>&lt;text&gt; "Send"+ message_name &lt;/text&gt;</li> <li>&lt;cond&gt;</li> <li>&lt;text tool="CPN Tools" version="2.9.11"&gt;</li> <li>"LCC Boolean conditions" &lt;/text&gt;</li> <li>&lt;/cond&gt;</li> <li>&lt;/trans&gt;</li> <li>&lt;place id="ID1423689035"&gt;</li> <li>&lt;text&gt; Message_name &lt;/text&gt;</li> <li>&lt;type id="ID1423689036"&gt;</li> <li>&lt;text tool="CPN Tools" version="2.9.11"&gt;</li> <li>Message &lt;/text&gt;</li> <li>&lt;/type&gt;</li> <li>&lt;initmark id="ID1423689037"&gt;</li> <li>&lt;text tool="CPN Tools" version="2.9.11"&gt;</li> <li>&lt;/initmark&gt;</li> <li>&lt;port id="ID1424205036" type="Out"&gt;</li> <li>&lt;/port&gt;</li> <li>&lt;/place&gt;</li> <li>&lt;arc id="ID1423689049"</li> <li>orientation="TtoP" order="1"&gt;</li> <li>&lt;transend idref="New transition ID"/&gt;</li> <li>&lt;placeend idref="New place ID"/&gt;</li> <li>&lt;annot id="ID1423689050"&gt;</li> <li>&lt;text tool="CPN Tools" version="2.9.11"&gt;</li> <li>Message arguments &lt;/text&gt;</li> <li>&lt;/annot&gt;</li> <li>&lt;/arc&gt;</li> </ol> |

TABLE III. LCC-CPNC+XML TRANSFORMATION TABLE (ROLE IN THE CPN SUPERPAGE)

| LCC Code<br>(Role)        | a(RoleName(Arguments, Topic),AgentID)  |
|---------------------------|--|
| CPNs Model                | CPNXML Structure   |
| <p><i>Role symbol</i></p> | <ol style="list-style-type: none"> <li>&lt;trans id="ID1414172135"&gt;</li> <li>&lt;text&gt; Role_Name &lt;/text&gt;</li> <li>&lt;subst subpage= "Corresponding subpage ID"</li> <li>portsock= "(socket ID, Port ID)"</li> <li>&lt;subpageinfo id="ID1414172175"</li> <li>name= "Corresponding subpage Name"&gt;</li> <li>&lt;/subpageinfo&gt;</li> <li>&lt;/subst&gt;</li> <li>&lt;/trans&gt;</li> <li>&lt;arc id="ID1423689049"</li> <li>orientation="TtoP" order="1"</li> <li>&lt;transend idref="Substitution transition ID"/&gt;</li> <li>&lt;placeend idref="Related socket ID"/&gt;</li> <li>&lt;annot id="ID1423689050"&gt;</li> <li>&lt;text tool="CPN Tools" version="2.9.11"&gt;</li> <li>Socket arguments (e.g. Role arguments, Message arguments)</li> <li>&lt;/text&gt;</li> <li>&lt;/annot&gt;</li> <li>&lt;/arc&gt;</li> </ol> |

TABLE IV. LCC-CPNC+XML TRANSFORMATION TABLE (DIAGLOUE TOIC)

| LCC Code<br>(Dialogue Topic Argument) | a(RoleName(Arguments, Topic),AgentID)  |
|---------------------------------------|--|
| CPNs Model                            | CPNXML Structure   |
| <p><i>Dialogue Topic symbol</i></p>   | <ol style="list-style-type: none"> <li>&lt;place id="ID1423689035"&gt;</li> <li>&lt;text&gt; OpenDialogue &lt;/text&gt;</li> <li>&lt;type id="ID1423689036"&gt;</li> <li>&lt;text tool="CPN Tools" version="2.9.11"&gt;</li> <li>Topic &lt;/text&gt;</li> <li>&lt;/type&gt;</li> <li>&lt;initmark id="ID1423689037"&gt;</li> <li>&lt;text tool="CPN Tools" version="2.9.11"&gt;</li> <li>&lt;/initmark&gt;</li> <li>&lt;port id="ID1424205036" type="In"&gt;</li> <li>&lt;/port&gt;</li> <li>&lt;/place&gt;</li> <li>&lt;arc id="ID1423689049"</li> <li>orientation="TtoP" order="1"&gt;</li> <li>&lt;transend idref="New substitution transition ID"/&gt;</li> <li>&lt;placeend idref="New place ID"/&gt;</li> <li>&lt;annot id="ID1423689050"&gt;</li> <li>&lt;text tool="CPN Tools" version="2.9.11"&gt;</li> <li>Topic arguments</li> <li>&lt;/text&gt;</li> <li>&lt;/annot&gt;</li> <li>&lt;/arc&gt;</li> </ol> |

**C. Generation of a CPN Superpage**

Three tables are used to automate the generation of one CPN superpage. Each LCC role is transformed into a high-level Petri net by creating (as shown in Table III):

1) One new substitution transition. (line 1 to 9 of Table III);

2) One or more arcs (line 10 to 20 of Table III);

3) If this role is the primary role (the first role in the LCC code, which is responsible for opening the dialogue), then:

a) Create one new place (line 1 to 12 of Table IV);

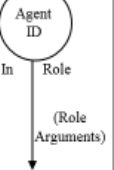
b) Create one arc (input arc) (line 13 to 22 of Table IV).

4) If this role is the agent's primary role, then:

a) Create one new place (line 1 to 12 of Table V);

b) Create one arc (input arc) (line 13 to 23 of Table V).

TABLE V. LCC-CPNC+XML TRANSFORMATION TABLE (AGENT'S STARTER ROLE ARGUMENTS)

| LCC Code (Starter Role Arguments)   | a(RoleName(Arguments, Topic),AgentID)  |
|---|--|
| LCC CPNs Model  | CPNXML Structure   |
| <p>Starter Role argument symbol</p> <p>Arguments initial values</p>  | <ol style="list-style-type: none"> <li>1. &lt;place id="ID1423689035"&gt;</li> <li>2. &lt;text&gt; <i>Agent ID</i> &lt;/text&gt;</li> <li>3. &lt;type id="ID1423689036"&gt;</li> <li>4. &lt;text tool="CPN Tools" version="2.9.11"&gt;</li> <li>5. <i>Role</i> &lt;/text&gt;</li> <li>6. &lt;/type&gt;</li> <li>7. &lt;initmark id="ID1423689037"&gt;</li> <li>8. &lt;text tool="CPN Tools" version="2.9.11"&gt;</li> <li>9. <i>Arguments initial values</i></li> <li>10. &lt;/initmark&gt;</li> <li>11. &lt;port id="ID1424205036" type="In"&gt;</li> <li>12. &lt;/port&gt;</li> <li>13. &lt;/place&gt;</li> <li>14. &lt;arc id="ID1423689049"</li> <li>15. orientation="PtoT" order="1"&gt;</li> <li>16. &lt;transend idref="New_substitution_transition_ID"&gt;</li> <li>17. &lt;placeend idref="New_place_ID"/&gt;</li> <li>18. &lt;annot id="ID1423689050"&gt;</li> <li>19. &lt;text tool="CPN Tools" version="2.9.11"&gt;</li> <li>20. <i>Role arguments</i></li> <li>21. &lt;/text&gt;</li> <li>22. &lt;/annot&gt;</li> <li>23. &lt;/arc&gt;</li> </ol> |

VI. STEP TWO: CONSTRUCTION OF STATE SPACE

The second step of our approach is to construct from the CPN model its state space (directed graph, which represents all possible executions of the CPN model). In the CPN Tool, state spaces can be constructed by:

- 1) Using the following CPN SML functions:  
*CalculateOccGraph( )* and *CalculateSccGraph( )*;
- 2) Or, using the CPN State Space (SS) tool palette: For more information about using the state space tools see [14]. In our approach, the user needs to open the CPNXML file using the CPN Tool and construct the state space in a manual way using the CPN state space tool palette.

VII. STEP THREE: APPLYING VERIFICATION MODEL

The third step of our approach concerns:

- 1) The use of CPN Tool to observe and dynamic simulation and execution of the CPN model of MAS protocol (this will be done by the user);
- 2) The full state space analysis by applying a semi-automated verification model.

The verification model is carried out by checking four basic properties, which are independent of any dialogue (protocol) types:

- 1) *Dialogue opening property*: to check that the LCC protocol begins with a proper *Starting message*;
- 2) *Termination of a dialogue property*: to determine if the LCC protocol terminates with a proper *Termination message*;
- 3) *Turn taking between agents property*: to guarantee that in the LCC protocol the turn-taking switches to the next agent after the current agent sends a message;
- 4) *Message sequencing property*: to check that the LCC protocol message exchange respects the gent messages expectation sequence.

In general, to verify each property, we use the following approach:

- 1) Create a new text file for each property and use the property name as the file name;
- 2) Extract the needed information from the state space graph and write this information in the property text file;

```

1. Read&Save SS=State Space information
2. Read&Save OpenDialogueMessages = information
3. Call CheckProperty1
4. Input (SS,OpenDialogueMessages)
5. Extract message1
6. val checkODM =
7. compare(OpenDialogueMessages,message1)
8. if (checkODM) then
9. "Property 1(Dialogue opening) is Satisfied"
10. else
11. "Property 1(Dialogue opening) is not Satisfied"
12. end CheckProperty1
13. Create&Save Property1 result file
    
```

Figure 2. Property 1 as an SML Function

- 3) Get the information of a the gent messages expectation from the protocol expectation property file (this could be done in a fully automatic way or in a manual way);
- 4) Call the CPN SML property function, where the function inputs are the protocol expectation property file and the LCC protocol state space information (property text file);
- 5) Create a new text file (property result file) and write the CPN SML property function result in the property result file;
- 6) Repeat steps 1 to 5 for each property;
- 7) Present a report to the user indicating which properties are satisfied and which are unsatisfied.

Space limitations prevent us from presenting each and every one of those properties instead we will discuss Property-1 Dialogue Opening.

This property should guarantee that the LCC protocol will start if, and only if, a proposal agent sends a *Starting message*. Figure 2 shows the CPN SML specification of this property:

- 1) *Line 1*: Read the state space graph information from the Property1 text file and save this information in the state space informaiton (SS) variable.
- 2) *Line 2*: Read the Starting message information of a protocol from protocol expectation property file and save this information in the *OpenDialogueMessages* variable.
- 3) *Line 3*: Call *CheckProperty1* function.
- 4) *Line 4*: *CheckProperty1* function inputs are *SS* and *OpenDialogueMessages*.
- 5) *Line 5*: Extract the first message from the *SS (message1)*
- 6) *Lines 6 and 7*: Compare the first exchanged message in the state space graph with the *Starting message* where:
  - a) compare function is used to compare the first message;

b) *checkODM* variable represents the compare function result. It is considered true if the first message in the state space graph is the same as the *Starting message*.

7) *Lines 8 to 11*: Check the result of the comparison. A positive (negative) result indicates that Property 1 is satisfied (unsatisfied).

8) *Line 13*: Create a Property1 result file and write the result of *CheckProperty1* in this file.

Our verification method identifies only four basic properties, which are general properties that may be applied to several dialogues (protocols). However, if the user needs to verify different properties, the user needs to specify these properties and feed them to the generated CPNXML file manually.

### VIII. EXAMPLE

An example of a MAS protocol is a persuasion dialogue (adapted from [15]), where a dialogue is presented as a game in which one participant (proponent 'P') attempts to persuade another participant (opponent 'O') to change their point of view about a particular topic 'T'. Our *LCC/CPNProtocol* Tool gets as input the LCC protocol of a persuasion dialogue (see Figure 3) and returns the corresponding CPN models (Space limitations prevent us from presenting each and every one of CPN models instead we will illustrate only two CPN models: *ClaimSender* in Figure 4 and *ClaimReceiver* CPN models in Figure 5) by using LCC-CPNXML tables. In practice, by using this tool, no additional programming is required.

User then can manually construct the state space of the generated CPN models using the SS tool palette in CPN Tools (see Figure 6). Then *LCC/CPNProtocol* Tool:

1) Gets agent's objectives properties expectation from user (*Figure 7 is an example of this properties where Starting Locutions file contains one message name claim, which is used to begin the persuasion dialogue*);

2) Automatically comparing the agent's objectives properties and the behavioral properties of the LCC protocol using CPN SML functions. To verify properties the following actions were performed:

a) Open the CPN model by using the CPN Tool;

b) Select the Evaluates a Text as ML Code(ML!) icon in the simulation tool palette and apply it to property page (*Figures 8 show the Dialogue opening property page after applying the ML! to it*);

3) Shows the verification result (see Figure 9).

### IX. CONCLUSION AND FUTURE WORK

In this paper, we have presented a methodology to support formal validation of MAS protocol. The main idea is to automatically transform the LCC protocol into an equivalent CPN model using a set of transformational rules

and then extracts four behavioral properties (Dialogue opening property, Termination of a dialogue property, Turn taking between agents property and Message sequencing property) of the LCC protocol from the CPN model state space. These properties can be used to check whether the protocol satisfies agents objectives and terminates correctly.

Our further work is targeted at investigating three questions: Can the user modify the available properties to suit their specific MAS protocol using our tool? Can our tool specify new properties in an automated manner? Can our tool take the new properties information from the user using a constrained form of natural language?

### REFERENCES

- [1] G. Chicoisne, "Dialogue between natural agents and artificial agents: An application to virtual communities", PhD thesis, National Institut of Polytechnique of Grenoble, pp. 71-74, 2004.
- [2] M. Koji, S. Jin-Hua, and Q. Yasuhiro, "Study on Common Coordinate System by using Relative Position of Other Autonomous Robot", SICE Annual Conference, Japan, pp. 795-797, August 20-23, 2012.
- [3] K. Jensen, "Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use", Berlin, Springer Verlag, 1997.
- [4] M. Westergaard and H. Verbeek, "*CPN Tools*" Eindhoven University of Technology, 2002, [retrieved 3, 2015], <http://cpntools.org/>.
- [5] S. Suriadi, C. Yang, J. Smith, and E. Foo, "Modeling and Verification of Privacy Enhancing Security Protocols", 11th International Conference on Formal Engineering Methods ICFEM, Janeiro, Brazi, ICFEM, pp. 127-146, 2009.
- [6] R. Milner, M. Tofte, R. Harper, and D. Macqueen, "The Definition of Standard ML" Cambridge, MA, USA, The MIT Press, revised edition, 1997.
- [7] K. Jensen and L. Kristensen, "Coloured Petri Nets Modelling and Validation of Concurrent Systems" Berlin, Springer Verlag, 2009.
- [8] K. Jensen, L. Kristensen, and L. Wells, "Coloured Petri Nets and CPN Tools for modelling and validation of concurrent systems" International Journal on Software Tools for Technology Transfer (STTT), 3rd ed., vol. 9, pp. 213-254, 2007.
- [9] M. Eunice, "Model transformation support for the analysis of large-scale systems" Texas Tech University Electronic Theses and Dissertations, Master Thesis in Software Engineering, 2005.
- [10] B. Bauer, J. Müller, and J. Odell, "Agent UML: A Formalism for Specifying Multiagent Interaction" Software Engineering and Knowledge Engineering, vol. 9, pp. 91-103, 2001.
- [11] D. Robertson, "Multi-agent coordination as distributed logic programming" In DEMOEN, BART and LIFSCHITZ, VLADIMIR, Logic programming. Saint-Malo, France, 20<sup>th</sup> International Conference, pp. 416-430, 2004.
- [12] R. Dijkman and M. Dumas, "Service-oriented Design: A Multi-viewpoint Approach" International Journal of Cooperative Information Systems, 4th ed., vol. 13, pp. 337-378, 2004.
- [13] J. Billington et al. "The Petri Net Markup Language: Concepts, Technology, and Tools" The Netherlands, 24th International Conference, ICATPN 2003 Eindhoven, 2003.
- [14] K. Jensen, S. Christensen, and L. Kristensen, "CPN Tools State Space Manual" University of Aarhus, Department of computer science, 2002, [retrieved 3, 2015].
- [15] H. Prakken, "Coherence and flexibility in dialogue games for argumentation" Journal of logic and computation, 6th ed., vol. 15, pp. 1009-1040, 2005.

| Agent P   | Agent O   |
|---|---|
| $a(\text{claimSender}_P(\text{KB}_P, \text{CS}_P, \text{CS}_O, T, \text{ID}_O), \text{ID}_P) ::=$ $\text{claim}(T) \Rightarrow$ $a(\text{claimReceiver}_O(\text{KB}_O, \text{CS}_O, \text{CS}_P, \text{ID}_P), \text{ID}_O)$ $\leftarrow \text{addTopicToCS}(T, \text{CS}_P)$ <p style="text-align: center;">then</p> $a(\text{replyToClaimReceiver}_P(\text{KB}_P, \text{CS}_P, \text{CS}_O, T, \text{ID}_O), \text{ID}_P).$ | $a(\text{claimReceiver}_O(\text{KB}_O, \text{CS}_O, \text{CS}_P, \text{ID}_P), \text{ID}_O) ::=$ $\text{claim}(T) \Leftarrow$ $a(\text{claimSender}_P(\text{KB}_P, \text{CS}_P, \text{CS}_O, T, \text{ID}_O), \text{ID}_P)$ <p style="text-align: center;">then</p> $a(\text{replyToClaimSender}_O(\text{KB}_O, \text{CS}_O, \text{CS}_P, T, \text{ID}_P), \text{ID}_O).$ |

Figure 3. Two roles of LCC Protocol for Persuasion Dialogue

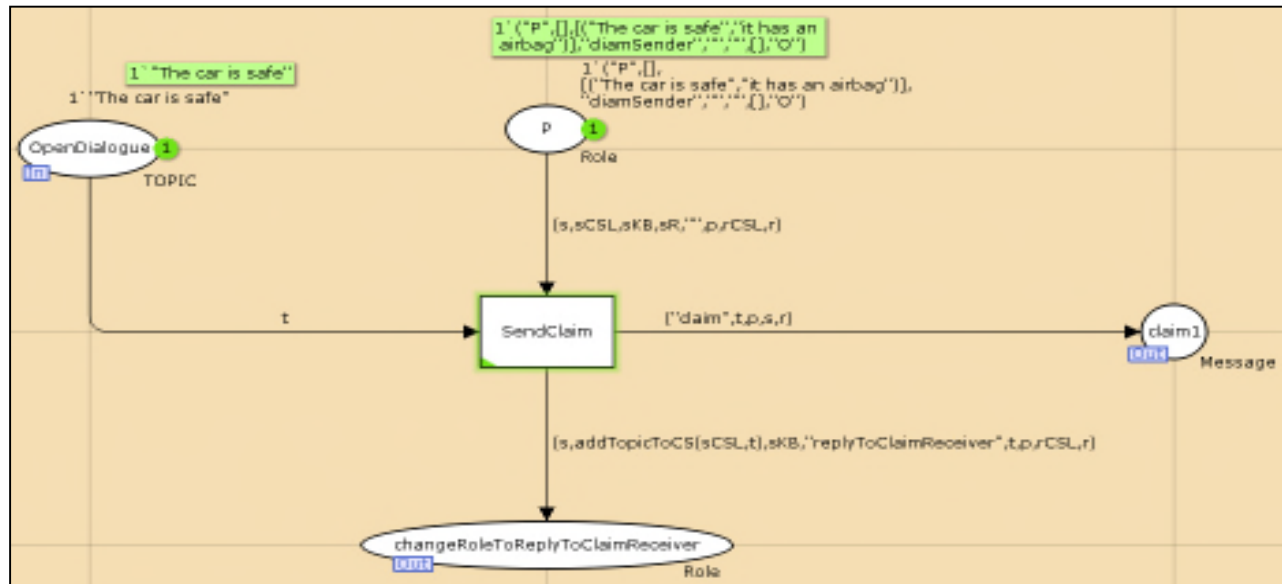


Figure 4. The claimSender<sub>P</sub> CPN Subpage

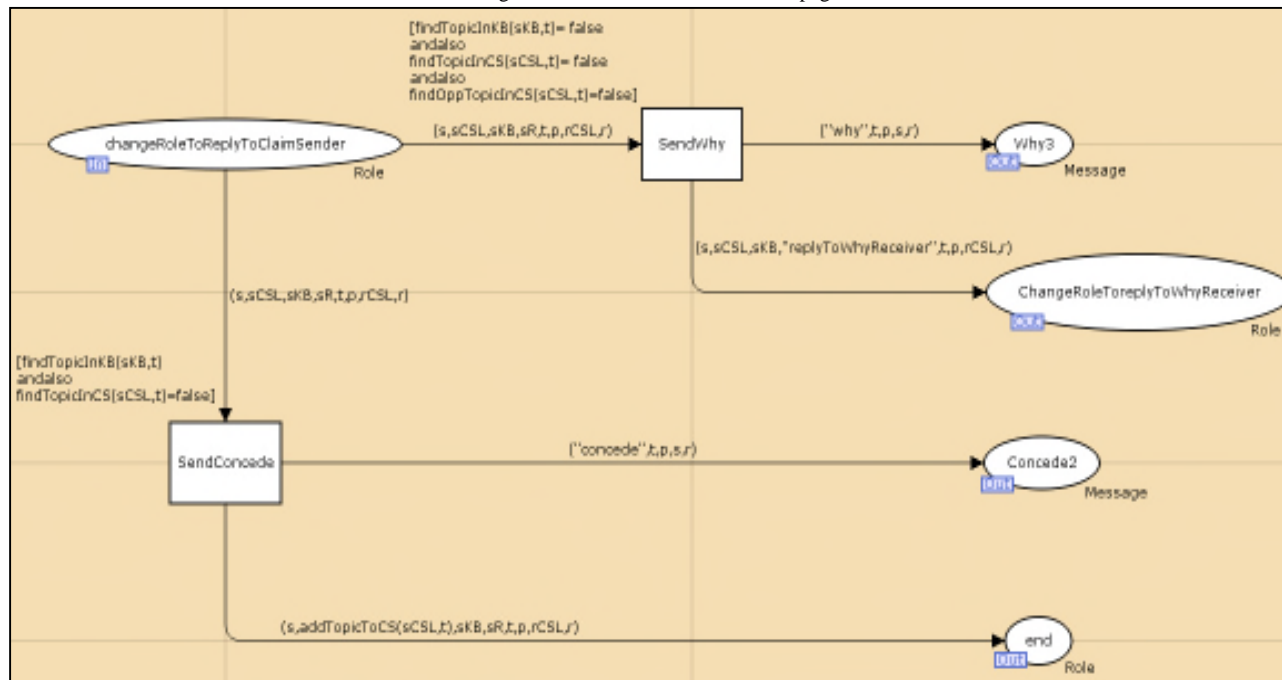


Figure 5. The claimReceiver<sub>O</sub> CPN Subpage

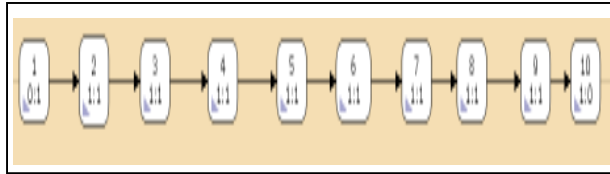


Figure 6. The State Space Graph

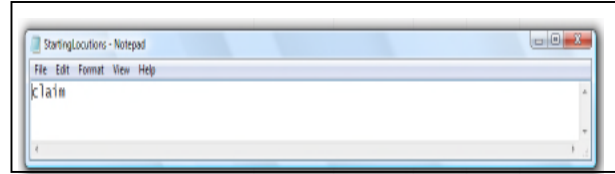


Figure 7. Starting Locutions file

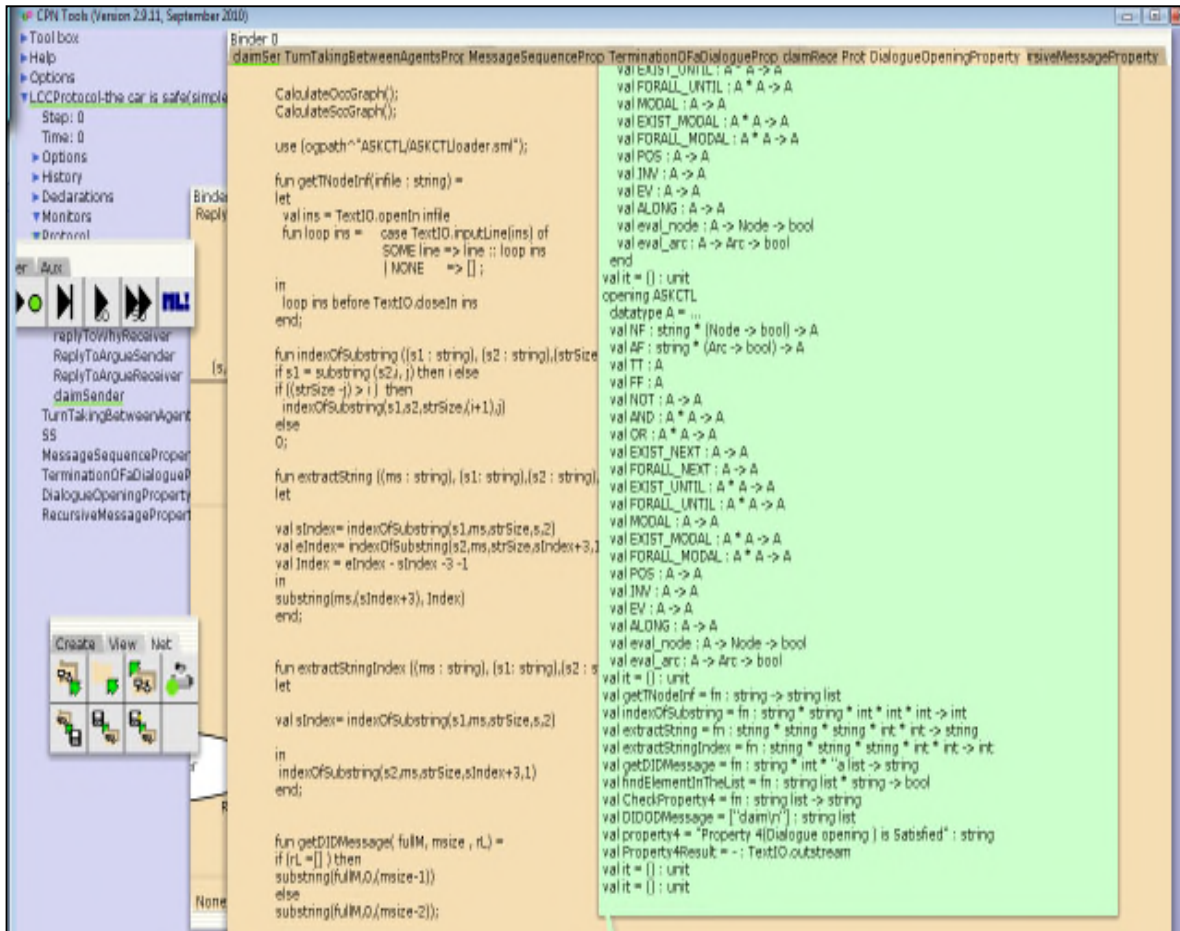


Figure 8. Dialogue Opening Property Page

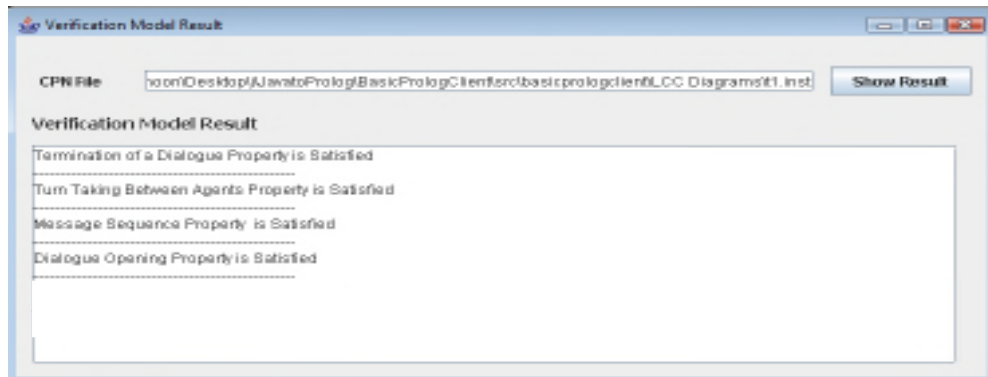


Figure 9. The Verification Result of the Five Basic Properties