Improving the Mapping of $\mathcal{M}$oise$^+$ to Colored Petri Nets

Ricardo A. Machado[1], **Diana F. Adamatti**[1], **Eder M. Gonçalves** [1]

[1] Programa de Pós-Graduação em Modelagem Computacional
Centro de Ciências Computacionais – Universidade Federal do Rio Grande (FURG)
Rio Grande – RS – Brasil

{ricardoarend,dianaada,eder.m.goncalves}@gmail.com

Abstract. *The demand for systems with artificial intelligence such as Multiagent Systems (MAS) is continuously growing. At the same time, the need for the development of tools that help software development increases, ensuring better fault tolerance for the project, since these systems have characteristics that make the system non-deterministic and increase the difficulty in carrying out tests. In previous work, an approach was presented for tests in MAS that use the organizational model $\mathcal{M}$oise$^+$, where test cases were generated using Colored Petri Nets (CPN) to map the system in a formal model. In this work, an update is presented for the inclusion of more elements contained in $\mathcal{M}$oise$^+$ that will be mapped in the CPN, more specifically, the minimum and maximum cardinality for each existing mission.*

1. Introduction

Multiagent Systems (MAS) represent an innovative paradigm for modeling, simulating, and designing complex distributed systems. Its effectiveness is due to the ability of agents to represent system entities, their behaviors, and their interactions. In fact, an agent has proactive and reactive characteristics that are very useful in decision-making. In addition, a fundamental feature of MAS is collective or distributed intelligence, and the ability to function even with the failure of one of its components (agent) without the need to interrupt the system as a whole [Boucherit et al. 2020].

Over time, MAS has gained importance in the most different areas, due to the unique characteristics of the agents, such as reactivity, proactivity, autonomy, and social capacity [Padgham and Winikoff 2005]. However, certain applications require a minimum of reliability so that the system does not present any risk to the user. One can cite as an example the control of autonomous vehicles, military applications, Smart Grids, and logistics.

To ensure that the system is reliable and can be made available to the user, software testing is necessary. Software testing consists of dynamically verifying the behavior of a program against a set of suitably selected test cases [Abran et al. 2004]. A test case is an input on which the program under test is executed while testing [Zhu et al. 1997], and testers often need to generate test cases to execute all statements in the program at least once.

Given that agents are often required to perform multiple tasks simultaneously, Petri nets are a particularly effective tool for specifying and testing the behavior of agent-based systems. Furthermore, Petri nets are one of the most reliable formalisms for simulation and analysis of behavioral modeling and MAS failure analysis [Boucherit et al. 2020].

In a previous work [Gonçalves et al. 2022], a formalization for generating test cases was described starting from an organizational model $\mathcal{M}$oise$^+$ and, mapping the model in a Colored Petri Net (CPN) to generate the test cases from two different coverage criteria. One of the limitations of this model is the fact that it does not take into account the number of agents committed to a mission.

In some tests, it was found that in $\mathcal{M}$oise$^+$ if a mission does not have the minimum number of agents indicated by the cardinality, it is not satisfied and the system ends up in a waiting state. On the other hand, if the number of agents is greater than the maximum cardinality, some agents are left out of the mission and an error message is generated, which does not prevent the other agents from completing the mission normally. To ensure that the generated CPN is as faithful as possible to the original $\mathcal{M}$oise$^+$ model and that more test cases can be generated, a proposal for updating the mapping in CPN is presented here, where the inclusion of minimum and maximum cardinality becomes necessary.

2. Related Works

As examples of related work, we can cite the work of [Frasheri et al. 2017], where an initial concept of an adaptive autonomous agent was analyzed about fault tolerance. Failure analysis is performed by modeling the agents through Colored Petri Nets. The simulation results indicate the number of tasks that are discarded or completed by the agents. Based on the experimental results, the analysis showed the correlation between the probability of system failure (increase in the number of lost tasks) and the failure of an agent.

In [Rehman et al. 2019] is presented a model-based testing approach that uses the *Prometheus* tool for MAS modeling. For the generation of test cases, some different coverage criteria defined by the authors were used. The generation of test paths is automated with the help of a tool that takes a test model as input, applies the different coverage criteria, and generates a test path for each coverage criterion.

[Dehimi and Mokhati 2019] proposes a model-based testing approach to test agent interactions. The approach uses an AUML sequence diagram as a model and constraints expressed in the Object Constraint Language (OCL). The approach generates a set of test cases capable of, individually, covering interactions between agents, as well as possible scenarios that can be executed in an inclusive, exclusive, or parallel way.

In [Boucherit et al. 2020] the authors use Petri nets and rewrite logic to facilitate the formalization of critical security applications based on multi-agent. They present an algorithm that allows the automatic generation of Maude specifications for models of multi-agent systems based on Petri nets. In addition, model verification and property-based testing techniques are integrated into the verification and testing phases.

3. Theoretical Background

This work demonstrates how a MAS with an organizational model can be mapped in a CPN. For this, the model $\mathcal{M}$oise$^+$ was chosen, which contains an independent Structural Specification (SS) and a Functional Specification (FE), linked by a Deontic Specification (DS) [Hübner and Sichman 2007]. The $\mathcal{M}$oise$^+$ is described using notions such as groups and roles and the links between them in the structural dimension. The functioning of the system is described by global goals that must be achieved and their missions in the

functional dimension. Finally, the deontic dimension unites the structural and functional aspects, defining permissions and obligations for each function and mission.

According [Boissier et al. 2020] in $\mathcal{M}$oise$^+$ there are two basic types of cardinality constraint: a *role cardinality* that defines upper and lower bounds on the number of agents that can play the role in the corresponding group entity and a *mission cardinality* that constrains the minimum and maximum number of agents that can commit to a mission. In this work, the focus is on mission cardinality.

A Petri Net is a graphical and mathematical modeling tool applicable to many systems. They are used as visual communication aid similar to flowcharts. Petri Nets are divided into two classes: low-level and high-level.

In high-level Petri nets, we have Colored Petri Nets (CPN) as an example, which combine the capabilities of an ordinary Petri Net with the capabilities of a programming language. In CPN, tokens have a data value attached called color. These colors can represent different processes or resources of a network, thus reducing the size of the models. For a given place, all tokens must have colors that belong to a specific type which is called the place color set. This use of color sets is equivalent to data types in programming languages, and the specification of color sets and network variables is done by declarations [Jensen 1997].

CPN composition contains a structure, inscriptions, and declarations. It is a directed graph with two types of vertices: (i) circles, or ellipses represent the places, and (ii) the transitions by rectangles. Each place, transition, and arc of the network has associated inscriptions. For example, places have three types of inscriptions: names, color sets, and initialization expression, which is the initial mark. The declarations are the specifications of the color sets and the declaration of the variables and functions.

4. Mapping the $\mathcal{M}$oise$^+$ Model to a CPN

This section will present all the tree steps for the mapping a $\mathcal{M}$oise$^+$ model in a CPN, namely: (i) Generate the Declarations, (ii) Model the Structure and (iii) Include the Inscriptions. Each of these steps is detailed below.

4.1. Generating Declarations

The first step in mapping is generating the declarations. These declarations play an important role in the functioning of the CPN. The first declaration is the color set (*colset*) *R* that represents the existing roles in the $\mathcal{M}$oise$^+$ specification. Here, as an example, we have two roles, *roleA* and *roleB*, which are separated by a vertical bar. Next, we have the *G* color set that represents the groups of the specification, here being *groupA* and *groupB* and below the *RG* color set that represents the product between the two previous sets in order to relate each role with a respective group.

Continuing using *var*, the variables are declared. For each role on the network, it is necessary to create a new variable, in this example the variable r1 is used for the role *roleA* and the variable *r2* for the role *roleB*, both of type *RG*, and *b* for a boolean type variable. The constants (*val*) are declared in sequence, *NroleA* and *Nrole*B being used to indicate the number of agents with *roleA* and *roleB* there are, and the constants referring to the minimum and maximum cardinalities for each mission are also declared.

Finally, in the declarations, we have the functions that are used in the arcs. Here as an example are two functions (*Fmission1* and *Fmission2*) because for each mission a related function must be created. The function code is always the same, changing only the variables and constants used. The purpose of these functions is to ensure that if the maximum cardinality is reached in a number of agents for a given role, it is not exceeded, so even with more agents available, the mission only uses the number informed in the maximum cardinality. On the other hand, if the number of agents is less than or equal to the maximum cardinality, this number will be used in the mission.

```
Declarations:
colset R = with roleA | roleB;
colset G = with groupA | groupB;
colset RG = product R*G;
var r1,r2 : RG;
var b : BOOL;
val NroleA = 5;
val NroleB = 4;
val Cmin_mission1 = 2;
val Cmax_mission1 = 4;
val Cmin_mission2 = 1;
val Cmax_mission2 = 2;
fun Fmission1(r1)=if NroleA <= Cmax_mission1 then NroleA`r1
                              else Cmax_mission1`r1;
fun Fmission2(r2)=if NroleB <= Cmax_mission2 then NroleB`r2
                              else Cmax_mission2`r2;
```

4.2. Modeling the Structure

In the structure modeling, the $\mathcal{M}$oise$^+$ goals are represented by the CPN places, and the different types of $\mathcal{M}$oise$^+$ plan operators (sequence, choice and parallelism) are transformed into net structures through the mapping model presented in Figure 1 [Gonçalves et al. 2022] thus defining the way places are distributed on the network. In addition to these elements, an initial place called *start* and a final place called *end* are included in the network, indicating where the system starts and ends, as well as a place that represents the agents that are in a waiting state named *Waiting*.

If there are different missions in $\mathcal{M}$oise$^+$, it is also necessary to include auxiliary places in the network called *aux_n*, where *n* represents a numerical sequence to differentiate each of these places.

In Figure 2 an example of a structure is presented that represents a sequence between two goals *goal1* and *goal2* and each of these goals belongs to a mission, with *goal1* belonging to *mission1* and *goal2* to *mission2*. As there is this alternation between missions, an auxiliary place aux_1 is also included between *goal1* and *goal2*. The inclusion of the names for the places is considered as part of the step of including inscriptions and not of the modeling of the structure, in Figure 2 these names were included for a better understanding of the elements exposed here.

4.3. Including the Inscriptions

Once the structure of the CPN is finished, the stage of the inclusion of inscriptions begins. In this step, the names of the places, their initial markings, and their type are included, as well as the names of the transitions, functions, and arc variables. Figure 3 shows the same example with all inscriptions. In the *Waiting* place there are tokens that represent the available agents waiting their turn to satisfy the goals of their missions and an initial

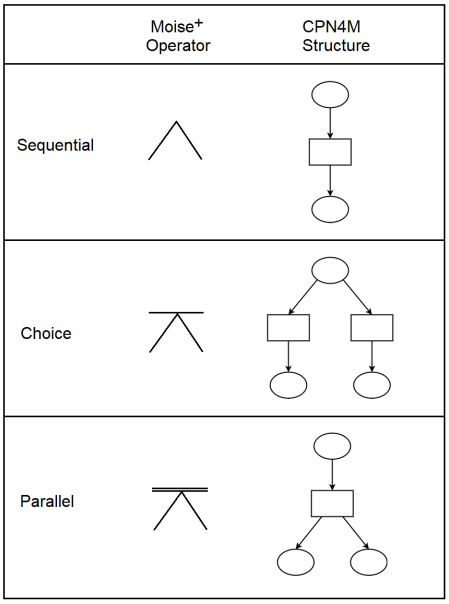

Figura 1. A Mapping between FS operators and CPN structures [Gonçalves et al. 2022].

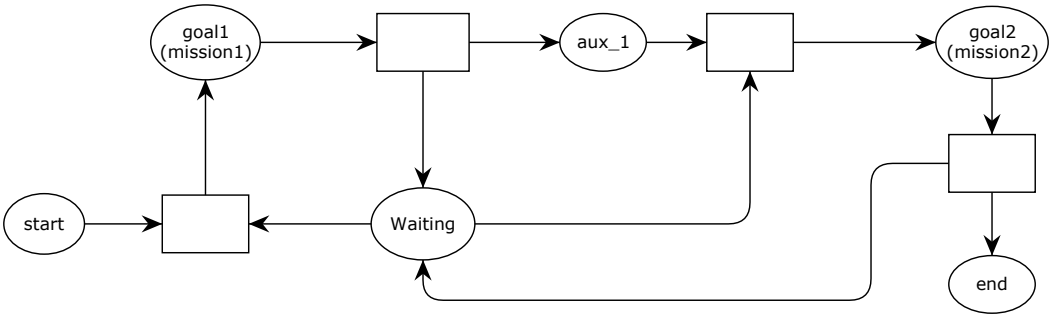

Figura 2. Structure of a Generic System mapped in CPN

marking is made with *NroleA'(roleA,groupA)* where *NroleA* is the number of *roleA* agents belonging to the group *groupA* , previously defined in the declarations.

The places that represent the goals and the *Waiting* place receive the color type *RG* that represents the set of colors of the roles combined with that of the groups. The initial, final and auxiliary places are of type *BOOL* (boolean) that receive a token named *true* as input, and as output the boolean variable *b*.

Functions calls are used on the entry and exit arcs of places with the type *RG*, for example between place *Waiting*, the transition *t1* and place *goal1* we have on the arcs the inscription *Fmission1(r1)* which is the call to a function that indicates the number of agents assigned to the mission *mission1*. In turn, *goal2* already belongs to *mission2*, and therefore the function *Fmission2(r2)* is used on the arcs.

Guard functions in the *t1* and *t3* transitions are used to ensure that the number of agents satisfies the minimum cardinality for each mission and also to define the type of role that will be used. For example, *t1* reads *NroleA* to be greater than or equal to *Cmin_mission1*, and that the variable *r1* contains *(roleA,groupA)*. It is only necessary to

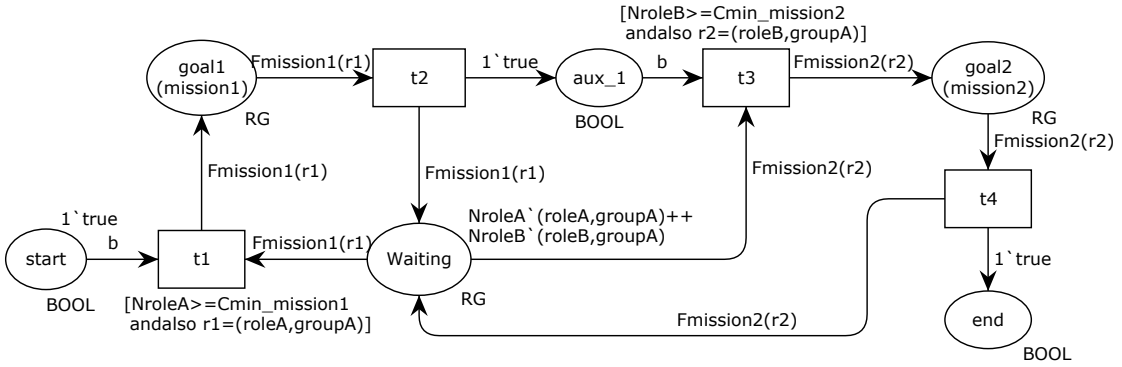

Figura 3. Inscriptions included in the CPN

use this type of guard function in transitions that receive as input an arc that came out of *Waiting*

After this stage, it is already possible to carry out simulations, using the generated model to identify the different test scenarios that the system is capable of executing. In the following section, a use case for this method will be presented, showing both a $\mathcal{M}$oise$^+$ model in detail and its respective mapping in an CPN.

5. Writing Paper Implementation Example

The scenario application used is the *Writing Paper* [Hübner et al. 2011]. It describes an agent group that has a goal to write a paper to be published. Figure 4 describes the structural specification composed by a *wpgroup* group that has two roles: *writer* and *editor*, and both roles are sub-roles from *author*.

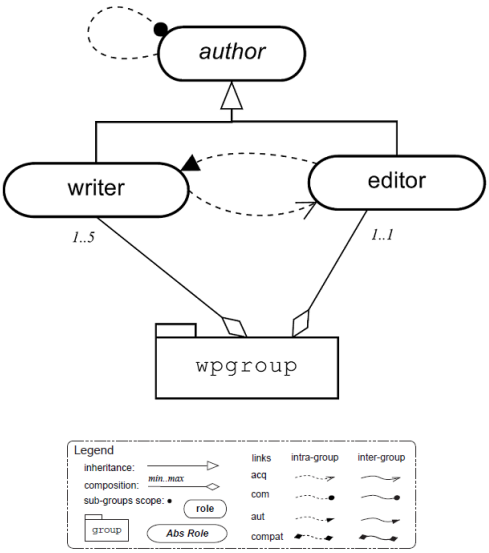

Figura 4. Structural Specification for a Writing Paper scenario, based on [Hübner et al. 2011].

The functional specification for this scenario is presented in Figure 5. According to this scheme, the first part is to conclude a paper draft (*fds*). Obeying a sequence, an

agent undertakes the *mMan* mission and must write a title (*wtitle*), an abstract (*wabs*), and write the section titles (*wsectitles*) ending this first version. The second branching, named (*sv*), the submission version is composed of the goals (*wsecs*), writing sections, and to finish the paper, it is necessary to reach two goals in parallel, (*wcon*) writing a conclusion and (*wrefs*) writing references.

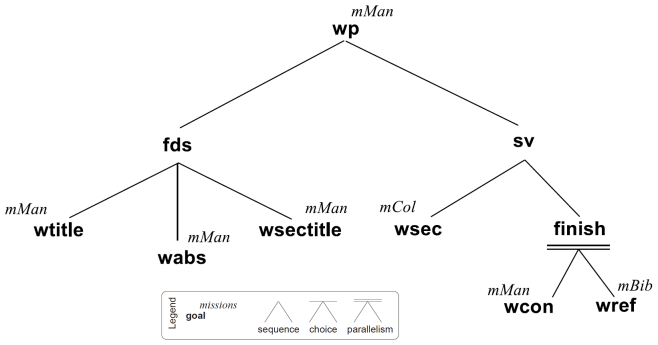

Figura 5. Functional Specification for a Writing Paper scenario, adapted from [Hübner et al. 2011].

Table 1 describes the Deontic Specification defining permissions and obligations for the roles that undertake the missions. The missions are *mMan*, project general managing; *mCol* collaborating for the writing process; and mission *mBib* for the agent which must gather and write the paper references. Here, the minimum and maximum cardinalities for each mission were also included.

Role	Deontic Relationship	Cmin	Cmax	Mission
editor	obligation	1	2	*mMan*
writer	permission	2	5	*mCol*
writer	obligation	3	3	*mBib*

Tabela 1. Deontic Specification for a Writing Paper scenario, based on [Hübner et al. 2011].

Now with the necessary information from each specification presented, will present the mapping using the sequence of steps defined in the previous section.

5.1. Generating Declarations

In the first stage and based on the $\mathcal{M}$oise$^+$ specifications presented above, the following elements are declared: The *editor* and *writer* roles, the *wpgroup* group, two variables *r1* and *r2*, and a third variable *b*, the constants for the number of agents in each role and for cardinality, and finally the functions, one for each mission:

```
Declarations:
colset R = with editor | writer;
colset G = with wpgroup;
colset RG = product R*G;
var r1,r2 : RG;
var b : BOOL;
val Neditor = 3;
val Nwriter = 4;
val Cmin_mMan = 1;
val Cmax_mMan = 2;
```

```
val Cmin_mCol = 2;
val Cmax_mCol = 5;
val Cmin_mBib = 3;
val Cmax_mBib = 3;
fun Fman(r1)=if Neditor <= Cmax_mMan then Neditor`r1 else Cmax_mMan`r1;
fun Fcol(r2)= if Nwriter <= Cmax_mCol then Nwriter`r2 else Cmax_mCol`r2;
fun Fbib(r2)= if Nwriter<=Cmax_mBib then Nwriter`r2 else Cmax_mBib`r2;
```

5.2. Modeling the Structure

With the statements done, now we present the second stage where the structure of the net is molded. Based on Figure 5 where we have the FE with a goal decomposition tree, we selected all the goals contained in a mission: *title*, *wabs*, *wsectitle*, *wcon* and *wp* belonging to the *mMan* mission; *wsec* to the *mCol* mission and finally *wref* to the *mBib* mission.

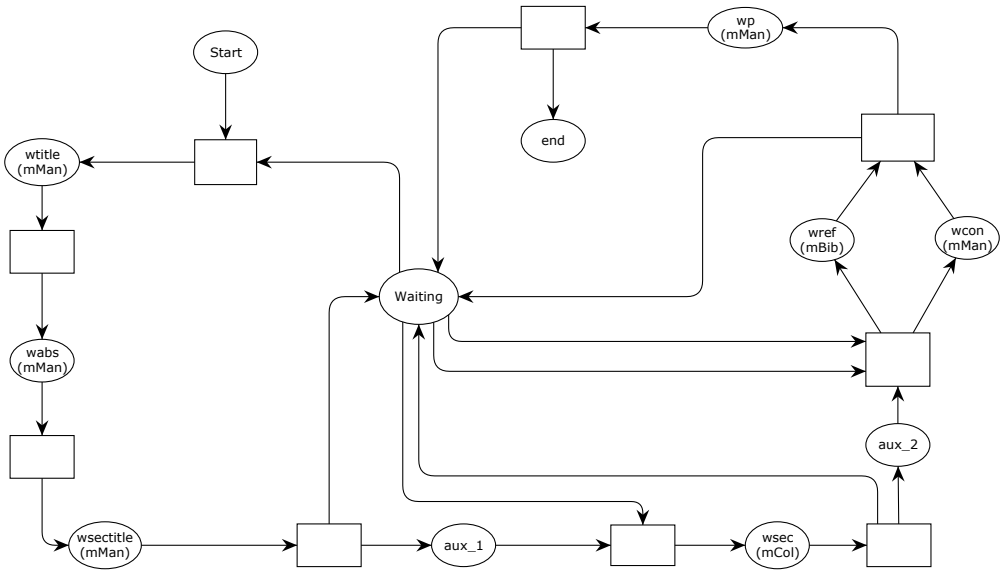

Figura 6. The Writing Paper Structure of a CPN

Figure 6 presents the structure of the CPN, where we first have a sequence of the first three goals belonging to *mMan* (*wtitle*, *wabs* and *wsectitle*). The next goal (*wsec*) also comes in sequence, but as it belongs to another mission an auxiliary place aux_1 is used. Following it we have a second auxiliary aux_2 because here the *mCol* mission ends and the net goes on to a parallelism between *wref* and *wcon* goals which belong to *mBib* and *mMan* missions respectively. As the *mBib* mission is done, a connection path to Waiting is necessary, but without the need for an auxiliary place since the net continues with the same *mMan* mission to the *wp* goal. Finally, a transition to the end place is made.

5.3. Including the Inscriptions

With the structure in place, the last step consists of including the inscriptions. Figure 7 shows the mapping in CPN ready with all inscriptions where guard functions were included in transitions *t1*, *t5* and *t7* that receive arcs from *Waiting*. The arc function calls and some expressions were also included as well as the color types of the places (*RG* and *BOOL*). In *Waiting* and *start*, initial markings are also placed.

With the CPN fully functional, it is now possible to carry out simulations by changing the number of agents available in the declarations *Neditor* and *Nwriter* and generating new test scenarios.

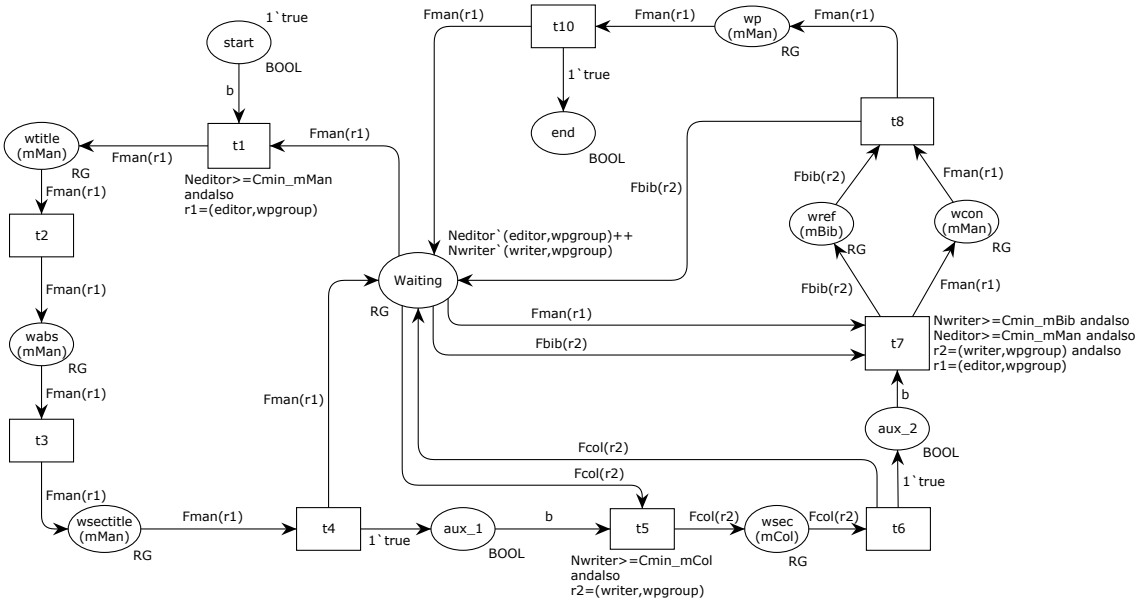

Figura 7. The Writing Paper Scenario Fully Mapped in a CPN

6. Conclusion and Future Work

In this work, a reformulation proposal was presented for the mapping method of a MAS that has the $\mathcal{M}$oise$^+$ organizational model for a CPN proposed in previous work. The purpose of this reformulation is to include elements such as cardinality, and the number of existing agents for a given role, thus generating a CPN model that is more faithful to the original system.

In future works, we have the possibility of using the method in a tool that automates the mapping, generating all the CPN elements from the $\mathcal{M}$oise$^+$ XML file. Another issue is to evaluate the need to include more elements of $\mathcal{M}$oise$^+$ so that the CPN can be used as a test tool that generates the possible test paths necessary for verifying the system. Finally, the formalization for this method will be reviewed, now taking into account the new elements included in this paper.

Referências

Abran, A., Moore, J. W., Bourque, P., Dupuis, R., and Tripp, L. (2004). Software engineering body of knowledge. *IEEE Computer Society, Angela Burgess*.

Boissier, O., Bordini, R. H., Hubner, J., and Ricci, A. (2020). *Multi-agent oriented programming: programming multi-agent systems using JaCaMo*. Mit Press.

Boucherit, A., Castro, L. M., Khababa, A., and Hasan, O. (2020). Petri net and rewriting logic based formal analysis of multi-agent based safety-critical systems. *Multiagent and Grid Systems*, 16(1):47–66.

Dehimi, N. E. H. and Mokhati, F. (2019). A novel test case generation approach based on auml sequence diagram. In *2019 International Conference on Networking and Advanced Systems (ICNAS)*, pages 1–4. IEEE.

Frasheri, M., Trinh, L. A., Cürüklü, B., and Ekström, M. (2017). Failure analysis for adaptive autonomous agents using petri nets. In *2017 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 293–297. IEEE.

Gonçalves, E. M. N., Machado, R. A., Rodrigues, B. C., and Adamatti, D. (2022). Cpn4m: Testing multi-agent systems under organizational model m oise+ using colored petri nets. *Applied Sciences*, 12(12):5857.

Hübner, J. F., Boissier, O., and Bordini, R. H. (2011). A normative programming language for multi-agent organisations. *Annals of Mathematics and Artificial Intelligence*, 62(1-2):27–53.

Hübner, J. and Sichman, J. (2007). Developing organised multi-agent systems using the $Moise^+$ model: Programming issues at the system and agent levels. In *Int. J. Accounting, Auditing and Performance Evaluation*, pages 1–10.

Jensen, K. (1997). *Coloured Petri nets: basic concepts, analysis methods and practical use*, volume 1. Springer Science & Business Media.

Padgham, L. and Winikoff, M. (2005). *Developing intelligent agent systems: A practical guide*, volume 13. John Wiley & Sons.

Rehman, S. U., Nadeem, A., and Sindhu, M. (2019). Towards automated testing of multi-agent systems using prometheus design models. *Int. Arab J. Inf. Technol*, 16(1):54–65.

Zhu, H., Hall, P. A., and May, J. H. (1997). Software unit test coverage and adequacy. *Acm computing surveys (csur)*, 29(4):366–427.